Tide Cycle Timer

The purpose of this document is to describe the hardware and software components of the Arduino microcontroller based Tide Cycle Timer. This documentation includes the SD Card file formats and the cabling between the Arduino Mega2560 to the two IoT relays. Also included are instructions for initial file configuration/setup, and the process for updating the Arduino software.

The Arduino Mega2560 is used as the Microcontroller for the system. The data logger shield uses an SD Card for configuration files and logging. The breadboard connects to the IoT relays to turn on/off the 120V receptacles used to control the solenoid for the "fill" operation and the pump for the "empty" operation.

The Controller software records temperature readings approximately every 60 seconds.
All temperature readings and settings are in degrees Centigrade.

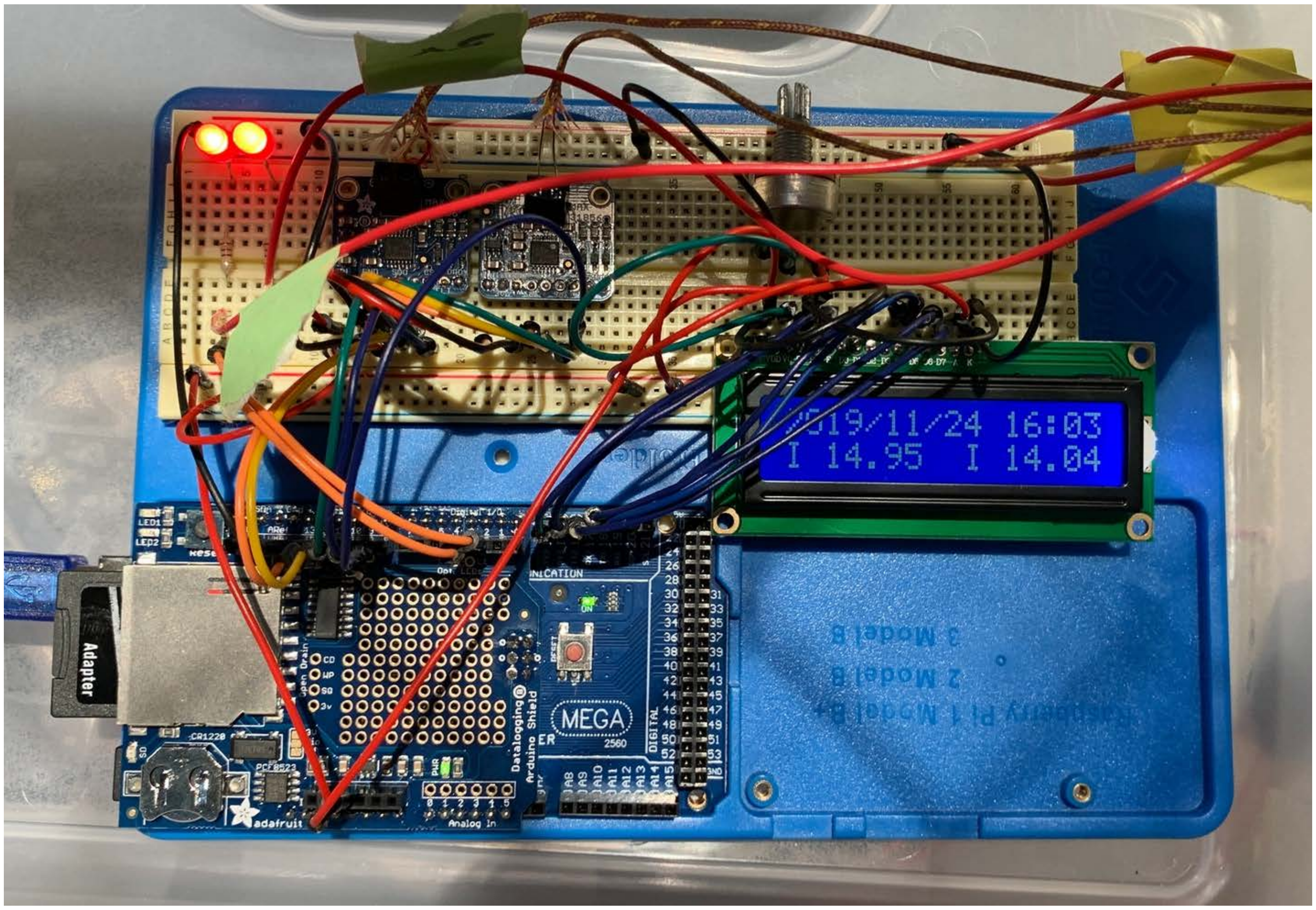
##
2019-11-25 Rev 0.2

Tom Rolander, MSEE
Mentor, Circuit Design & Software
Miller Library, Fabrication Lab
Hopkins Marine Station, Stanford University

Table of Contents

Controller Operation

The Controller is operated by plugging in the USB power connector on the Arduino Mega 2560 microcontroller.



The SD Card reader is used for input (CLOCK.CSV, CONTROL1.CSV, CONTROL2.CSV) and for output (LOGGING1.CSV, LOGGING2.CSV) from the Arduino.

The LCD displays the current date and time

YYYY/MM/DD HH:MM

and the tide states and temperature in each of the tank systems.

I = Fill

O = Empty

The temperature is in degrees Centigrade.

Note that the colon between HH and MM is toggled on/off at approximately every second to indicate that the Tide Cycle Timer is functioning correctly.

Max31856 Thermocouple to Digital Converters are used to obtain the tank system temperatures.

SD Card Files:

CLOCK.CSV (and PRVCLOCK.CSV)

The Data Logging Shield for the Arduino Mega2560 has a Real Time Clock (RTC) and battery. Since there is no Internet access from the controller, the RTC must be initialized with date and time. Once the RTC date and time are initialized, the time will be maintained by the Data Logging Shield. The RTC date and time are input from an optional CLOCK.CSV file on the SD Card. If the CLOCK.CSV file is present at start up of the Controller, the PRVCLOCK.CSV file, is deleted if present, and the date and time are set by the contents of the CLOCK.CSV file. After completing this operation the CLOCK.CSV file is renamed to PRVCLOCK.CSV.

CSV File format:

```
"Year","Month","Day","Hour","Minute","Second"
2019,5,9,10,55,0
```

Note that the RTC will lose or gain time over a period of time (weeks) and will require updating with the CLOCK.CSV file for the current date and time.

CONTROL1.CSV and CONTROL2.CSV

The Tide Cycle Timer uses values that are input from the respective CONTROLx.CSV file. This file has the date and time for each "fill" and "empty" tide cycle for each of the two tank systems.

CSV File Format:

```
"date","time","tide"
10/20/19,10:06,fill
10/20/19,18:59,empty
10/20/19,22:32,fill
10/21/19,7:25,empty
...
```

LOGGING1.CSV and LOGGING2.CSV

These are the CSV files where the temperatures are recorded for each tank system, once every minute. Each time the Controller is powered ON, a log entry of "Start up" is made. The date / time and operation (Startup, fill, empty) with a temperature reading are recorded in the log file.

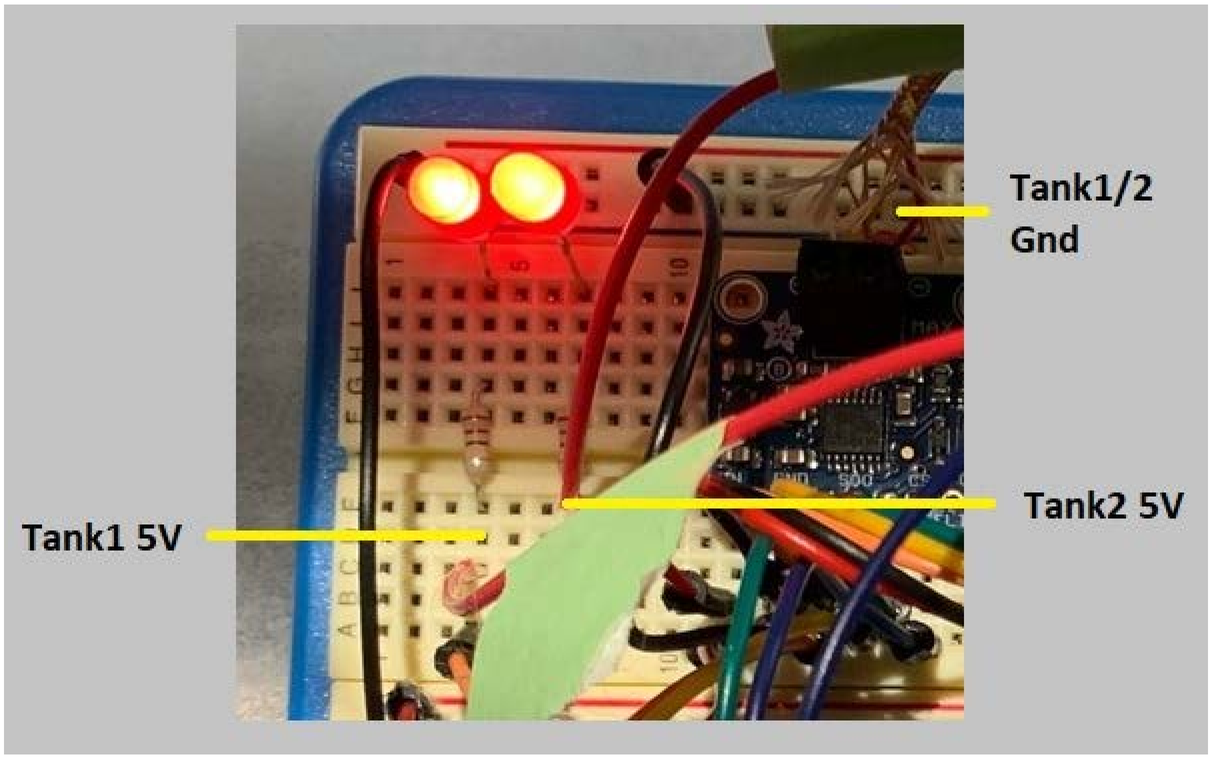
CSV File Format:

```
DateTime,Tide,Temp
10/25/2019 15:50,StartUp,0
10/16/2019 18:28,empty,24.44
10/16/2019 18:29,fill,24.4
10/16/2019 18:30,fill,24.47
…
```

Cabling between the Arduino Mega2560 and the two IoT Relays

A pair of cables (5V and Gnd) go from the Arduino breadboard to each of the two IoT relays. These are simply press fits into the breadboard. A Ferrite Core has been added to reduce voltage spikes when the relay is switched. This is an attempt to reduce the LCD display corruption.

Note that the Arduino microcontroller should be power on/off reset iif the LCD display has stopped refreshing or shows random characters.



Initial File Configuration and Setup

CLOCK.CSV

The Controller does not have a connection to the Internet to obtain current date and time. However, a real time clock (RTC) is included on the data logging shield. This RTC can be set using a CLOCK.CSV file. The contents of this file are described in the file format section CLOCK.CSV.

Before powering ON the Controller create a CLOCK.CSV file on the SD Card with the desired Date and Time. The SD Card has a PRVCLOCK.CSV file which can be used as a template for the CLOCK.CSV file. Since it will take a few minutes between the time you create, edit and store the CLOCK.CSV file on the SD Card, enter a time that is ahead by several minutes. With the SD Card inserted in the Controller, turn it ON a few seconds before the time which you have edited in the CLOCK.CSV file.

CONTROL1.CSV and CONTROL2.CSV

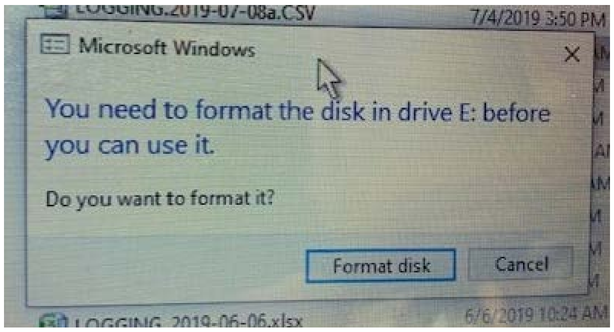
Prepare the CONTROLx.CSV files with the fill and empty date / time for each of the two tank systems.

Working with the LOGGINGx.CSV Files

The LOGGINGx.CSV files contain a log of temperature readings taken approximately each 60 seconds for each of the two thermocouples. An entry (Startup) is also made in the log file each time the Controller is powered ON.

The SD Card can be removed from the Controller and inserted in a laptop computer. Copy the LOGGING1.CSV and LOGGING2.CSV files to a laptop. It may be desirable to edit the LOGGINGx.CSV files on the SD Card to trim the data. If the data is trimmed, leave the first line in the CSV file containing the column headings.

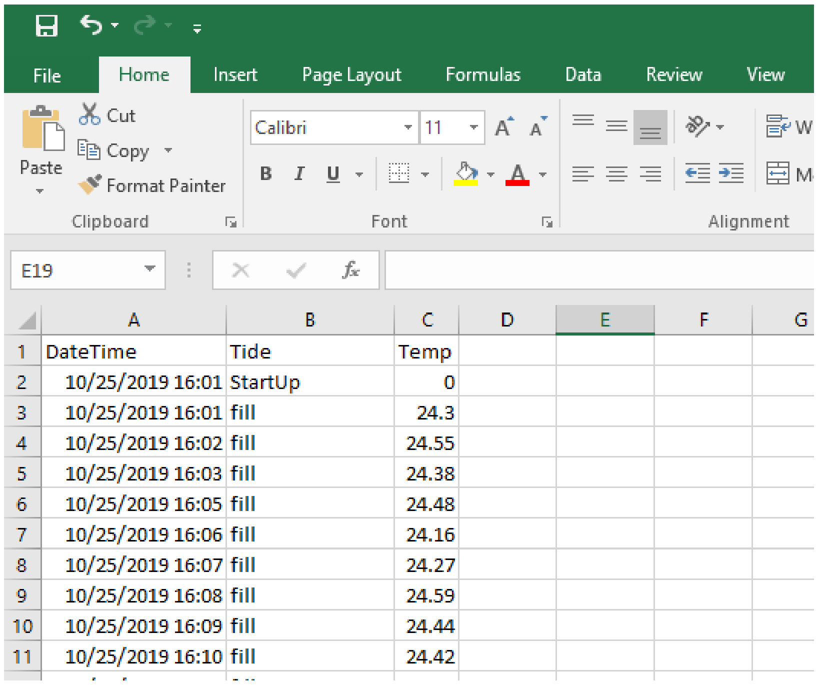
When the SD Card is inserted in a Windows / Mac computer you may see a warning to format the disk. Do not format the disk, select "Cancel", remove the SD Card and reinsert it in your laptop.



You need to format the disk in drive E: before you can use it.

Do you want to format it?

Sample of LOGGINGx.CSV



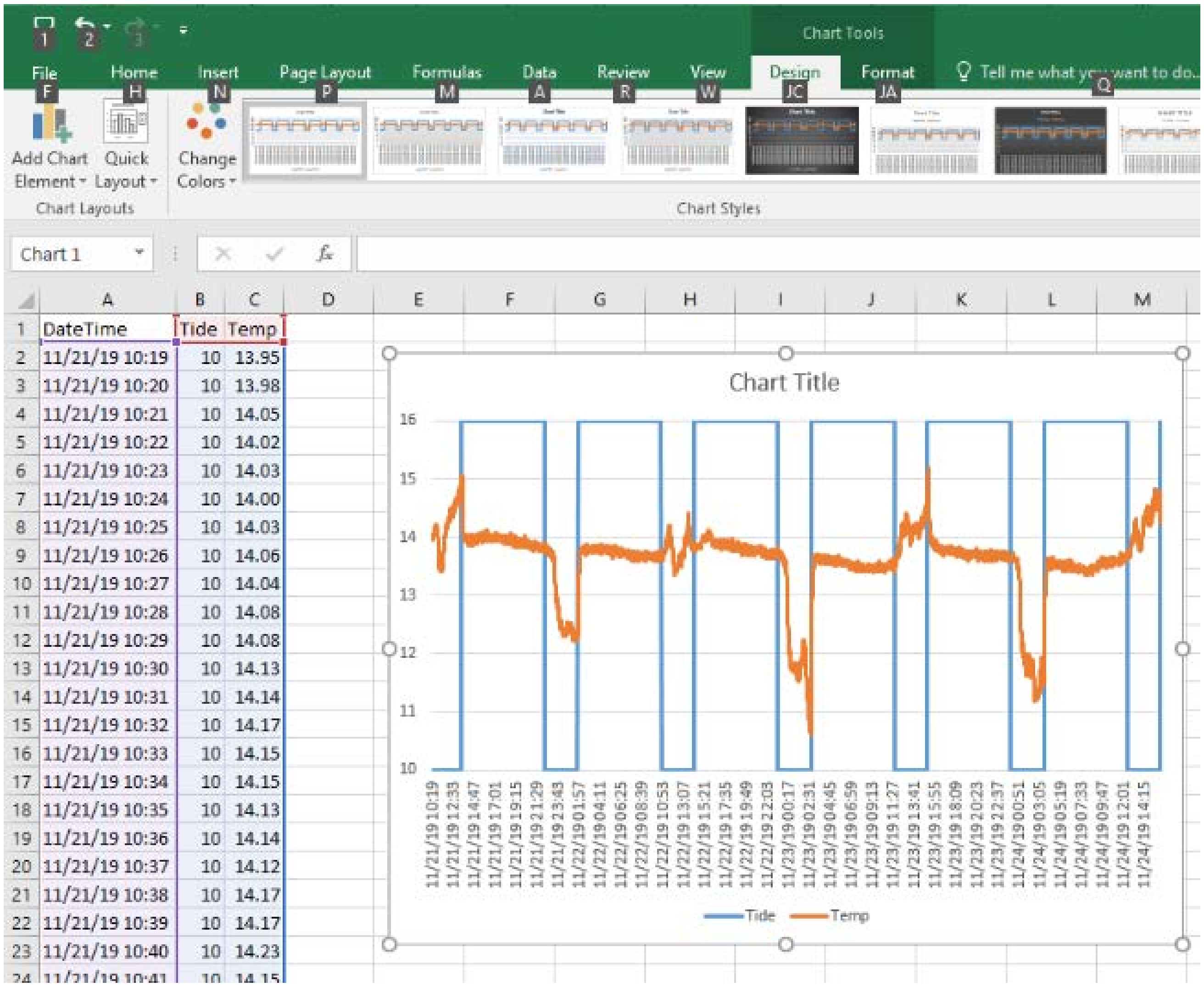
	A	B	C	D	E	F	G
1	DateTime	Tide	Temp				
2	10/25/2019 16:01	StartUp	0				
3	10/25/2019 16:01	fill	24.3				
4	10/25/2019 16:02	fill	24.55				
5	10/25/2019 16:03	fill	24.38				
6	10/25/2019 16:05	fill	24.48				
7	10/25/2019 16:06	fill	24.16				
8	10/25/2019 16:07	fill	24.27				
9	10/25/2019 16:08	fill	24.59				
10	10/25/2019 16:09	fill	24.44				
11	10/25/2019 16:10	fill	24.42				

Note that the temperature readings are separately recorded for each of the two thermocouples in the two tank systems.

In order to plot the temperatures and the tide cycles, use a text editor to edit the CSV file:

- Replace "fill" with a suitable high number such as 16
- Replace "empty" with a suitable low number such as 10
- Delete all of the "Startup" lines (to avoid zero values)

Here is an example of plotting the data for Tank1.



	A	B	C
1	DateTime	Tide	Temp
2	11/21/19 10:19	10	13.95
3	11/21/19 10:20	10	13.98
4	11/21/19 10:21	10	14.05
5	11/21/19 10:22	10	14.02
6	11/21/19 10:23	10	14.03
7	11/21/19 10:24	10	14.00
8	11/21/19 10:25	10	14.03
9	11/21/19 10:26	10	14.06
10	11/21/19 10:27	10	14.04
11	11/21/19 10:28	10	14.08
12	11/21/19 10:29	10	14.08
13	11/21/19 10:30	10	14.13
14	11/21/19 10:31	10	14.14
15	11/21/19 10:32	10	14.17
16	11/21/19 10:33	10	14.15
17	11/21/19 10:34	10	14.15
18	11/21/19 10:35	10	14.13
19	11/21/19 10:36	10	14.14
20	11/21/19 10:37	10	14.12
21	11/21/19 10:38	10	14.17
22	11/21/19 10:39	10	14.17
23	11/21/19 10:40	10	14.23
24	11/21/19 10:41	10	14.15

Updating the Arduino INO file

The software for the Controller can be updated by means of a USB cable between a laptop computer and the Arduino Controller, and the Arduino IDE (Integrated Development Environment).

The Arduino IDE can be downloaded here for Windows / Mac:
https://www.arduino.cc/en/main/software

After installing the Arduino IDE the following libraries must be installed:

Tools -> Manage Libraries -> Install

`Adafruit_MAX31856`	`Adafruit MAX31856 library by Adafruit Version 1.0.3`
`SPI`	`Built-In`
`LiquidCrystal`	`LiquidCrystal by Adafruit Version 1.0.7`
`RTClib`	`Built-In`
`SD`	`Built-In`

The process for updating the Arduino Controller is as follows:

- Copy the Tide_Cycle_Timer.ino file into the folder:
  Documents\Arduino\Tide_Cycle_Timer
- Sketch -> Verify/Compile
  To confirm that the new INO file will compile correctly. Note that the first time this is done on a newly installed Arduino IDE, you will likely need to add some missing libraries (MAX31856, etc) see above listing of required Libraries.
- Connect the USB cable from the laptop to the Arduino
- Select the port used by the USB connection
  Tools -> Port -> xxx (e.g. COM5)
- Sketch -> Upload
  The INO file will be compiled, uploaded to the Arduino, and execution will begin
- Disconnect the USB cable
- Alternatively for additional Debug Output the Serial Monitor can be opened to display debug output

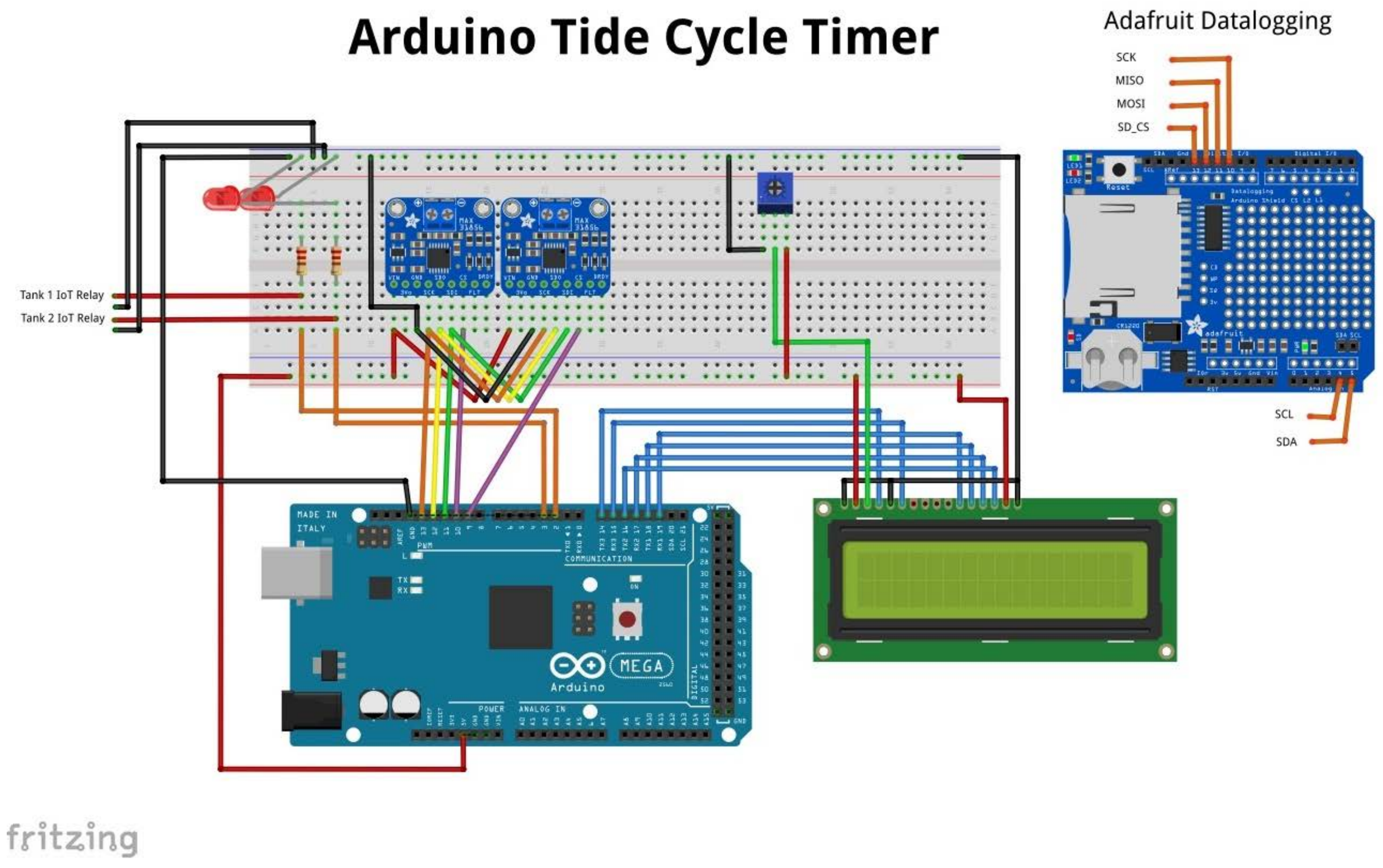
Formatting SD Card under Windows/Mac

Chances a new SD Card is already pre-formatted with a FAT filesystem. However you may have problems with how the factory formats the card, or if it's an old card it needs to be reformatted. The Arduino SD library used supports both FAT16 and FAT32 file systems. Note that formatting will erase the card so save anything you want first.

The official SD formatter is available from https://www.sdcard.org/downloads/formatter_4/

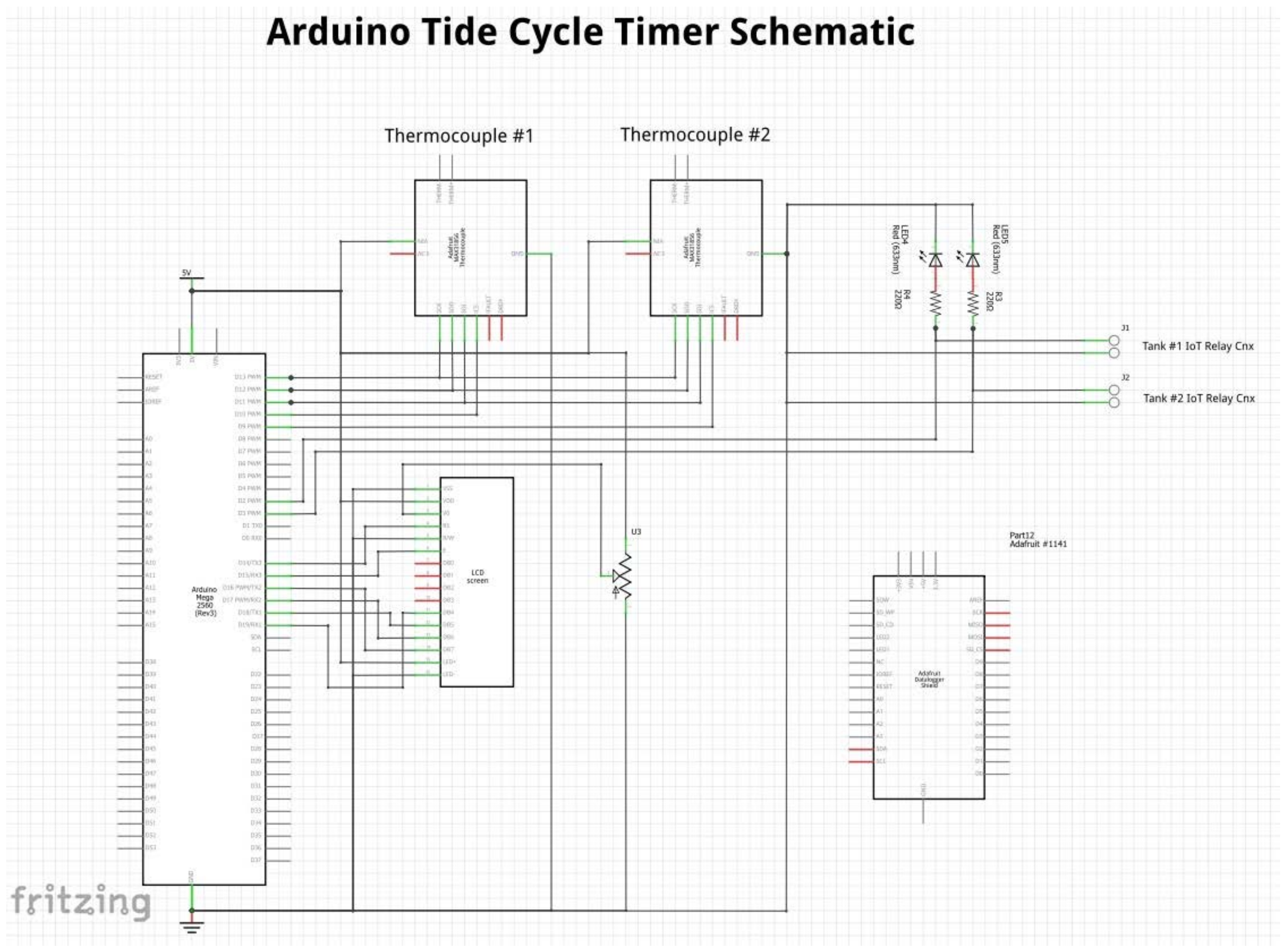
Download it and run it on your computer, there's also a manual linked from that page for use.

Breadboard

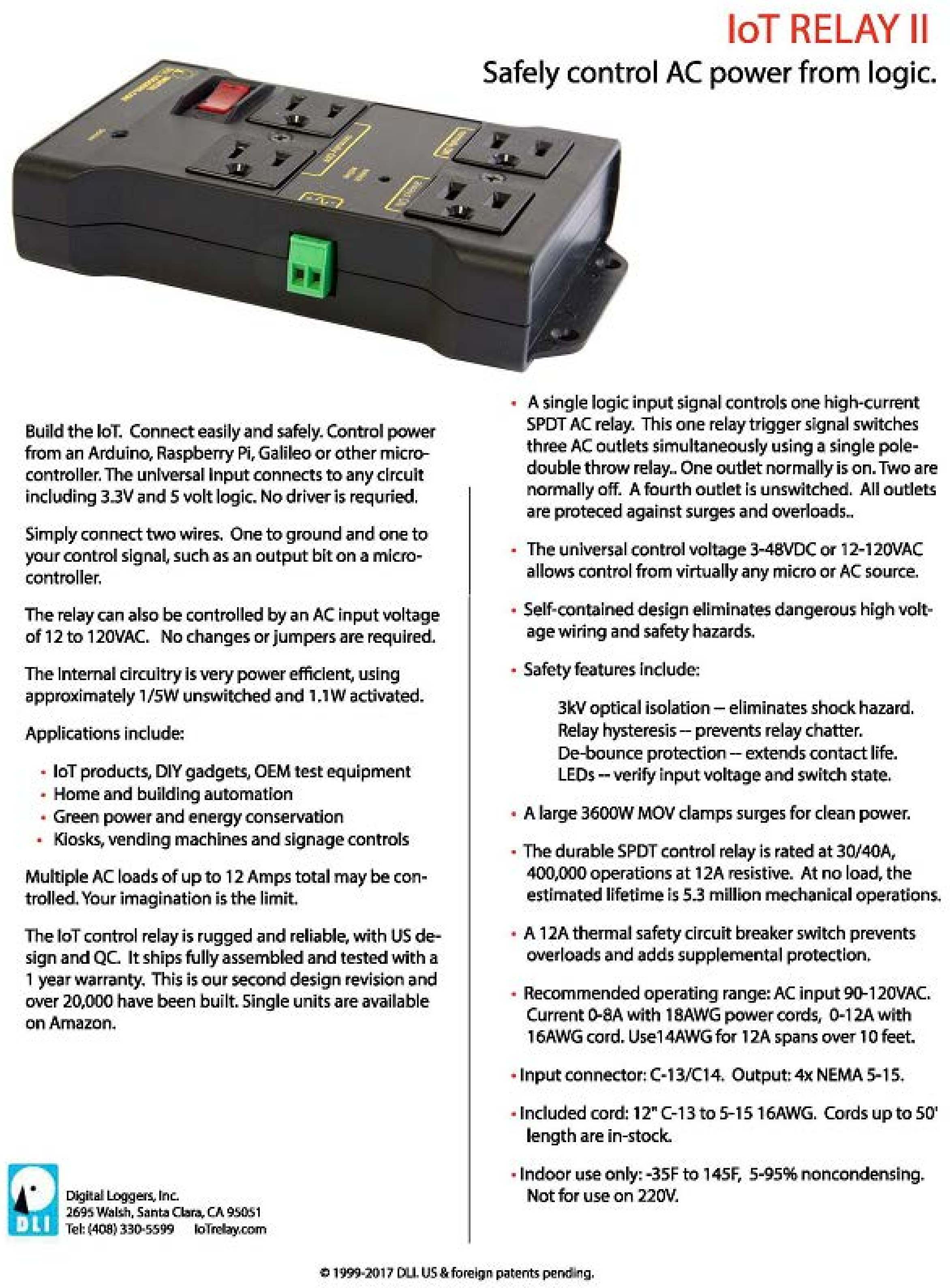


Arduino Tide Cycle Timer
Adafruit Datalogging
SCK
MISO
MOSI
SD_CS
Tank 1 IoT Relay
Tank 2 IoT Relay
SCL
SDA
fritzing

Schematic


Arduino Tide Cycle Timer Schematic
Thermocouple #1
Thermocouple #2
Adafruit MAX31856 Thermocouple
5V
Arduino Mega 2560 (Rev3)
LCD screen
U3
LED4 Red (633nm)
LED5 Red (633nm)
R4 220Ω
R3 220Ω
J1
Tank #1 IoT Relay Cnx
J2
Tank #2 IoT Relay Cnx
Part12 Adafruit #1141
Adafruit Datalogger Shield
fritzing

IoT Relay



IoT RELAY II

Safely control AC power from logic.

Build the IoT. Connect easily and safely. Control power from an Arduino, Raspberry Pi, Galileo or other micro-controller. The universal input connects to any circuit including 3.3V and 5 volt logic. No driver is requried.

Simply connect two wires. One to ground and one to your control signal, such as an output bit on a micro-controller.

The relay can also be controlled by an AC input voltage of 12 to 120VAC. No changes or jumpers are required.

The internal circuitry is very power efficient, using approximately 1/5W unswitched and 1.1W activated.

Applications include:

- IoT products, DIY gadgets, OEM test equipment
- Home and building automation
- Green power and energy conservation
- Kiosks, vending machines and signage controls

Multiple AC loads of up to 12 Amps total may be controlled. Your imagination is the limit.

The IoT control relay is rugged and reliable, with US design and QC. It ships fully assembled and tested with a 1 year warranty. This is our second design revision and over 20,000 have been built. Single units are available on Amazon.

- A single logic input signal controls one high-current SPDT AC relay. This one relay trigger signal switches three AC outlets simultaneously using a single pole-double throw relay.. One outlet normally is on. Two are normally off. A fourth outlet is unswitched. All outlets are proteced against surges and overloads..
- The universal control voltage 3-48VDC or 12-120VAC allows control from virtually any micro or AC source.
- Self-contained design eliminates dangerous high voltage wiring and safety hazards.
- Safety features include:

  3kV optical isolation – eliminates shock hazard.
  Relay hysteresis – prevents relay chatter.
  De-bounce protection – extends contact life.
  LEDs -- verify input voltage and switch state.

- A large 3600W MOV clamps surges for clean power.
- The durable SPDT control relay is rated at 30/40A, 400,000 operations at 12A resistive. At no load, the estimated lifetime is 5.3 million mechanical operations.
- A 12A thermal safety circuit breaker switch prevents overloads and adds supplemental protection.
- Recommended operating range: AC input 90-120VAC. Current 0-8A with 18AWG power cords, 0-12A with 16AWG cord. Use14AWG for 12A spans over 10 feet.
- Input connector: C-13/C14. Output: 4x NEMA 5-15.
- Included cord: 12" C-13 to 5-15 16AWG. Cords up to 50' length are in-stock.
- Indoor use only: -35F to 145F, 5-95% noncondensing. Not for use on 220V.

DLI
Digital Loggers, Inc.
2695 Walsh, Santa Clara, CA 95051
Tel: (408) 330-5599 IoTrelay.com

Arduino Mega 2560 -- INO Source Code

```
/**************************************************************************
  Tide Cycle Timer

  Original Code:  2019-10-07

  Tom Rolander, MSEE
  Mentor, Circuit Design & Software
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 **************************************************************************/

#define PROGRAM "Tide Cycle Timer"
#define VERSION "V 0.4 2019-10-28"

#define STARTUP_PAUSE_DELAY 2000

#define MAX_TIMERS  2
int imaxTimers = 2;

#define USE_DEBUG_INPUT 0

typedef struct
{
  uint16_t Year;
  uint8_t  Month;
  uint8_t  Day;
  uint8_t  Hour;
  uint8_t  Minute;
  uint8_t  Fill;     // 1==Fill, 0=Empty
} Tides;

int nTides[2] = {0, 0};

Tides tidesController_1[4] =
{
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0}
};

Tides tidesController_2[4]
{
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0}
};

unsigned long timeLastLog = 0;
int currentYear  = 0;;
int currentMonth = 0;
int currentDay   = 0;
int currentHour  = 0;
int currentMin   = 0;

#include <SPI.h>
```

```
// SD Card used for data logging
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// SD Shield
//
// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
// MKRZero SD: SDCARD_SS_PIN
//#define chipSelectSDCard 4
#define chipSelectSDCard 10

File fileSDCard;

// Date and time functions using a DS1307 RTC connected via I2C and Wire lib
#include "RTClib.h"

RTC_PCF8523 rtc;
DateTime now;

#include <LiquidCrystal.h>

// Initialize the LCD library by associating LCD interface pins
// with the arduino pin number it is connected to
const int rs = 14, en = 15, d4 = 19, d5 = 18, d6 = 17, d7 = 16;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

bool bSDLogFail = false;

const int powerPinController_1 = 2;
const int powerPinController_2 = 3;

const int powerPinControllers[2] = {powerPinController_1, powerPinController_2};

int powerStateController_1 = LOW;
int powerStateController_2 = LOW;

#include <Adafruit_MAX31856.h>

// Use software SPI: CS, DI, DO, CLK
Adafruit_MAX31856 max[MAX_TIMERS] = {
  Adafruit_MAX31856(10, 11, 12, 13),
  Adafruit_MAX31856( 9, 11, 12, 13)
};

// String constants
char ERROR_NO_RTC[]       = "Couldnt find RTC";
char ERROR_NO_SDCARD[]    = "SD Card Missing?";

void setup()
{
  Serial.begin(9600);
  Serial.println("");
  Serial.println(PROGRAM);
  Serial.println(VERSION);
  Serial.println("Serial Initialized...");
```

```
// set up the LCD's number of columns and rows:
lcd.begin(16, 2);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(PROGRAM);
lcd.setCursor(0, 1);
lcd.print(VERSION);
delay(STARTUP_PAUSE_DELAY);

pinMode(powerPinController_1, OUTPUT);
pinMode(powerPinController_2, OUTPUT);
digitalWrite(powerPinController_1, powerStateController_1);
digitalWrite(powerPinController_2, powerStateController_2);

Serial.println("MAX31856 thermocouple test");
Serial.println("Thermocouple type: ");

for (int i = 0; i < imaxTimers; i++) {
  max[i].begin();
  max[i].setThermocoupleType(MAX31856_TCTYPE_K);

  Serial.print("Controller ");
  Serial.print(i + 1);
  Serial.print(" ");
  switch ( max[i].getThermocoupleType() ) {
    case MAX31856_TCTYPE_B: Serial.println("B Type"); break;
    case MAX31856_TCTYPE_E: Serial.println("E Type"); break;
    case MAX31856_TCTYPE_J: Serial.println("J Type"); break;
    case MAX31856_TCTYPE_K: Serial.println("K Type"); break;
    case MAX31856_TCTYPE_N: Serial.println("N Type"); break;
    case MAX31856_TCTYPE_R: Serial.println("R Type"); break;
    case MAX31856_TCTYPE_S: Serial.println("S Type"); break;
    case MAX31856_TCTYPE_T: Serial.println("T Type"); break;
    case MAX31856_VMODE_G8: Serial.println("Voltage x8 Gain mode"); break;
    case MAX31856_VMODE_G32: Serial.println("Voltage x8 Gain mode"); break;
    default: Serial.println("Unknown"); break;
  }
  max[i].setTempFaultThreshholds(0.0, 200.0);
}

// Initialize the Real Time Clock
if (! rtc.begin())
{
  Serial.println(ERROR_NO_RTC);
  LCD_DisplayErrorAndHALT(ERROR_NO_RTC);
}
if (! rtc.initialized())
{
  Serial.println("RTC isnt running");
}
now = rtc.now();
currentYear = now.year();
currentMonth = now.month();
currentDay  = now.day();
currentHour = now.hour();
currentMin  = now.minute();

Serial.println("*** DATE ***    ");
Serial.print(currentYear, DEC);
Serial.print("/");
SerialPrintTwoDigits(currentMonth);
Serial.print("/");
SerialPrintTwoDigits(currentDay);
Serial.print(" ");
```

```
  SerialPrintTwoDigits(currentHour);
  Serial.print(":");
  SerialPrintTwoDigits(currentMin);
  Serial.println("");

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("*** DATE ***    ");
  lcd.setCursor(0, 1);
  lcd.print(currentYear, DEC);
  lcd.print("/");
  LCD_PrintTwoDigits(currentMonth);
  lcd.print("/");
  LCD_PrintTwoDigits(currentDay);
  lcd.print(" ");
  LCD_PrintTwoDigits(currentHour);
  lcd.print(":");
  LCD_PrintTwoDigits(currentMin);
  delay(STARTUP_PAUSE_DELAY);

  SetupSDCardOperations();

  Serial.println("Setup complete.");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Setup complete.");
  lcd.setCursor(0, 1);
  delay(STARTUP_PAUSE_DELAY);
}

void loop()
{
  now = rtc.now();
  currentYear = now.year();
  currentMonth = now.month();
  currentDay  = now.day();
  currentHour = now.hour();
  currentMin  = now.minute();

#if USE_DEBUG_INPUT

#if 0
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Serial Interface");
  lcd.setCursor(0, 1);
  lcd.print("DEBUG INPUT");
  delay(STARTUP_PAUSE_DELAY);
#endif

  char line[64];
  while (1)
  {
    Serial.println("Enter: MM/DD/YYYY,HH:MM");
    while (!Serial.available());
    int size = Serial.readBytesUntil('\n', line, sizeof(line));
    line[size] = '\0';
    Serial.print("[");
    Serial.print(line);
    Serial.println("]");

    if (size != 16 ||
        line[2] != '/' ||
        line[5] != '/' ||
        line[10] != ',' ||
        line[13] != ':')
    {
      Serial.println("ERROR: Incorrect input format!");
      Serial.print(">>");
      Serial.print(line);
      Serial.println("<<");
      continue;
    }
    break;
  }

  char *ptr;

  ptr = &line[0];
  currentMonth = atoi(ptr);
  while (*ptr) {
    if (*ptr++ == '/') break;
  }
  currentDay = atoi(ptr);
  while (*ptr) {
    if (*ptr++ == '/') break;
  }
  currentYear = atoi(ptr);

  while (*ptr) {
    if (*ptr++ == ',') break;
  }

  currentHour = atoi(ptr);
  while (*ptr) {
    if (*ptr++ == ':') break;
  }
  currentMin = atoi(ptr);
#endif

  Tides* tidesController_X;

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(currentYear, DEC);
  lcd.print("/");
  LCD_PrintTwoDigits(currentMonth);
  lcd.print("/");
  LCD_PrintTwoDigits(currentDay);
  lcd.print(" ");
  LCD_PrintTwoDigits(currentHour);
  lcd.print(":");
  LCD_PrintTwoDigits(currentMin);

  for (int i = 0; i < 2; i++)
  {
    int iTide = 0;
    int iFill = 0;

    lcd.setCursor((i * 9), 1);

    tidesController_X = (i == 0) ? &tidesController_1[0] : &tidesController_2[0];

    if (nTides[i] == 0 ||
        currentYear != tidesController_X->Year ||
        currentMonth != tidesController_X->Month ||
        currentDay  != tidesController_X->Day)
      nTides[i] = GetSDCardTides(i + 1, currentYear, currentMonth, currentDay, tidesController_X);
```

```
Serial.print(currentYear);
Serial.print("/");
SerialPrintTwoDigits(currentMonth);
Serial.print("/");
SerialPrintTwoDigits(currentDay);
Serial.print(",");
SerialPrintTwoDigits(currentHour);
Serial.print(":");
SerialPrintTwoDigits(currentMin);
Serial.print(" ");

Serial.print("Tank ");
Serial.print(i + 1);
Serial.print(" ");

if (nTides[i] == 0)
{
  Serial.println("ERROR: No tide cycle information for that date!");
  continue;
}

while (iTide < nTides[i])
{
  if (currentHour < tidesController_X[iTide].Hour)
    break;
  if (currentHour == tidesController_X[iTide].Hour &&
      currentMin  <  tidesController_X[iTide].Minute)
    break;
  iTide++;
}
if (iTide >= nTides[i])
{
  iFill = tidesController_X[nTides[i] - 1].Fill;
}
else
{
  iFill = (tidesController_X[iTide].Fill == 1) ? 0 : 1;
}
if (iFill == 1)
{
  lcd.print("I ");
  Serial.print(" fill ");
  digitalWrite(powerPinControllers[i], HIGH);
}
else
{
  lcd.print("O ");
  Serial.print(" empty");
  digitalWrite(powerPinControllers[i], LOW);
}

float temp;
temp = max[i].readThermocoupleTemperature();

int iTemp = (int) ((temp + 0.005) * 100.0);
LCD_PrintTwoDigits(iTemp / 100);
lcd.print(".");
LCD_PrintTwoDigits(iTemp % 100);

Serial.print(" Temp = ");
Serial.print(temp);
Serial.print("c");
```

```
    // Check and print any faults
    uint8_t fault = max[i].readFault();
    if (fault) {
      Serial.print(" Fault = ");
      Serial.print(fault);

      if (fault & MAX31856_FAULT_CJRANGE) Serial.print("Cold Junction Range Fault");
      if (fault & MAX31856_FAULT_TCRANGE) Serial.print("Thermocouple Range Fault");
      if (fault & MAX31856_FAULT_CJHIGH)  Serial.print("Cold Junction High Fault");
      if (fault & MAX31856_FAULT_CJLOW)   Serial.print("Cold Junction Low Fault");
      if (fault & MAX31856_FAULT_TCHIGH)  Serial.print("Thermocouple High Fault");
      if (fault & MAX31856_FAULT_TCLOW)   Serial.print("Thermocouple Low Fault");
      if (fault & MAX31856_FAULT_OVUV)    Serial.print("Over/Under Voltage Fault");
      if (fault & MAX31856_FAULT_OPEN)    Serial.print("Thermocouple Open Fault");
    }

    Serial.println("");

    SDLogging(currentYear, currentMonth, currentDay, currentHour, currentMin, i + 1, (iFill == 1) ? "fill" :
"empty", temp);

  }

  while (1)
  {
    now = rtc.now();
    if (currentMin  != now.minute())
      break;
    delay(1000);
    lcd.setCursor(13, 0);
    if ((now.second() & 0x0001) == 0x0001)
      lcd.print(":");
    else
      lcd.print(" ");
  }
}

// Initialize the SD for operations
// If the LOGGINGx.CSV file is not present create the file with the first line of column headings
void SetupSDCardOperations()
{
  Serial.print("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelectSDCard)) {
    Serial.println("initialization failed.");
    Serial.println(ERROR_NO_SDCARD);
    LCD_DisplayErrorAndHALT(ERROR_NO_SDCARD);
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.println();
  Serial.print("Card type:         ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
    while (1);
  }

  Serial.print("Clusters:          ");
  Serial.println(volume.clusterCount());
  Serial.print("Blocks x Cluster:  ");
  Serial.println(volume.blocksPerCluster());

  Serial.print("Total Blocks:      ");
  Serial.println(volume.blocksPerCluster() * volume.clusterCount());
  Serial.println();

  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("Volume type is:    FAT");
  Serial.println(volume.fatType(), DEC);

  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
  volumesize /= 2;                           // SD card blocks are always 512 bytes (2 blocks are 1KB)
  Serial.print("Volume size (Kb):  ");
  Serial.println(volumesize);
  Serial.print("Volume size (Mb):  ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Gb):  ");
  Serial.println((float)volumesize / 1024.0);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);

  Serial.println("");
  Serial.println("*** STATUS ***  ");
  Serial.println("SD Init Start   ");

  if (!SD.begin(chipSelectSDCard)) {
    Serial.println("*** ERROR ***   ");
    Serial.println("SD Init Failed  ");
    Serial.println("System HALTED!");
    while (1);
  }

  Serial.println("* Test CLOCK.CSV");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("* Test CLOCK.CSV");
  lcd.setCursor(0, 1);

  if (SD.exists("CLOCK.CSV"))
  {
    fileSDCard = SD.open("CLOCK.CSV", FILE_READ);
    if (fileSDCard)
    {
      char *ptr3 = 0;

      if (fileSDCard.available())
      {
        char strClockSetting[256];
        char strClockSettingCopy[256];
        fileSDCard.read(strClockSetting, sizeof(strClockSetting));
        fileSDCard.close();
        strClockSetting[sizeof(strClockSetting) - 1] = '\0';
        char *ptr1 = strchr(&strClockSetting[0], '\n');
        if (ptr1 != 0)
        {
          *ptr1++ = '\0';
        }
        Serial.println("Set Clock:");
        Serial.println(strClockSetting);

        ptr3 = strchr(ptr1, '\n');
        if (ptr3 != 0)
        {
          *ptr3++ = '\0';
          strcpy(strClockSettingCopy, ptr1);
        }
        Serial.println(ptr1);

        int iDateTime[6] = {0, 0, 0, 0, 0, 0};
        for (int i = 0; i < 6; i++)
        {
          char *ptr2 = strchr(ptr1, ',');
          if (ptr2 != 0)
          {
            *ptr2 = '\0';
            iDateTime[i] = atoi(ptr1);
            ptr1 = &ptr2[1];
          }
          else
          {
            if (i == 5)
              iDateTime[5] = atoi(ptr1);
            break;
          }
        }

        rtc.adjust(DateTime(iDateTime[0], iDateTime[1], iDateTime[2], iDateTime[3], iDateTime[4],
iDateTime[5]));

        if (SD.exists("PRVCLOCK.CSV"))
        {
          SD.remove("PRVCLOCK.CSV");
        }
        SD.remove("CLOCK.CSV");

        fileSDCard = SD.open("PRVCLOCK.CSV", FILE_WRITE);
        if (fileSDCard != 0)
        {
          fileSDCard.println("\"Year\",\"Month\",\"Day\",\"Hour\",\"Minute\",\"Second\"");
          fileSDCard.println(strClockSettingCopy);
          fileSDCard.close();
        }

        Serial.println("* processed *");
        lcd.print("* processed *");
      }
      else
      {
        fileSDCard.close();
      }
    }

  }
  else
  {
    Serial.println("* does not exist");
    lcd.print("* does not exist");
  }

  delay(STARTUP_PAUSE_DELAY);

  // DEBUG TESTING
  //GetSDCardTides(1, 2019, 10, 8, tidesController_1);
  //GetSDCardTides(2, 2019, 10, 9, tidesController_2);


//////////////////////////////////////////////////////////////////////////////////////////////////////////////
////

  // LOGGINGx.CSV file processing
  for (int i = 1; i < 3; i++)
  {
    char szLOGGINGx[32] = "LOGGINGx.CSV";

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("* Test CLOCK.CSV");
    lcd.setCursor(0, 1);

    szLOGGINGx[7] = '0' + i;

    Serial.print("* Test ");
    Serial.println(szLOGGINGx);

    fileSDCard = SD.open(szLOGGINGx);
    if (fileSDCard)
    {
      if (fileSDCard.available())
      {
      }
      fileSDCard.close();
    }
    else
    {
      fileSDCard = SD.open(szLOGGINGx, FILE_WRITE);
      if (fileSDCard)
      {
        fileSDCard.println("\"DateTime\",\"Tide\",\"Temp\"");
        fileSDCard.close();
      }
      else
      {
        Serial.println("*** ERROR ***   ");
        Serial.println("SD Write Failed ");
        while (1);
      }
    }
    Serial.println("* processed *");
  }
  SD.end();

  Serial.println("*** STATUS ***  ");
  Serial.println("SD Init Finish  ");

  SDLogging(currentYear, currentMonth, currentDay, currentHour, currentMin, 1, "StartUp", 0.0);
  SDLogging(currentYear, currentMonth, currentDay, currentHour, currentMin, 2, "StartUp", 0.0);
}

void SDLogging(int iYear, int iMonth, int iDay, int iHour, int iMinute, int iUnit, char *szOperation, float
fTemp)
{
  if (!SD.begin(chipSelectSDCard))
  {
    Serial.println("SD LogFail");
    return;
  }
  bSDLogFail = false;


  {
    char szLOGGINGx[32] = "LOGGINGx.CSV";
    szLOGGINGx[7] = '0' + iUnit;

    fileSDCard = SD.open(szLOGGINGx, FILE_WRITE);

    // if the file opened okay, write to it:
    if (fileSDCard)
    {
      fileSDCard.print(iYear, DEC);
      fileSDCard.print("/");
      fileSDCard.print(iMonth, DEC);
      fileSDCard.print("/");
      fileSDCard.print(iDay, DEC);
      fileSDCard.print(" ");
      fileSDCard.print(iHour, DEC);
      fileSDCard.print(":");
      fileSDCard.print(iMinute, DEC);
      fileSDCard.print(",");
      fileSDCard.print(szOperation);
      fileSDCard.print(",");
      fileSDCard.print(fTemp);
      fileSDCard.println("");
      fileSDCard.close();
    }
    else
    {
      // if the file didn't open, display an error:
      Serial.println("*** ERROR ***   ");
      Serial.println("Open LOGGING.CSV");
    }
  }

  SD.end();

}

void SerialPrintTwoDigits(int iVal)
{
  if (iVal < 10)
    Serial.print("0");
  Serial.print(iVal, DEC);
}
```

```
void LCD_PrintTwoDigits(int iVal)
{
  if (iVal < 10)
    lcd.print("0");
  lcd.print(iVal, DEC);
}

bool readLine(char* line, size_t maxLen) {
  for (size_t n = 0; n < maxLen; n++) {
    int c = fileSDCard.read();
    if ( c < 0 && n == 0) return false;  // EOF
    if (c < 0 || c == '\n') {
      line[n] = 0;
      return true;
    }
    line[n] = c;
  }
  return false; // line too long
}

int ReadTides(uint16_t Year_Current, uint8_t Month_Current, uint8_t Day_Current, Tides Tides_Current_Day[])
{
  int iTide = 0;

  for (int i = 0; i < 4; i++)
  {
    Tides_Current_Day[i].Year = 0;
    Tides_Current_Day[i].Month = 0;
    Tides_Current_Day[i].Day = 0;
    Tides_Current_Day[i].Hour = 0;
    Tides_Current_Day[i].Minute = 0;
    Tides_Current_Day[i].Fill = 0;
  }

  while (true)
  {
    char line[40], *ptr, *str;
    int iMonth, iDay, iYear;
    int iHour, iMinute;
    int iFill;
    if (!readLine(line, sizeof(line))) {
      return false;  // EOF or too long
    }

    ptr = &line[0];
    iMonth = atoi(ptr);
    while (*ptr) {
      if (*ptr++ == '/') break;
    }
    iDay = atoi(ptr);
    while (*ptr) {
      if (*ptr++ == '/') break;
    }
    iYear = atoi(ptr);

    if (iYear > Year_Current)
      break;
    if (iYear < Year_Current)
      continue;
    if (iMonth > Month_Current)
      break;
    if (iMonth < Month_Current)
      continue;
    if (iDay > Day_Current)
```

```
      break;
    if (iDay < Day_Current)
      continue;

    while (*ptr) {
      if (*ptr++ == ',') break;
    }

    iHour = atoi(ptr);
    while (*ptr) {
      if (*ptr++ == ':') break;
    }
    iMinute = atoi(ptr);

    while (*ptr) {
      if (*ptr++ == ',') break;
    }

    if (strstr(ptr, "fill"))
      iFill = 1;
    else
      iFill = 0;

    Tides_Current_Day[iTide].Year = iYear;
    Tides_Current_Day[iTide].Month = iMonth;
    Tides_Current_Day[iTide].Day = iDay;
    Tides_Current_Day[iTide].Hour = iHour;
    Tides_Current_Day[iTide].Minute = iMinute;
    Tides_Current_Day[iTide].Fill = iFill;

    iTide++;
  }
  return (iTide);
}

int GetSDCardTides(int iUnit, uint16_t Year_Current, uint8_t Month_Current, uint8_t Day_Current , Tides
Tides_Current_Day[])
{
  char szCONTROLx[32] = "CONTROLx.CSV";
  int iTide = 0;

  szCONTROLx[7] = '0' + iUnit;
  //  Serial.print("* Test ");
  //  Serial.println(szCONTROLx);

  //  lcd.clear();
  //  lcd.setCursor(0, 0);
  //  lcd.print("* ");
  //  lcd.print(szCONTROLx);
  //  lcd.setCursor(0, 1);

  if (!SD.begin(chipSelectSDCard))
  {
    Serial.println("SD Fail");
    return (0);
  }

  if (SD.exists(szCONTROLx))
  {
    fileSDCard = SD.open(szCONTROLx);
    if (fileSDCard)
    {
      if (fileSDCard.available())
      {
        iTide = ReadTides(Year_Current, Month_Current, Day_Current, Tides_Current_Day);

        Serial.print("Controller: ");
        Serial.println(iUnit);

        for (int i = 0; i < iTide; i++)
        {
          Serial.print("  Tide: ");
          Serial.print(i + 1);
          Serial.print(" ");
          SerialPrintTwoDigits(Tides_Current_Day[i].Month);
          Serial.print("/");
          SerialPrintTwoDigits(Tides_Current_Day[i].Day);
          Serial.print("/");
          Serial.print(Tides_Current_Day[i].Year);
          Serial.print(" ");
          SerialPrintTwoDigits(Tides_Current_Day[i].Hour);
          Serial.print(":");
          SerialPrintTwoDigits(Tides_Current_Day[i].Minute);
          if (Tides_Current_Day[i].Fill == 1)
            Serial.print(" fill");
          else
            Serial.print(" empty");
          Serial.println("");
        }
        //        Serial.println("* processed *");
        //        lcd.print("* processed *");
      }
      else
      {
        fileSDCard.close();
      }
    }
  }
  else
  {
    //    Serial.println("* does not exist");
    //    lcd.print("* does not exist");
  }
  //  delay(STARTUP_PAUSE_DELAY);

  SD.end();

  return (iTide);
}

void LCD_DisplayErrorAndHALT(char *errorMessage)
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("*** ERROR ***");
  lcd.setCursor(0, 1);
  lcd.print(errorMessage);
  while (1);
}
```