CS 431: Weight Scale Project

Ivan Johnson Soham Karanjikar

2020-05-01

1 Introduction

The main motivating factor behind this project was to make a scale that could take the weight readings and upload them the data to our `private servers`. As much of this is beyond the scope of what was taught in class, however, for the purposes of this class the project is limited to the much simpler goal of simply measuring and displaying the force that is applied to a sensor.

Besides the learning experience of building the system ourselves, the main benefit of this approach over buying a standard smart scale is that it gives us complete control of the system. This alleviates many privacy concerns and allows for the possibility of integrating the scale with arbitrary `health tracking systems`.

This was a non-trivial task requiring that we research the construction of commercial bathroom scales, choose the most practical parts for our own design, build then test the design, make the software to run the system, and finally optimize the software to minimize power usage. This process enhanced our learning experience in the course by giving us a more hands-on experience with how embedded systems are designed and used.

The initial plan was to use an existing scale and try to take readings through communicating with the micro controller on the scale, however, this proved to be more work than creating a new scale. The components, procedure, results, limitations, and potential improvements are explained in this paper. The appendices contain our original proposal, our code, and a list of what we used for this project.

2 Components

2.1 Microcontroller

The board used was an Arduino Uno, a hobbyist development board used in many projects at their preliminary stage, with the MCU being an ATMEGA4809. The reason for choosing this board was because of the ease of using it and the libraries already available for communicating with various chips. Furthermore, the Arduino IDE is also very user friendly and allowed us to quickly test code. We could have potentially used a cheaper and smaller option such as an ESP32, but there were Arduinos already present in the lab which allowed us to start immediately.

2.2 Load Cells and Amplifier Chip

While researching methods to communicate with a controller on an existing scale we found that load cells and their controller chip are very cheap and easy to use. This is the reason we decided to build our own scale rather than tapping into a pre-made scale.

The chip used for reading values from load cells is the HX711, an amplifier chip specifically designed for load cells.

The bundle purchased came with four load cells included in it. These load cells had an individual rating of 50Kg each allowing us to make a scale that could provide accurate readings of masses up to 200Kg. The load cells were generic and came with no model/serial number or brand.

3 Procedure

3.1 Wiring

Load cells use strain gauges to measure weight. A strain gauge contains a wire carefully shaped such that its resistance varies when the strain gauge undergoes elastic deformation. To form the load cell, the strain gauge is attached to a piece of metal in such a way that the strain gauge can be used to measure the deformation of the metal and infer the force that is being applied to the load cell.

The load cells are wired according to the specification given to use in a Wheatstone Bridge configuration, as shown in figure 1.

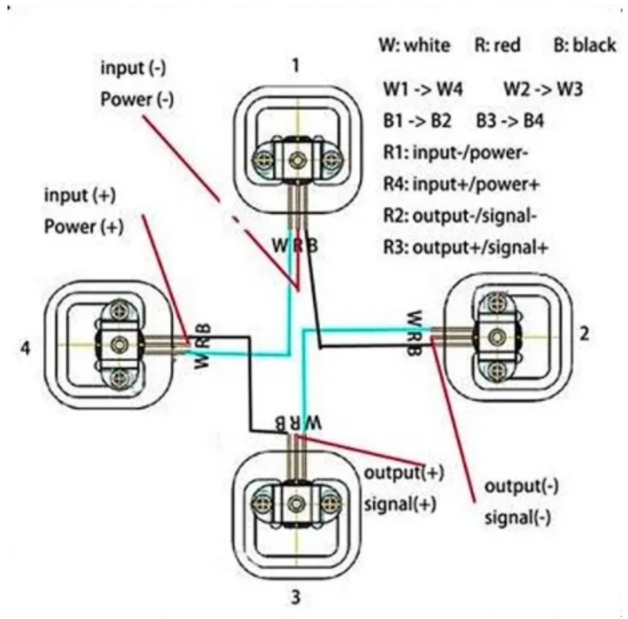


Figure 1: Image obtained from Bang Good

Since the change in resistance is quite small, we a HX711 chip to amplify the change. This chip also has ADC included in it which allows us to directly communicate with it using the serial protocol given in its documentation. The whole system's wiring diagram is provided in figure 2.

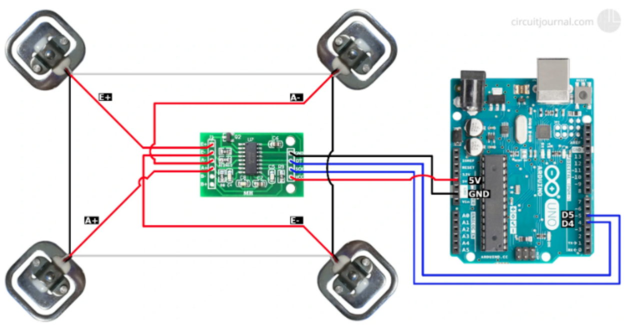


Figure 2: Image obtained from Circuit Journal

The Arduino uses its 5V out to power the HX711 chip and two GPIOs to handle the communication, one for clock and one for data.

3.2 Software

All the code was written in the Arduino IDE as a sketch; the code is available in section B of the appendix. We only had one external dependency, a `HX711 library`. This library allowed us to communicate with the amplifier chip quite easily as the low level communication was already written for us. Figure 3 shows a state diagram of our software.

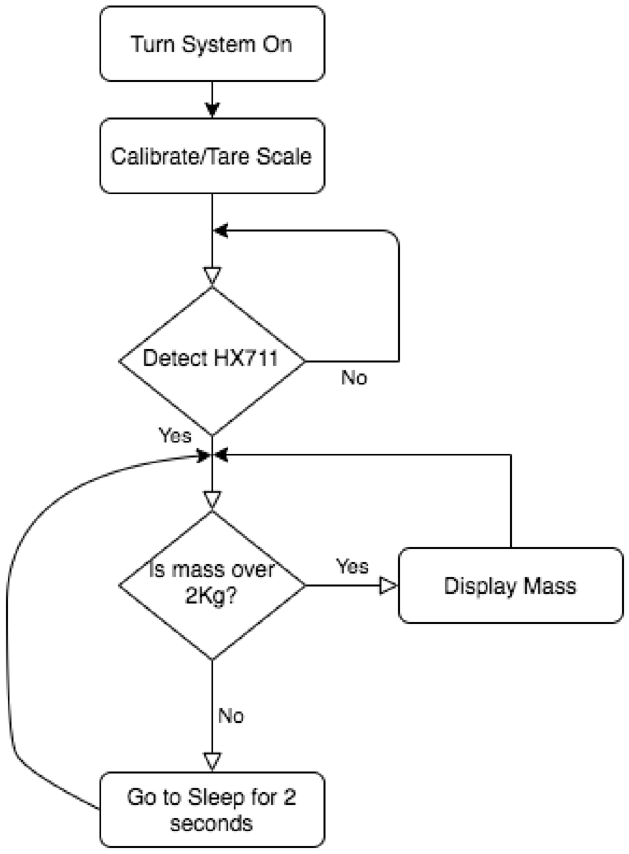


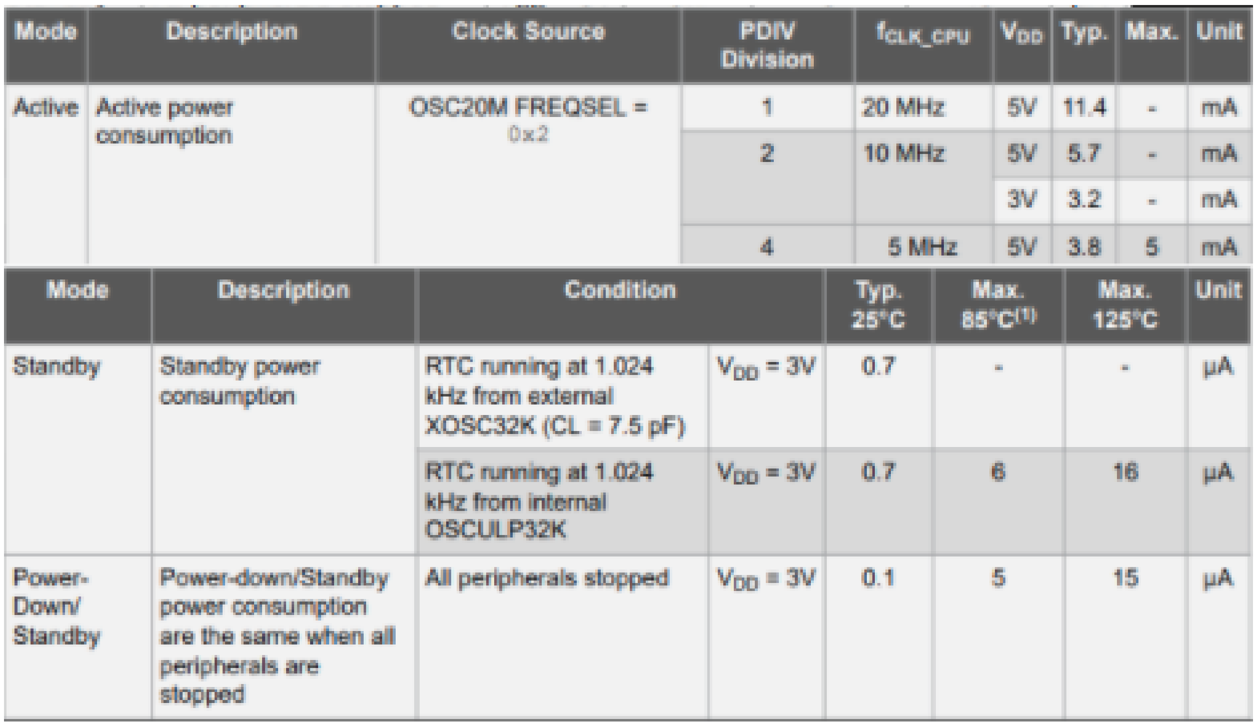
Figure 3: The state diagram of our code

The detection of HX711 is done by using the "is_ready()" function from the library which just checks if there is valid communication. Then the system starts polling to read values from the chip, if they are above a certain threshold (2Kg in our case) then the value is displayed on the console. If the reading is not above that threshold then the system goes to sleep by turning the CPU to its lowest sleep state and the HX711 to the sleep state. This allows for a lot of power saving as most of the time the scale is not used.

The sleep mode used is the "Power Down Mode" which is the lowest power consumption state of the MCU. The sleep modes and their power consummations are shown in figure 4

By putting the MCU to sleep we save 10^3W of power as the current consumption goes from mA to uA.

The only way to wake up from this sleep mode is using an external pin interrupt or an Real Time Clock Interrupt. We would have liked to used an external interrupt that triggered only when there is a mass above 2Kg on the scale, however, the digital communication did not allow us to do



Mode	Description	Clock Source	PDIV Division	f_{CLK_CPU}	V_{DD}	Typ.	Max.	Unit
Active	Active power consumption	OSC20M FREQSEL = 0x2	1	20 MHz	5V	11.4	-	mA
			2	10 MHz	5V	5.7	-	mA
					3V	3.2	-	mA
			4	5 MHz	5V	3.8	5	mA

Mode	Description	Condition		Typ. 25°C	Max. 85°C[1]	Max. 125°C	Unit
Standby	Standby power consumption	RTC running at 1.024 kHz from external XOSC32K (CL = 7.5 pF)	V_{DD} = 3V	0.7	-	-	µA
		RTC running at 1.024 kHz from internal OSCULP32K	V_{DD} = 3V	0.7	6	16	µA
Power-Down/ Standby	Power-down/Standby power consumption are the same when all peripherals are stopped	All peripherals stopped	V_{DD} = 3V	0.1	5	15	µA

Figure 4: Image obtained from ATMEGA 4809 Datasheet

that. Instead we had to use the on board RTC to send a periodic interrupt to the MCU every 2 seconds to wake it up and check the mass.

3.3 Problems Encountered

There were not many issues faced during the initial build of the system as everything was documented well. It just took a fair bit of time to come to an understanding of how load cells work and choose what hardware to use. The problems occurred after we had to move away from campus due to COVID-19 and did not have access to the Arduino we were using as it belonged to the lab. We had to switch to a new Arduino board that did not support the same sleep modes available. This process was tedious as a lot of testing had to be done to figure out how to use the RTC to wake up the MCU. In the end we did figure it out. Further, working on the project when the team had to communicate online was also difficult as only one of us could see the errors firsthand.

Another problem was that the scale had to be recalibrated each time the scale was moved. This was due to the fact that the slightest movement of the sensors placement affected the readings from the HX711 vastly. Since we did not have a proper mounted fit, the load cells shifted quite easily. This issue was largely mitigated by taping the sensors to the surface they rested on, and by not moving the scale without recalibrating it afterwords.

4 Current Limitations and Potential Improvements

The biggest limitation of this project is the calibration issue. Currently the load cells rest on mounts that were made for sensors of a different shape, and thus they shift very easily. We expect that by 3D printing a custom mount for the load cells the need for recalibration would be almost entirely removed, possibly even to the point of making it unnecessary to tare the scale each time it turns on.

The system could be improved by implementing our long-term goal of storing readings in a database, and using the data for tracking weight over time or finding correlations with other health data.

Another notable limitation of our current design is that readings are only visible on the console. Modifying the hardware design to add an LCD screen would allow the system to function without the need for the Arduino to be connected to an external computer. In a similar vein, it would useful to compute a single measurement instead of just reporting a stream of data from the sensor. In particular, an algorithm could be made to monitor the stream of data, detect when it is relatively stable, and only display the average of the stable measurements.

The power efficiency of the current system is limited by poor foresight in the hardware design. It might be possible to design the hardware such that the resistance of the load cells can be amplified and measured as an analogue voltage. If this were possible, then the Arduino could be configured so that instead of waking up every few seconds to check if there is a weight on the scale, the presence of a weight triggers an interrupt that wakes the Arduino from sleep.

A Proposal

This is the original proposal that we submitted:

> For our project we will be making the software for controlling a digital bathroom scale.
>
> We will be making hardware modifications to an ordinary digital bathroom scale, removing some of the internal electronics and replacing them with a soldered connection to an external processor, such as a Raspberry Pi or Arduino. Depending on what we expect to be easiest with the particular scale we end up using, the new controller will either be reading raw data from the scale's sensors or simply be reading the values that the original electronics send to the display.
>
> Since the scale will only be used for a few seconds each day, we will design our controller such that the main processor is able to be turned completely off while the scale is not in use. This will greatly reduce power usage, and possibly even make it feasible to run the device off of battery power.

B Source Code

The code for our project is available on GitHub and on GitLab. For convenience, we have reproduced the code below.

```
#include "HX711.h"
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#include <avr/power.h>
#include <avr/io.h>
#include <avr/cpufunc.h>

const int LOADCELL_DOUT_PIN = 4;
const int LOADCELL_SCK_PIN = 5;

HX711 scale;

void RTC_init(void)
{
    cli();
    uint8_t temp;

    RTC.CLKSEL |= B00000001;
    RTC.PITCTRLA |= B01001001;

    /* Initialize RTC: */
    while (RTC.STATUS > 0)
    {
        Serial.println("Waiting for registers to synchronize");
    }
    RTC.PITINTCTRL = 1; /* Periodic Interrupt: enabled */

    sei();
}


// interrupt raised by the RTC PIT firing
// when the watchdog fires during sleep, this function will be executed
// remember that interrupts are disabled in ISR functions
ISR(RTC_PIT_vect)
{
    /* Clear flag by writing '1': */
    RTC.PITINTFLAGS = 1;
}

void SLPCTRL_init(void)
{
    SLPCTRL.CTRLA |= SLPCTRL_SMODE_PDOWN_gc;
    SLPCTRL.CTRLA |= SLPCTRL_SEN_bm;
}

// Put the Arduino to deep sleep. Only an interrupt can wake it up.
void sleep(int ncycles)
{
  scale.power_down();
  int nbr_remaining = ncycles; // defines how many cycles should sleep

  while (nbr_remaining > 0){
    delay(10);
    sleep_cpu();

    // CPU is now asleep and program execution completely halts!

    nbr_remaining = nbr_remaining - 1;
  }

  scale.power_up();
}


void setup()
{
  Serial.begin(57600);
  Serial.println("Hello, World!");

  RTC_init();
  SLPCTRL_init();
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  scale.set_scale(-22800);
  scale.tare();
}

volatile int sleepCount = 4;

void loop()
{
  delay(100);
  if (scale.is_ready()) {
    float x = scale.get_units(10);
    Serial.println(x);
    if (x > 2) {
      sleepCount = 0;
      return;
    }
    sleepCount++;
    sleepCount%=5;
    if (sleepCount == 0) {
      Serial.println("Going to sleep");
      while(scale.get_units(10) < 2) {
          sleep(2);
      }
    }
  } else {
    Serial.println("HX711 not found.");
  }
}
```

C List of Purchased Hardware and Useful Third-Party Software

- For the controller, we used ultimately ended up using an Uno WiFi r2. Previous versions of our code worked with the Uno r3, which is typically cheaper.
- HX711 chip and load cells
- We used the standard Arduino IDE
- As a dependency, we used ‘‘HX711’’. It can be installed from the Arduino IDE; Menu Bar -> Tools -> Manage Libraries. Search for and install “HX711 Arduino Library”