Passive Perch Scale design and implementation

**Problem:** HawkWatch International (HWI) identified a need to measure golden eagle weight in the field without human intervention or trapping. The first part of the problem was to create a semi-natural perch that the eagles would regularly perch on. Once the perch problem was solved, the next problem was to design a system that could measure, display and log a bird’s weight.

**Solution Requirements:**

1. Scale resolution: 0.1 kilograms or better
2. Accuracy: +/-0.05 kilograms
3. Range of measurement: 0.0 to 6.0 kilograms
4. Display: LED display that could be captured by a trail camera
5. Power: Battery or solar or a combination of the two
6. Storage: SD card or similar for logging weight of all perch events with date and time
7. Environmental: IP65 or better rating for the scale and perch
8. Location: Outdoor and exposed to all seasonal weather conditions
9. Perch: Post top design that will be accepted by golden eagles
10. Service interval: 1 visit per week

**Possible Solutions:** HWI researched existing methods and found a design from 1982 which was documented in the Journal of Ornithology Autumn of 1982. [A Scale for Weighing Birds at Habitual Perches on JSTOR](https://www.jstor.org/stable/4512767). This paper inspired the starting point for trials of a perch and scale design. For the perch, a t-bar design and a flat 6” post-type design were both evaluated. The t-bar design was tested for a few months, but it had issues with the weight not centered over the transducer and it allowed the potential for multiple birds to be present at once. With limited testing, it appeared that the eagles did not accept it as a perch. Instead, a simple flat 6” octagon post top was proposed and tested. This was quickly adopted by the birds, and it also made the perch transducer assembly much easier to design/build. For the scale, initially a mechanical solution using a spring and a linear graduation were evaluated. The idea was similar to a spring-based fish scale. However, after a few prototypes, it was determined that the motion of this type of design would spook the birds. The resolution and accuracy requirements also seemed difficult to attain. After the mechanical scale design an electronic design was trialed which quickly proved to be superior and was the direction pursued. This solution is described in this paper.

**Solution:** To satisfy the requirements, a perch and electronic scale were designed. The perch was integrated with a strain gauge transducer for converting the bird’s weight into an electrical signal. The perch was designed to replicate the top of a simple 6 to 7 inch circular post that eagles are often seen on. An octagon shaped perch was used since it was easy to cut and seemed a good approximation of a circular post top. The strain gauge transducer was mounted just under the perch top. The transducer needed to be IP65 rated to tolerate the outdoor conditions. A waterproof cable was selected and strain relieved to prevent damage to the transducer. This assembly was mounted to the top of a 4”x4” post with screws. The post was buried 3 feet to accommodate the landing momentum of the bird.

For the electronic scale, a waterproof box was designed which included a microcontroller to process the transducer data, to sense bird presence and to display/log the weight. The box was equipped with batteries and a solar charger to power the scale. Buttons were added to the side to control power and to initiate a calibration process. A 4 digit display was used to display the weight for observers or a monitoring trail camera. It was also used for prompts during calibration and time/date setting. The scale design incorporated a micro-SD card to log weight every 10 seconds when a bird was present on the scale. Each logged measurement includes a date and time stamp for cross referencing with trail camera photos. The time/date are correlated with the camera time/date stamps to specifically identify birds.

The designed system includes a bird perch, transducer assembly, post, electronic scale and solar panel as shown below.

Bird Perch

A device with a wire attached to a wooden post

Description automatically generated

Electronic scale

Solar panel

4”x4” post

Transducer Assembly

Twenty feet of waterproof wire was included between the electronic scale and the perch transducer. This allowed the box to be located away from the post. A handle was attached for clamping the box to a post or stake to keep it off the ground. The electronic scale and solar panel should be in an area that will receive maximum sunlight.

The system block diagram below shows each of the components utilized to meet the requirements.

**A diagram of a computer system

Description automatically generated**

**Microcontroller.** The scale was designed using an Arduino Nano controller. This module was chosen because of its simplicity, low current draw and tremendous support in the hobbyist electronics world. The microcontroller was programmed in C and contains the logic/software for taking measurements, displaying and logging data, implementing the tare and calibration requirements and monitoring for bird presence. The clock speed was reduced to 4Mhz instead of the default 16Mhz to reduce power consumption by 50 milliwatts. The power LED was also removed to reduce power by 30 milliwatts.

A green circuit board with many pins

Description automatically generated

**Transducer.** A 20 Kg strain gauge transducer was utilized to convert the bird mass into an electrical signal that is measured by the analog to digital converter and then converted into weight. As the bird mass increases, so does the differential voltage developed by the Wheatstone bridge of the transducer. Provisions were made for the additional force realized at the moment of a bird landing. The transducer selected is rated for a maximum force of 1.5x the full scale reading of 20 Kg. That 30 Kg rating is more than double the largest expected bird. Ideally a 10 Kg transducer would be used to improve scale resolution but the tradeoff was determined to be acceptable and necessary.

A silver rectangular object with wires

Description automatically generated

**Differential Analog to Digital Converter (ADC)**. The HX711 ADC IC selected amplifies the signal from the transducer and converts it to a 24 bit digital value. This digital value is read by the Arduino and converted to a weight in kilograms. The known calibration weight provides the scale factor needed to convert the 24 bit value to a weight in kilograms.

**Four digit display.** A 4-digit display was utilized to show the scale measurement in kilograms. The 4 digits allowed a format of XX.XX Kg. The display was also used for the calibration process and to set the time and date. The display is only active if a bird with a mass greater than 0.5 Kg is present on the perch. One of the requirements was that the scale display could be captured by a trail camera. This requirement drove the selection of a display that doesn’t refresh or scan. Trail camera shutter speed will reduce display visibility if it is a refreshing display. The initial prototype utilized a display with a 120Hz refresh rate, this caused some digits to not show up in the pictures when the camera shutter speed was greater 1/120th of a second.

A close-up of a digital timer

Description automatically generated

**Non-volatile memory (NVM).** NVM on the Arduino Nano module was critical to store the calibration values when power shuts off or the battery completely discharges. It can also be used to store a scale serial number.

**Real time clock (RTC).** A real time clock was included in the design so that all measurements could be logged with a date and time for correlation to trail camera pictures of the perch. The RTC requires a small coin cell battery to retain time when power is off to the scale. This module was also able to handle Daylight Saving Time adjustments that occur twice during annual scale usage.

**Power Supply**. The power supply consists of three components. The solar charge controller directs power to the battery and to the electronic scale. It automatically switches between battery power and solar power based on sunlight levels. The rechargeable lithium-ion battery allows the scale to operate at night and during periods of limited sunlight. The scale consumes about 80 milliwatts of power, therefore the 37 watt-hour battery can support 17days of operation without the solar panel. The solar panel was added to extend battery life. For climates with moderate sunlight, the system can sustain operation indefinitely. In other environments, recharging the battery with an external source is required in the winter months. The solar panel is a 2 watt panel. It is mounted to the top of the electronic scale enclosure. With a 2 watt panel, it takes approximately 1.5 hours of full sunlight to completely replenish 1 day of scale usage.

A solar panel with a battery and a blue battery

Description automatically generated

Solar charge controller

Battery

Solar Panel

**SD Card.** The scale design included a micro-SD card. The SD card provides a method for data to be stored of each bird event and to also store hourly scale measurements. These hourly “periodic” recordings help with monitoring temperature drift of the measurements . All records are stored with a time and date stamp. An example is shown below. When a bird > 0.5 Kg is present, the scale logs measured weight every 10 seconds. An SD card greater than 1 gigabyte in size can store more than a year of data.

A table with numbers and letters

Description automatically generated

Periodic measurements logged every hour.

Bird on the scale – 3.54 Kg

**Printed Circuit board (PCB)**. A PCB was designed with KiCad to connect all the components together in a reliable and organized implementation. The PCB significantly reduces build time of the scale and improves robustness. The circuit board was designed to securely slot into the enclosure.

A red circuit board with many different components

Description automatically generated

**Software**. All code was developed and compiled using the Arduino IDE. Some features of the software include reading the scale weight from the ADC, converting the ADC value into a weight, monitoring for a weight above 0.5 Kg, displaying the weight, logging to the SD card (time, date, weight) and handling the calibration process. The software also includes an algorithm for monitoring the no-load weight to adjust for a drifting tare point. The software applies some filtering to the measured scale reading to reduce the impact of bird movement.

**Perch and transducer assembly.** The perch and transducer assembly were topped with a 1.5 inch thick by 6 inch octagon cedar block mounted to an aluminum L-bracket. That L-bracket was attached to the load end of the transducer. The fixed end of the transducer was secured to a z-bracket. The z-bracket incorporates a strain relief for the 20 foot cable to the scale. The z-bracket is attached to a post bracket that can be easily secured to any 4” x 4” post with 4-6 screws.

A piece of wood with a metal plate

Description automatically generatedA bird perched on a post

Description automatically generated

**Scale electronics and enclosure.** An IP65 enclosure was utilized to protect the electronics and to mount the solar panel. A handle was attached which provided a feature to clamp the scale to a post or stake. Two connectors, one for the solar panel and one for the perch transducer cable, were installed on the right side of the enclosure. Above the two connectors are two switches. One is to control scale power and the other switch is a Tare/Cal switch. Pressing and holding this switch enters calibration mode.

A black box with red numbers and wires

Description automatically generated

Weight in kilograms

Switches

Solar panel

Transducer connection

Solar panel connection

**Calibration and tare.** The scale automatically tares when it is powered on. Tare is required to eliminate the weight of the L-bracket and wood perch that apply a constant force to the transducer. The scale learns this offset and subtracts it from the measured weight. The scale software also includes a dynamic tare routine which will adjust the tare as it changes over time due to snow, rain or temperature.

Calibration can be performed by holding the Tare/Cal switch for 5 seconds. After 5 seconds, the scale will ask for a known 3.5 Kg weight to be placed. Once it is placed on the perch, the Cal/Tare button is pressed again, and the scale calibrates. Calibration can be confirmed by placing the known 3.5 Kg weight on the perch again and verifying the scale reading.

**System cost.** The engineering and design of the entire system required 180 hours. This time includes feasibility, design and debug of the system. Each individual system costs $325 in parts and each scale takes approximately 7-8 hours to build. Below is a complete bill of materials.

A screenshot of a computer screen

Description automatically generated

**Conclusion:** After a few months of usage, the scale seems to meet the overall goals and expectations to passively acquire eagle weights. The perch design has been accepted by the eagles as is evident by the frequency of birds perching. The initial data indicates accurate measurements can be acquired and correlated to trail camera pictures. There are still opportunities for improvement which are listed below.

1. Reduce the power consumption below 80milli-watts. If the scale is installed in shade or used in areas with less seasonal sunlight, the scale will run out of energy.
2. Improve the method for extracting data from the scale. Incorporating a WiFi hot spot in the scale is being considered so that a technician could download data using a smartphone or laptop. The design currently requires the removal of four screws and the extraction of the micro-SD card.
3. Implementing a method to indicate low battery condition will be helpful for troubleshooting issues.
4. Installation of a GPS receiver could be beneficial for more accurate time, date and location.

**Design Documentation**

PCB design & schematic

A computer screen shot of a circuit board

Description automatically generated

|  |  |
| --- | --- |
| A computer circuit board with many small points  Description automatically generated | A green circuit board with many small components  Description automatically generated |

Enclosure wiring

Perch Top Design

Software flowchart

A diagram of a machine

Description automatically generated

A diagram of a flowchart

Description automatically generated

A diagram of process flow

Description automatically generated

Firmware/Code for the Nano (Arduino)

// Revision Log

// 1.0 Initial Revision

// 2.0 Shipped September 2023

// 2.1 12/3/2023  - Fixed negative temperature logging bug

// 3.0 12/28/2023 - Add dynamic tare adjustment, additional logging and an event file

// 3.1 2/1/2024 - Added code to store a summary file

// 3.2 3/1/2024 - Reduced scale sampling / averaging.  Shut off display after 1 minute

// 4.0 12/29/2024 - Rewrite to clean up logic and add Welfords algorithm

// THINGS TO REMEMBER

// KEEP THE dynamic memory above 520 bytes free or so to make sure we don't run out of memory for local functions - you will get strange behavior with sd.h library

#include <SD.h>       //SD Card library - SD by Arduinu,Sparkfun - 1.3.0

#include "HX711.h"    //HX711 library - HX711 Arduino library by Bogdan 0.7.5

//#include <Wire.h>     //I2C library - I don't think we need this

#include "EEPROM.h"   //library for built in nano eeprom

#include <SPI.h>      //used by SD Card interface - SPI library

#include "RTClib.h"   //real time clock library - RTClib by Adafruit 2.1.4

#include "DST\_RTC.h"  //daylight savings time adjustment library - DST RTC by Andy Doro 1.1.1

// HX711 circuit wiring

HX711 scale;

const int LOADCELL\_DOUT\_PIN = 2;

const int LOADCELL\_SCK\_PIN = 3;

//## How to calibrate your load cell

//1. Call `set\_scale()` with no parameter.

//2. Call `tare()` with no parameter.

//3. Place a known weight on the scale and call `get\_units(10)`.

//4. Divide the result in step 3 to your known weight. You should

//   get about the parameter you need to pass to `set\_scale()`.

//5. Adjust the parameter in step 4 until you get an accurate reading.

// Details on the ADC/Transducer

// 24 bit ADC that is formated 2's complement - +/-20mV range for +/-8,388,608 counts on the ADC

// 10 kg is about 1,150,000 counts with the specified design

// 6.5 kg is about 709,000 counts with the specified design

// 4.0 kg is about 440,000 counts with the specified design

// or 114,855 counts per kg with a 20kg sensor

//Globals for this project

unsigned long two\_sec\_timer, one\_hour\_timer, event\_timer;

double current\_reading, last\_reading, two\_sec\_reading, averaged\_reading, saved\_tare, w\_avg, w\_max, w\_min, w\_stdev, w\_m2;

int n; //number of samples used in welfords algorithm

bool birdpresent; //true = if scale senses value above weight\_threshold

bool finalevent;  //store data 5 seconds after departure

int birdpresentcounter; // how long has he/she been present

int button\_ctr = 0;  //counter for 4 second button press

// CONFIGURATIONS

const double weight\_threshold = 0.5;  //theshold to trigger scale to bird presence in kg

const double cal\_weight = 3.50;  //weight in kilograms of calibration standard -add 0.005 for truncation instead of rounding

//IO Pins

const int CALbuttonPin = 4; //input pin for the tare/cal button

const int CLKScaler = 0x02; //scaler is a power so 2 is 1/4th the 16Mhz frequency

const int CHGSTATPin = 5;

const int PGOODPin = 6;

const int BATTERYVOLTPin = A7;

const int DISPLAY\_PWR = A1;

//74HC595 Shift register based display (STATIC DISPLAY without refresh or scan rate concerns)

static const uint8\_t numbers[10] =               // 7 segment values for decimals 0..9

{

//TRUTH TABLE    |   0 = segment on

//ABCD EFGH      |   1 = segment off

B00000011,  //0  |        A

B10011111,  //1  |      -----

B00100101,  //2  |   F |     | B

B00001101,  //3  |     |  G  |                                       YEAr B3, 61, 11, F5  1f

B10011001,  //4  |      -----

B01001001,  //5  |   E |     | C

B01000001,  //6  |     |     |

B00011111,  //7  |      -----

B00000001,  //8  |        D

B00011001   //9  |

};

//Pin connected to ST\_CP of 74HC595

int latchPin = 9;

//Pin connected to SH\_CP of 74HC595

int clockPin = 7;

////Pin connected to DS of 74HC595

int dataPin = 8;

//SD CARD CONFIGS

/\*

  SD card attached to SPI bus as follows:

 \*\* MOSI - pin 11

 \*\* MISO - pin 12

 \*\* CLK  - pin 13

 \*\* CS   - pin 10

\*/

static const int chipSelect = 10;

bool SDpresent;

//real time clock object creation

RTC\_DS3231 rtc;

DST\_RTC dst\_rtc; // DST object

const char rulesDST[] = "US"; // US DST rules

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN LOOP OF THE APPLICATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

void loop() {

  // Check for button press - initiate calibration process if pressed and held

  CheckForCalButton();

  //Read the scale's transducer

  ReadSensor();

  //Process sensor measurement if bird present

  ProcessMeasurement();

  //Process 2 second operation

  TwoSecondOp();

  //Process 1 hour operation

  OneHourOp();

  //Update display

  UpdateDisplay();

  //loop rate

  delay(200/pow(2,CLKScaler));

}

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MAIN LOOP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

void ReadSensor()

{

  //Read the scale's transducer if it is ready

  if (scale.is\_ready())   // check if HX711 is ready - DOUT == LOW

  {

    current\_reading = scale.get\_units(5);  //this call retrieves atare adjusted and scaled weight - average of 5 readings

    averaged\_reading = 0.5 \* current\_reading + 0.5 \* averaged\_reading;

  } else

  {

    //Serial.println("HX711 not found.");

  }

}

void ProcessMeasurement()

{

  if(averaged\_reading - two\_sec\_reading > weight\_threshold) //weight above threshold?  //two\_sec\_reading stops updating if bird present

  {

    if(birdpresent == false)  //initial landing

    {

      birdpresent = true;

      event\_timer = millis();  //reset event timer to now

      WriteSummaryFile(0,0,0,0, true);  // write opening line of landing

      saved\_tare = two\_sec\_reading;

      Write2Values(current\_reading, saved\_tare, "Trigger and prelanding");

      birdpresentcounter = 1;

      //reset max, min and avg for welfords algo

      w\_max=w\_min=w\_avg=w\_stdev=0;

    }

    if (millis() - event\_timer > 15000/pow(2,CLKScaler)) // 15 seconds?

    {

      ProcessData(current\_reading);

      birdpresentcounter++;

      event\_timer = millis(); // reset timer for next event

      if(birdpresentcounter%2 == 0)  //log 15 seconds after landing and then every 30 seconds

      {

        Write2Values(current\_reading, current\_reading - saved\_tare, "30 Sec");

      }

    }

  }else

  {

    if(birdpresent == true)

    {

      birdpresent = false;

      WriteSummaryFile(w\_max, w\_min, w\_avg, w\_stdev, false);  // write welford data and date/time as bird departs

      event\_timer = millis();

      finalevent = false;

    }

    if((birdpresent == false) && (finalevent == false))

    {

      two\_sec\_timer = millis(); // keep resetting this until post weight is stored

      if(millis() - event\_timer > 5000/pow(2,CLKScaler))

      {

        finalevent = true;

        Write2Values(current\_reading, current\_reading - saved\_tare, "Post departure weight");

      }

    }

  }

}

void ProcessData(float datapoint)  //Welfords algorithm

{

  float delta, delta2, variance;

  if(w\_avg == 0)

  {

    n = 0;

    w\_max = w\_min = datapoint;

  }

  n++;

  delta = datapoint - w\_avg;

  w\_avg += delta/n;

  delta2 = datapoint- w\_avg;

  w\_m2 += delta \* delta2;

  variance = w\_m2/(n-1);

  w\_stdev = sqrt(variance);

  if(datapoint > w\_max)

  {

    w\_max = datapoint;

  }

  if(datapoint < w\_min)

  {

    w\_min = datapoint;

  }

}

void TwoSecondOp()  //only do this if no bird and 2 seconds has elapsed

{

  if(birdpresent == false && millis() - two\_sec\_timer > 2000/pow(2,CLKScaler))  // only do this if no bird present

  {

    two\_sec\_reading = last\_reading;

    last\_reading = current\_reading;

    two\_sec\_timer = millis();  //reset timer

  }

}

void OneHourOp()

{

  if(birdpresent == false && millis() - one\_hour\_timer > 3600000/pow(2,CLKScaler))  //only do this if no bird and 1 hour elapsed

  {

      Write2Values(current\_reading, averaged\_reading, "Periodic");

      one\_hour\_timer = millis();  //reset timer

  }

}

void UpdateDisplay()

{

  if((birdpresent == true && birdpresentcounter < 6) || (birdpresent == false && millis() - event\_timer < 10000/pow(2,CLKScaler)))

  {

    displayNumber2D(current\_reading - saved\_tare + 0.005);

  }else

  {

    displayBlank();

  }

}

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SETUP CODE - CALLED FIRST BEFORE LOOP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

void setup() {

  //initialize clock slower to save power

  CLKPR = 0x80;

  CLKPR = CLKScaler;  //Set clock prescaler

  //initialize IO pins

    //Display IO

  pinMode(latchPin, OUTPUT);

  pinMode(clockPin, OUTPUT);

  pinMode(dataPin, OUTPUT);

  pinMode(DISPLAY\_PWR, OUTPUT);

     //Charger IO

  pinMode(PGOODPin, INPUT);

  pinMode(CHGSTATPin, INPUT);

     //Button IO

  pinMode(CALbuttonPin, INPUT\_PULLUP);

  //Serial.begin(38400);  //setup serial debug port

  //initialize variables

  int setupbuttonstate = 0;

  current\_reading = last\_reading = two\_sec\_reading = 0;  // start off with all trackers at zero

  float  startup\_cal\_value;

  saved\_tare = 0;

  birdpresent = false;  //preset to no bird present

  finalevent = true;  // present to having processed final event

  //Display revision

  digitalWrite(DISPLAY\_PWR, LOW); //Turn on display

  DisplayItOnLED(0x91, 0x9f, 0XEf, 0x99);  // "HI\_4"  last digit is the revision

  delay(1000/pow(2,CLKScaler));  //let it show on the display

  //Lithium Ion Polymer Battery - 3.7V 10050mAh

  //Display battery level - Power supply is set to 4.9 volts so 1024 = 4.9. Battery 100% = 4.1V, Battery 1% =2.8V

  double battLevel = ((analogRead(BATTERYVOLTPin) \* 4.90/1024 - 2.8) \* 0.77);

  if (battLevel > 1)

  { battLevel = 1;}

  if (battLevel < 0.01)

  { battLevel =0.01;}

  displayNumber2D(battLevel);

  //check if Real time clock connected and talking

  if (! rtc.begin())

  {

   // Serial.println("Couldn't find RTC");

    DisplayItOnLED(0xD5, 0xC5, 0x73, 0XE1);  // "nort"

    delay(5000/pow(2,CLKScaler));  //let it show on the display

  }

  //check if power on with tare/cal button held - enter time/date setting

  setupbuttonstate = digitalRead(CALbuttonPin);

  if (setupbuttonstate == LOW)

  {

    delay(1000/pow(2,CLKScaler));

    if (setupbuttonstate == LOW)

    {

      delay(1000/pow(2,CLKScaler));

      if (setupbuttonstate == LOW)

      {

        SetDateAndTime();

      }

    }

  }

  //Serial.println("Initializing the HX711 sensor");

  // Initialize library with data output pin, clock input pin and gain factor.

  // Channel selection is made by passing the appropriate gain:

  // - With a gain factor of 64 or 128, channel A is selected

  // - With a gain factor of 32, channel B is selected

  // By omitting the gain factor parameter, the library

  // default "128" (Channel A) is used here.

  scale.begin(LOADCELL\_DOUT\_PIN, LOADCELL\_SCK\_PIN);

  //Serial.println("Before setting up the scale:");

  //Serial.print("read: \t\t");

  //Serial.println(scale.read());      // print a raw reading from the ADC

  scale.read();

  //Serial.print("read average: \t\t");

  //Serial.println(scale.read\_average(20));   // print the average of 20 readings from the ADC

  scale.read\_average(20);

  //Serial.print("get value: \t\t");

  //Serial.println(scale.get\_value(5));   // print the average of 5 readings from the ADC minus the tare weight (not set yet)

  scale.get\_value(5);

  //Serial.print("get units: \t\t");

  //Serial.println(scale.get\_units(5), 1);  // print the average of 5 readings from the ADC minus tare weight (not set) divided

  scale.get\_units(5);

  //read external eeprom - 0x55 present if the EEPROM has been iniatilized and a cal value stored

  uint8\_t eeprom\_valid = EEPROM.read(0);

  if(eeprom\_valid == 0x55)  //configured previously if =0x55

  {

    EEPROM.get(1, startup\_cal\_value);

    //Serial.print("Read cal value from EEPROM: ");

    //Serial.println(startup\_cal\_value, 8);

  }else

  {

    startup\_cal\_value = -980; //default value to use based on 8 scales and 3.5 Kg

  }

  //Setup scale cal factor

  scale.set\_scale(startup\_cal\_value);   //Use EEPROM value if found, otherwise use default value with variable definition

  delay(1000/pow(2,CLKScaler));

  //Perform scale tare operation after each power cycle

  InitialTare();

  delay(1000/pow(2,CLKScaler));

  //Initialize the SD Card - see if the card is present and can be initialized:

  //Serial.print("Initializing SD card...");

  SDpresent = true;

  if (!SD.begin(chipSelect)) {

    //Serial.println("Card failed, or not present");

    SDpresent = false;

    DisplayItOnLED(0xD5, 0xC5, 0x49, 0X85);  // "noSd"

    delay(5000/pow(2,CLKScaler));

  }

  //Serial.println("card initialized.");

  //setup the RTC

  /\*if (rtc.lostPower()) {

    Serial.println("RTC lost power, let's set the time!");

    // When time needs to be set on a new device, or after a power loss, the

    // following line sets the RTC to the date & time this sketch was compiled

    rtc.adjust(DateTime(F(\_\_DATE\_\_), F(\_\_TIME\_\_)));

    // This line sets the RTC with an explicit date & time, for example to set

    // January 21, 2014 at 3am you would call:

    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));

  }

\*/

  if (rtc.lostPower()) {

    //Serial.println("RTC is NOT running!");

    // following line sets the RTC to the date & time this sketch was compiled

    rtc.adjust(DateTime(\_\_DATE\_\_, \_\_TIME\_\_));

    //rtc.adjust(DateTime(2023, 1, 2, 13, 15, 0)); //set for SLC time

    // DST? If we're in it, let's subtract an hour from the RTC time to keep our DST calculation correct. This gives us

    // Standard Time which our DST check will add an hour back to if we're in DST.

    DateTime standardTime = rtc.now();

    if (dst\_rtc.checkDST(standardTime) == true) { // check whether we're in DST right now. If we are, subtract an hour.

      standardTime = standardTime.unixtime() - 3600;

    }

    rtc.adjust(standardTime);

  }

  //read, format and display time at power on

  DateTime standardTime = rtc.now();

  DateTime theTime = dst\_rtc.calculateTime(standardTime); // takes into account DST

  double timetemp = theTime.hour();

  timetemp += (double)theTime.minute()/100.0;

  displayNumber2D(timetemp);

  delay(1000/pow(2,CLKScaler));

  //Store header at power on

  WriteSDFile(false, "Date,Time,Int Temp,Battery,Raw Reading,Weight(Kg),Notes");

  char str[70];

  strcpy(str, "0,0,Powered on using calibration: ");

  dtostrf(startup\_cal\_value, 7, 2, &str[strlen(str)]);

  strcpy(&str[strlen(str)], " offset: ");

  ltoa(scale.get\_offset(), &str[strlen(str)], 10);

  WriteSDFile(true, str);

  WriteSummaryFile(0,0,0,0, false);  // insert blank line to indicate power on

  //initialize timers

  two\_sec\_timer = one\_hour\_timer = millis();

}

void CheckForCalButton(void)

{

  if (digitalRead(CALbuttonPin) == LOW)

  {

    button\_ctr++;

    if(button\_ctr > 3)  //3 loops through main with the button pressed will fire the calibration routine

    {

      button\_ctr = 0;

      PushbuttonCalibration();

    }

  }else

  {

    button\_ctr = 0;

  }

}

void Write2Values(double raw, double adjusted, String stufftowrite)

{

    String localString;

    localString = String(raw, 2);

    localString.concat(",");

    localString.concat(String(adjusted, 2));

    localString.concat(",");

    localString.concat(stufftowrite);

    WriteSDFile(true, localString);

}

// Write log file data - make sure there are 512bytes or more for local variables

void WriteSDFile(bool dataprefix, String stufftowrite)

{

  DateTime standardtime = rtc.now();

  DateTime theTime = dst\_rtc.calculateTime(standardtime); // takes into account DST

  // if the file is available, write to it:

  File dataFile = SD.open ("datalog.csv", FILE\_WRITE);

  if (dataFile)

  {

    char c[10];

    if (dataprefix == true)

    {

      sprintf(c, "%02d", theTime.month());    // build integer string using C integer formatters  (m is length, and not used in this code)

      dataFile.print(c);

      dataFile.print('/');

      sprintf(c, "%02d", theTime.day());

      dataFile.print(c);

      dataFile.print('/');

      dataFile.print(theTime.year(), DEC);

      dataFile.print(",");

      sprintf(c, "%02d", theTime.hour());

      dataFile.print(c);

      dataFile.print(':');

      sprintf(c, "%02d", theTime.minute());

      dataFile.print(c);

      dataFile.print(':');

      sprintf(c, "%02d", theTime.second());

      dataFile.print(c);

      dataFile.print(",");

      double temp\_temp = rtc.getTemperature();

      if (temp\_temp > 127)

      {

        temp\_temp = temp\_temp -256;

      }

      dataFile.print(temp\_temp);

      dataFile.print(",");

      dataFile.print(analogRead(BATTERYVOLTPin) \* 4.90/1024);

      //dataFile.print(",");

      //dataFile.print(digitalRead(PGOODPin)==LOW);

      //dataFile.print(",");

      //dataFile.print(digitalRead(CHGSTATPin)==LOW);

      dataFile.print(",");

    }

    dataFile.println(stufftowrite);

    dataFile.close();

  }

}

void WriteSummaryFile(double max, double min, double weight, double stdev, bool landing)

{

  DateTime standardtime = rtc.now();

  DateTime theTime = dst\_rtc.calculateTime(standardtime); // takes into account DST

  // if the file is available, write to it:

  File dataFile = SD.open ("summary.csv", FILE\_WRITE);

  if (dataFile)

  {

    char c[10];

    sprintf(c, "%02d", theTime.month());    // build integer string using C integer formatters  (m is length, and not used in this code)

    dataFile.print(c);

    dataFile.print('/');

    sprintf(c, "%02d", theTime.day());

    dataFile.print(c);

    dataFile.print('/');

    dataFile.print(theTime.year(), DEC);

    dataFile.print(",");

    sprintf(c, "%02d", theTime.hour());

    dataFile.print(c);

    dataFile.print(':');

    sprintf(c, "%02d", theTime.minute());

    dataFile.print(c);

    dataFile.print(':');

    sprintf(c, "%02d", theTime.second());

    dataFile.print(c);

    dataFile.print(",");

    if (landing == false)  //only write weight on departure

    {

      dataFile.println(weight);

    }

    dataFile.close();

  }

}

bool InitialTare(void)

{

  DisplayItOnLED(0xE1, 0x11, 0x73, 0X61);  //"tARE"

  delay(1000/pow(2,CLKScaler));

  scale.tare();

  current\_reading = last\_reading = two\_sec\_reading = 0;  // start off with all drift trackers at zero

  return(true);

}

//Initiated by holding CAL button for 3 seconds

bool PushbuttonCalibration(void)

{

  int buttonstate = 0;

  float w\_counts, cal\_value;

  scale.set\_scale();

  InitialTare();

  delay(1000/pow(2,CLKScaler));

  //tell user to hang weight (CAL message)

//  matrix.println("CAL");

//  matrix.writeDisplay();

  DisplayItOnLED(0x63, 0x11, 0xE3, 0xFF);  //Display CAL

  //wait for button press to acknowledge weight is placed

  while(buttonstate == LOW)

  {

    buttonstate = digitalRead(CALbuttonPin);

  }

  //wait for button release

  while(buttonstate == HIGH)

  {

    buttonstate = digitalRead(CALbuttonPin);

  }

  w\_counts = scale.get\_units(10);

  cal\_value = w\_counts/cal\_weight;

  //cal\_value = cal\_value/100; //do this to give us an int we can display with 2 digits

  scale.set\_scale(cal\_value);

  // tell the user it is calibrated

  //matrix.println("CALD");

  //matrix.writeDisplay();

  DisplayItOnLED(0x63, 0x11, 0xE3, 0x85);  //Display CALd

  //Serial.println("Writing calibration factor to address 0x01");

  //Serial.println(cal\_value);

  //store value to External EEProm

  //uint8\_t buffer[4];  // floats are 4 bytes!

  //memcpy(buffer, (void \*)&cal\_value, 4);

  //EEPROM.write(0x01, buffer, 4);

  //store value to INternal EEPROM

  EEPROM.put(1, cal\_value);

  delay(1000/pow(2,CLKScaler));

  //i2ceeprom.write(0x0, 0x55);  //write byte to indicate EEProm has valid calibration data

  EEPROM.write(0,0x55); //Write byte to indicate EEPROM has valid calibration data for use at power up

  delay(2000/pow(2,CLKScaler));

  char str[70];

  strcpy(str, "Calibration completed. New value: ");

  dtostrf(cal\_value, 8, 2, &str[strlen(str)]);

  strcpy(&str[strlen(str)], " offset: ");

  ltoa(scale.get\_offset(), &str[strlen(str)], 10);

  WriteSDFile(true, str);

  return true;

}

//SHIFT REGISTER BASED DISPLAY (no scanning or refresh rate - sucking a lot of power though)

void DisplayItOnLED(int digit1, int digit2, int digit3, int digit4)

{

 // send out data

  digitalWrite(latchPin, LOW);

  shiftOut(dataPin, clockPin, LSBFIRST, digit4);

  shiftOut(dataPin, clockPin, LSBFIRST, digit3);

  shiftOut(dataPin, clockPin, LSBFIRST, digit2);

  shiftOut(dataPin, clockPin, LSBFIRST, digit1);

  digitalWrite(latchPin, HIGH);

}

void displayError(void)

{

  DisplayItOnLED(0xFD, 0xFD, 0xFD, 0XFD);  // four dashes

}

void displayBlank(void)

{

  DisplayItOnLED(0xFF, 0xFF, 0xFF, 0XFF);  //blank screen

}

//convert a number to a stream to shift out to the seven segment display

bool displayNumber2D(double number)

{

  int temp, digit1, digit2, digit3, digit4; //displayed left to right in that order

  int tempsum;

  int intnumber;

  //Serial.println(number);

  intnumber = number \* 100; //get rid of decimal places and work in that arena

  if(abs(intnumber) < 9999)

  {

    if(abs(intnumber) < 1000)

    {

      temp = 0;  //1st digit is zero (also known as blank)

      if(intnumber == abs(intnumber))

      {

        digit1 = 255; //blank display

      }else

      {

        if(intnumber < -1000) //not enough digits on the display

        {

          //Serial.println("number too negative");

          displayError();

          return(false);

        }

        digit1 = B11111101;  //show negative symbol

        intnumber = abs(intnumber);  //show negative and treat like positive

      }

    }else

    {

      temp = intnumber/1000;

      digit1 = numbers[temp];

    }

    tempsum = temp \* 1000;

    temp = intnumber - tempsum;

    temp = temp /100;

    digit2 = numbers[temp] & 0XFE; //and on the decimal point

    tempsum = tempsum + temp\*100;

    temp = intnumber - tempsum;

    temp = temp/10;

    digit3 = numbers[temp];

    tempsum = tempsum + temp \* 10;

    temp = intnumber - tempsum;

    digit4 = numbers[temp];

    DisplayItOnLED(digit1, digit2, digit3, digit4);

    return(true);

  }else

  {

    //Serial.println("number too large");

    displayError();

    return(false);

  }

}

void SetDateAndTime(void)

{

    int month, day, year, minute, hour;

    DisplayItOnLED(0x49, 0xE1, 0x83, 0X31);  // "StUP"

    delay(5000/pow(2,CLKScaler));

    while(digitalRead(CALbuttonPin) == LOW){}  //Wait for button release

    DateTime standardtime = rtc.now();

    DateTime theTime = dst\_rtc.calculateTime(standardtime); // takes into account DST

    month = theTime.month();

    day = theTime.day();

    year = theTime.year() - 2000;

    hour = theTime.hour();

    minute = theTime.minute();

    //Get hour = d1,c5, c7, f5

    DisplayItOnLED(0xD1, 0xC5, 0xC7, 0xF5);

    delay(2500/pow(2,CLKScaler));

    hour = GetValueFromUser(hour, 0,23);

    //Get min =

    DisplayItOnLED(0x13, 0x13, 0xF7, 0xD5);

    while(digitalRead(CALbuttonPin) == LOW){}  //Wait for button release from previous

    delay(2500/pow(2,CLKScaler));

    minute = GetValueFromUser(minute, 0,59);

    // Get month =

    DisplayItOnLED(0x13, 0x13, 0xC5, 0xD5);

    while(digitalRead(CALbuttonPin) == LOW){}  //Wait for button release

    delay(2500/pow(2,CLKScaler));

    month = GetValueFromUser(month, 1,12);

    //Get day = ff, 85, 11, B1

    DisplayItOnLED(0xFF, 0x85, 0x11, 0xB1);

    while(digitalRead(CALbuttonPin) == LOW){}  //Wait for button release

    delay(2500/pow(2,CLKScaler));

    day = GetValueFromUser(day, 1,31);

    //Get year = YEAr B1, 61, 11, F5

    DisplayItOnLED(0xB1, 0x61, 0x11, 0xF5);

    while(digitalRead(CALbuttonPin) == LOW){}  //Wait for button release

    delay(2500/pow(2,CLKScaler));

    year = GetValueFromUser(year, 23,43) + 2000;

   if (dst\_rtc.checkDST(standardtime) == true) { // check whether we're in DST right now. If we are, subtract an hour since the adjust for DST will add an hour if we are in DST

      if(hour > 0)

      {

        hour--; //This won't work well if set between midnight and 1AM - don't do that

      }else

      {

        hour = 23;

      }

    }

    rtc.adjust(DateTime(year, month, day, hour, minute, 30));

}

int GetValueFromUser(int varval, int lowerbound, int upperbound)

{

    unsigned long currentMillis = millis();

    unsigned long btnholdtimer;

    int loopcounter = 0;

    while(millis() - currentMillis < 70000/pow(2,CLKScaler))  //give on minute per value

    {

      displayNumber2D(varval);

      btnholdtimer = millis();

      while(digitalRead(CALbuttonPin) == LOW)

      {

         delay(100/pow(2,CLKScaler)); //little bit of debounce

         loopcounter = 1;

         if(millis() - btnholdtimer > 3000/pow(2,CLKScaler))

         {

           return varval;

         }

      }

      if(loopcounter > 0)

      {

        loopcounter = 0;

        varval++;

        if (varval > upperbound)

        {

          varval = lowerbound;

        }

      }

    }

}