Colt Thomas

Lab 4 Source Code

Oct 14, 2014

ECEn 330

Index of Code Files

1. clockMain.c
2. clockControl.h
3. clockControl.c
4. clockDisplay.h
5. clockDisplay.c

Pertinent notes

All files except clockMain.c were written by me. The mentioned files were provided to us by professor Hutchings. These files were also tested on the ZYBO board that is provided for the BYU ECEN 330 embedded programming class and have proved to be functional for those specific boards. The state machine design was also provided by professor Hutchings, with slight modification for timing purposes. This code compiled correctly should implement a clock that will increment automatically by the second. The hour, minute and second components can also be modified to change the time on the display. The code text size below may have been scaled down to conserve the number of pages.

clockMain.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"

**#define** TOTAL\_SECONDS 60

// The formula for computing the load value is based upon the formula from 4.1.1 (calculating timer intervals)

// in the Cortex-A9 MPCore Technical Reference Manual 4-2.

// Assuming that the prescaler = 0, the formula for computing the load value based upon the desired period is:

// load-value = (period \* timer-clock) - 1

**#define** TIMER\_PERIOD 10.0E-3

**#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**()

{

// 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.

clockDisplay\_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** ( 1/\*interrupts\_isrInvocationCount() < (TOTAL\_SECONDS \* privateTimerTicksPerSecond)\*/) {

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

// Count ticks.

personalInterruptCount++;

clockControl\_tick();

interrupts\_isrFlagGlobal = 0;

}

}

interrupts\_disableArmInts();

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

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

**return** 0;

}

clockControl.h

/\*

\* clockControl.h

\*

\* Created on: Oct 9 , 2014

\* Author: coltmt

\*/

**void** **clockControl\_tick**(); // Calls the clock state function

clockControl.c

/\*

\* clockControl.c

\*

\* Created on: Oct 9 , 2014

\* Author: coltmt

\*/

**#include** "clockControl.h"

**#include** "clockDisplay.h"

**#include** "stdio.h"

**#include** "supportFiles/display.h"

**#include** "supportFiles/utils.h"

**#define** CLKCTRL\_IMAX 100

**#define** CLKCTRL\_AUTOTIMER\_INC 500

**#define** CLKCTRL\_EXPIRED\_VALUE 5000

**enum** ClkStates {*init\_st* ,*touch\_start\_st*, *waiting\_for\_touch\_st* , *ad\_timer\_running\_st*, *auto\_timer\_running\_st*, *rate\_timer\_running\_st*, *rate\_timer\_expired\_st*} ClkState;

/\*

\* Timer initialization values; these are set to allow the

\* timer to auto-increment at a rate of 10x per second after holding

\* for .5 seconds.

\*/

uint32\_t adTimer;

uint32\_t rateTimer;

uint32\_t autoTimer;

// increment counter for function clockDisplay\_advanceTimeOneSecond()

uint8\_t i = 0;

**void** **clockControl\_tick**() {

/\*

\* Below is the implementation of our state machine ClkStates.

\* Note that there is an added state called touch\_start\_st. This

\* is added in addition to the init state to await the initial

\* touch of the display before incrementing.

\*

\* Another functionality that I added was in the waiting\_for\_touch\_st.

\* A counter i increments to 100 before calling the advanceTimeOneSecond()

\* function. This allows the timer to increment every second.

\*

\* See the lab 4 description for an illustration of the state machine

\*/

**switch**(ClkState) {

// All states are explicitly shown in the case statements

// Assigning ClkState indicates a state change

**case**(*init\_st*):

ClkState = *touch\_start\_st*;

**break**;

**case**(*touch\_start\_st*): //Added state waits for first touch

// Transitions when display is touched

**if** (display\_isTouched()) {

ClkState = *waiting\_for\_touch\_st*;

}

**break**;

**case**(*waiting\_for\_touch\_st*):

// All timers reset to zero, and i increments

adTimer = 0;

autoTimer = 0;

rateTimer = 0;

i++;

// This added if statement waits 1 second to increment using counter i

**if** ( i == CLKCTRL\_IMAX) {

clockDisplay\_advanceTimeOneSecond();

i=0;

}

// Transitions when display is touched

**if**(display\_isTouched()) {

display\_clearOldTouchData();

ClkState = *ad\_timer\_running\_st*;

}

**break**;

**case**(*ad\_timer\_running\_st*):

// adTimer incremented as Moore action

adTimer = adTimer + CLKCTRL\_EXPIRED\_VALUE;

// If user continues holding display, transiton auto increment state

**if** (display\_isTouched() && adTimer == CLKCTRL\_EXPIRED\_VALUE) {

ClkState = *auto\_timer\_running\_st*;

}

// If display tapped, one increment/decrement will occur as Mealy

**else** **if** (!display\_isTouched() && adTimer == CLKCTRL\_EXPIRED\_VALUE) {

clockDisplay\_performIncDec();

ClkState = *waiting\_for\_touch\_st*;

}

**break**;

**case**(*auto\_timer\_running\_st*):

// autoTimer incremented

autoTimer = autoTimer + CLKCTRL\_AUTOTIMER\_INC;

// Increment ceases as display released

**if** (!display\_isTouched()) {

clockDisplay\_performIncDec();

ClkState = *waiting\_for\_touch\_st*;

}

// Waits for .5 seconds before auto inc/dec state transitions

**else** **if** (display\_isTouched() && autoTimer == CLKCTRL\_EXPIRED\_VALUE) {

clockDisplay\_performIncDec();

ClkState = *rate\_timer\_running\_st*;

}

**break**;

**case**(*rate\_timer\_running\_st*):

// Controls the auto increment timing to 10x a second

rateTimer = rateTimer + CLKCTRL\_AUTOTIMER\_INC ;

// Stops increment as display is released

**if** (!display\_isTouched()) {

clockDisplay\_performIncDec();

ClkState = *waiting\_for\_touch\_st*;

}

// Loop that continues increment

**else** **if** (display\_isTouched() && rateTimer == CLKCTRL\_EXPIRED\_VALUE) {

ClkState = *rate\_timer\_expired\_st*;

}

**break**;

**case**(*rate\_timer\_expired\_st*):

// rateTimer reset to control transition rate

rateTimer = 0;

// Loop that continues increment

**if** (display\_isTouched()) {

clockDisplay\_performIncDec();

ClkState = *rate\_timer\_running\_st*;

}

// Stops increment as display is released

**else** **if** (!display\_isTouched()) {

ClkState = *waiting\_for\_touch\_st*;

}

**break**;

**default**:

**break**;

}

}

clockDisplay.h

/\*

\* clockDisplay.h

\*

\* Created on: Oct 9 , 2014

\* Author: coltmt

\*/

**#include** <stdbool.h>

**void** **clockDisplay\_init**(); // Called only once - performs any necessary inits.

**void** **clockDisplay\_updateTimeDisplay**(**bool** forceUpdateAll); // Updates the time display with latest time.

**void** **clockDisplay\_performIncDec**(); // Performs the increment or decrement, depending upon the touched region.

**void** **clockDisplay\_advanceTimeOneSecond**(); // Advances the time forward by 1 second.

**void** **clockDisplay\_runTest**(); // Run a test of clock-display functions.

clockDisplay.c

/\*

\* clockDisplay.c

\*

\* Created on: Oct 9 , 2014

\* Author: coltmt

\*/

**#include** "clockDisplay.h"

**#include** "stdio.h"

**#include** "supportFiles/display.h"

**#include** "supportFiles/utils.h"

// Used colors

**#define** CLOCKDISPLAY\_GREEN 0x07E0

**#define** CLOCKDISPLAY\_\_BLACK 0x0000

**#define** CLOCKDISPLAY\_YELLOW 0xFFE0

// Display constants for chars and triangle setup

**#define** CLOCKDISPLAY\_DISP\_TXT\_SIZE 4

**#define** CLOCKDISPLAY\_DISP\_X\_MAX 320

**#define** CLOCKDISPLAY\_DISP\_Y\_MAX 240

**#define** CLOCKDISPLAY\_DISP\_X\_ORIGIN 160

**#define** CLOCKDISPLAY\_DISP\_Y\_ORIGIN 120

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_0 0

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_1 1

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_2 2

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_3 3

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_4 4

**#define** CLOCKDISPLAY\_DISP\_TRIANGLE\_5 5

**#define** CLOCKDISPLAY\_CHAR\_MOD 3

**#define** CLOCKDISPLAY\_CHAR\_WIDTH 6

**#define** CLOCKDISPLAY\_CHAR\_HEIGHT 8

**#define** CLOCKDISPLAY\_CHAR\_0 0

**#define** CLOCKDISPLAY\_CHAR\_1 1

**#define** CLOCKDISPLAY\_CHAR\_2 2

**#define** CLOCKDISPLAY\_CHAR\_3 3

**#define** CLOCKDISPLAY\_CHAR\_4 4

**#define** CLOCKDISPLAY\_CHAR\_5 5

**#define** CLOCKDISPLAY\_CHAR\_6 6

**#define** CLOCKDISPLAY\_CHAR\_7 7

// Constants that handle touchscreen return values

**#define** CLOCKDISPLAY\_DISP\_TOUCH\_REGION\_X 3\*CLOCKDISPLAY\_CHAR\_WIDTH\*CLOCKDISPLAY\_DISP\_TXT\_SIZE

**#define** CLOCKDISPLAY\_DISP\_TOUCH\_REGION\_Y 2\*CLOCKDISPLAY\_CHAR\_HEIGHT\*CLOCKDISPLAY\_DISP\_TXT\_SIZE

**#define** CLOCKDISPLAY\_X\_CURSOR CLOCKDISPLAY\_DISP\_X\_ORIGIN - 4 \* CLOCKDISPLAY\_CHAR\_WIDTH \* CLOCKDISPLAY\_DISP\_TXT\_SIZE

**#define** CLOCKDISPLAY\_Y\_CURSOR CLOCKDISPLAY\_DISP\_Y\_ORIGIN - (CLOCKDISPLAY\_CHAR\_HEIGHT\* CLOCKDISPLAY\_DISP\_TXT\_SIZE)/2

// RunTest constants

**#define** CLOCKDISPLAY\_RUNTEST\_CONST 100

**#define** CLOCKDISPLAY\_ROLL\_CONST1 65

**#define** CLOCKDISPLAY\_ROLL\_CONST2 50

/\*

\* These are the display chars that will be updated by multiple functions

\* They will display in format (Hr1 Hr0) : (Min1 Min0) : (Sec1 Sec0)

\*/

**char** Hr0 = '2';

**char** Hr1 = '1';

**char** Min0 = '9';

**char** Min1 = '5';

**char** Sec0 = '9';

**char** Sec1 = '5';

**char** Colon = ':';

// Upper row of triangle touch buttons and coordinates

**static** uint32\_t X0 = CLOCKDISPLAY\_DISP\_X\_ORIGIN - CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH;

**static** uint32\_t Y0 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN - (CLOCKDISPLAY\_CHAR\_HEIGHT\*CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**static** uint32\_t X1 = CLOCKDISPLAY\_DISP\_X\_ORIGIN + CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH;

**static** uint32\_t Y1 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN - (CLOCKDISPLAY\_CHAR\_HEIGHT\*CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**static** uint32\_t X2 = CLOCKDISPLAY\_DISP\_X\_ORIGIN;

**static** uint32\_t Y2 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN - (CLOCKDISPLAY\_CHAR\_HEIGHT \* CLOCKDISPLAY\_DISP\_TXT\_SIZE) - CLOCKDISPLAY\_CHAR\_WIDTH \* CLOCKDISPLAY\_DISP\_TXT\_SIZE;

// Lower row of triangle touch buttons and coordinates

**static** uint32\_t X3 = CLOCKDISPLAY\_DISP\_X\_ORIGIN + CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH;

**static** uint32\_t Y3 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN + (CLOCKDISPLAY\_CHAR\_HEIGHT\*CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**static** uint32\_t X4 = CLOCKDISPLAY\_DISP\_X\_ORIGIN - CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH;

**static** uint32\_t Y4 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN + (CLOCKDISPLAY\_CHAR\_HEIGHT\*CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**static** uint32\_t X5 = CLOCKDISPLAY\_DISP\_X\_ORIGIN;

**static** uint32\_t Y5 = CLOCKDISPLAY\_DISP\_Y\_ORIGIN + (CLOCKDISPLAY\_CHAR\_HEIGHT \* CLOCKDISPLAY\_DISP\_TXT\_SIZE) + CLOCKDISPLAY\_CHAR\_WIDTH \* CLOCKDISPLAY\_DISP\_TXT\_SIZE;

// Gives x coordinate for passed char place

uint32\_t **clockDisplay\_x\_char** (uint32\_t char\_place) {

**return** CLOCKDISPLAY\_X\_CURSOR + CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH \* char\_place;

}

// Gives y coordinate for passed char place

uint32\_t **clockDisplay\_y\_char** (uint32\_t char\_place) {

**return** CLOCKDISPLAY\_Y\_CURSOR;

}

// Displays triangle based on passed color and triangle position

**void** **clockDisplay\_Triangle**(uint8\_t num , uint32\_t color) {

**switch** (num) {

**case** 0 :

display\_fillTriangle(X0 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y0,X1 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y1,X2 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y2,color);

**break**;

**case** 1 :

display\_fillTriangle(X0,Y0,X1,Y1,X2,Y2,color);

**break**;

**case** 2:

display\_fillTriangle(X0 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y0,X1 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y1,X2 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y2,color);

**break**;

**case** 3 :

display\_fillTriangle(X3 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y3,X4 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y4,X5 - CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y5,color);

**break**;

**case** 4 :

display\_fillTriangle(X3,Y3,X4,Y4,X5,Y5,color);

**break**;

**case** 5:

display\_fillTriangle(X3 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y3,X4 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y4,X5 + CLOCKDISPLAY\_CHAR\_MOD \* CLOCKDISPLAY\_DISP\_TXT\_SIZE \* CLOCKDISPLAY\_CHAR\_WIDTH,Y5,color);

**break**;

**default**:

**printf**("error; invalid triangle access");

}

}

// Clears the clock

**void** **clockDisplay\_Clr\_Time**() {

Hr0 = '2';

Hr1 = '1';

Min0 = '0';

Min1 = '0';

Sec0 = '0';

Sec1 = '0';

}

// Increments the seconds with appropriate rollover

**void** **clockDisplay\_IncSec**(){

**if** (Sec0 == '9') {

Sec0 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**if** (Sec1 == '5') {

Sec1 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_6),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Sec1++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_6),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** {

Sec0++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

// Decrements the seconds with appropriate rollover

**void** **clockDisplay\_DecSec**(){

**if** (Sec0 == '0') {

**if** (Sec1 == '0') {

Sec0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

Sec1 = '5';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_6),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Sec1--;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_6),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

Sec0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** {

Sec0--;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

// Increments the minutes with appropriate rollover

**void** **clockDisplay\_IncMin**(){

**if** (Min0 == '9') {

Min0 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

**if** (Min1 == '5') {

Min1 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_3),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Min1++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_3),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** {

Min0++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

// Decrements the minutes with appropriate rollover

**void** **clockDisplay\_DecMin**(){

**if** (Min0 == '0') {

**if** (Min1 == '0') {

Min0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

Min1 = '5';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_3),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Min1--;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_3),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

Min0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** {

Min0--;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

// Increments the hours with appropriate rollover

**void** **clockDisplay\_IncHr**(){

**if** (Hr1 == '1') {

**if** (Hr0 == '2') {

Hr1 = ' ';

Hr0 = '1';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Hr0++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** {

**if** (Hr0 == '9') {

**if** (Hr1 == ' '){

Hr1 = '0';

}

Hr1++;

Hr0 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Hr0++;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

}

// Decrements the hours with appropriate rollover

**void** **clockDisplay\_DecHr**(){

**if** (Hr0 == '1') {

**if** (Hr1 == ' ') {

Hr0 = '2';

Hr1 = '1';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** **if** (Hr1 == '1') {

Hr0 = '0';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Hr1--;

**if** (Hr1 == '0') {

Hr1 = ' ';

}

Hr0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char(CLOCKDISPLAY\_CHAR\_0), Hr0, CLOCKDISPLAY\_GREEN, CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

**else** **if** (Hr0 == '0') {

Hr1--;

**if** (Hr1 == '0') {

Hr1 = ' ';

}

Hr0 = '9';

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK, CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

**else** {

Hr0--;

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

}

// Returns the coordinates of last touched area

uint8\_t **clockDisplay\_touchedArea**() {

int16\_t x = 0;

int16\_t y = 0;

uint8\_t z = 0;

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

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

**if** (y <= 115) {

**if** (x <= (int32\_t) X0) {

clockDisplay\_IncHr();

}

**else** **if** ( x >= (int32\_t) X1) {

clockDisplay\_IncSec();

}

**else** {

clockDisplay\_IncMin();

}

}

**else** {

**if** (x <= (int32\_t) X0) {

clockDisplay\_DecHr();

}

**else** **if** ( x <= (int32\_t) X1) {

clockDisplay\_DecMin();

}

**else** {

clockDisplay\_DecSec();

}

}

x = 0;

y = 0;

z = 0;

**return** 0;

}

/\*

\* Public functions

\*/

// Init function

**void** **clockDisplay\_init**() {

display\_init(); // Must init all of the software and underlying hardware for LCD.

display\_fillScreen(CLOCKDISPLAY\_\_BLACK); // Blank the screen.

display\_setRotation(true);

display\_setTextColor(CLOCKDISPLAY\_GREEN);

display\_setTextSize(CLOCKDISPLAY\_DISP\_TXT\_SIZE);

clockDisplay\_updateTimeDisplay(true);

// Sets the colons on display; they never change

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_5),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Colon,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_2),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Colon,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_0 ,CLOCKDISPLAY\_GREEN);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_1 ,CLOCKDISPLAY\_GREEN);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_2 ,CLOCKDISPLAY\_GREEN);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_3 ,CLOCKDISPLAY\_GREEN);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_4 ,CLOCKDISPLAY\_GREEN);

clockDisplay\_Triangle(CLOCKDISPLAY\_DISP\_TRIANGLE\_5 ,CLOCKDISPLAY\_GREEN);

}

// Updates the time values on clock

**void** **clockDisplay\_updateTimeDisplay**(bool forceUpdateAll) {

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_0),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_1),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Hr0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_3),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_4),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Min0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_6),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec1,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

display\_drawChar(clockDisplay\_x\_char (CLOCKDISPLAY\_CHAR\_7),clockDisplay\_y\_char (CLOCKDISPLAY\_CHAR\_0),Sec0,CLOCKDISPLAY\_GREEN,CLOCKDISPLAY\_\_BLACK,CLOCKDISPLAY\_DISP\_TXT\_SIZE);

}

// Increments or decrements depending on touched region of display

**void** **clockDisplay\_performIncDec**() {

clockDisplay\_touchedArea();

}

// Time is advanced one second; rolls over sec -> min -> hr

**void** **clockDisplay\_advanceTimeOneSecond**() {

clockDisplay\_IncSec();

**if** (Sec0 == '0' && Sec1 == '0') {

clockDisplay\_IncMin();

**if** (Min0 == '0' && Min1 == '0') {

clockDisplay\_IncHr();

**if** (Hr0 == '0' && Hr1 == '0') {

clockDisplay\_Clr\_Time();

clockDisplay\_updateTimeDisplay(1);

}

}

}

}

// Test function that verifies the inc/dec function of each clock element and the touch feature

**void** **clockDisplay\_runTest**() {

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_IncSec();

}

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_IncMin();

}

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_IncHr();

}

clockDisplay\_Clr\_Time();

clockDisplay\_updateTimeDisplay(true);

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_DecSec();

}

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_DecMin();

}

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_ROLL\_CONST1 ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_DecHr();

}

clockDisplay\_Clr\_Time();

clockDisplay\_updateTimeDisplay(true);

**for** (uint32\_t i = 0 ; i < CLOCKDISPLAY\_RUNTEST\_CONST ; i++) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_advanceTimeOneSecond();

}

int16\_t x = 0;

int16\_t y = 0;

uint8\_t z = 0;

uint8\_t counter = 0;

**printf**("touchscreen test:");

**while**(1) {

utils\_msDelay(CLOCKDISPLAY\_RUNTEST\_CONST);

clockDisplay\_performIncDec();

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

**if** (display\_isTouched()) {

**printf**("Coordinates (%u ,",x);

**printf**(" %u)",y);

**printf**("with pressure %u \n\r",z);

counter++;

}

**if** (counter == CLOCKDISPLAY\_ROLL\_CONST2) {

counter = 0;

}

x= 0;

y= 0;

z = 0;

}

}