

EAST WEST UNIVERSITY



UNDERGRADUATE PROJECT REPORT

ON

Microcontroller Based Continuous Weight Measurement & Automatic Data Log
System for a Vehicle

Submitted by : Sayeda Suraya Akter ID : 2011-3-60-006

Anannya Ekram ID : 2012-1-60-023

Supervised By : Dr. Anisur Rahman,

Assistant Professor, Dept. of Computer Science and Engineering,
East West University

A project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science and Engineering to the Department of Computer Science and Engineering.

Dec 1, 2016

DECLARATIONS

This is to certify that this project is an original work and was done by us and it has not been submitted elsewhere for the requirement of any other purposes.

Signature of the students

.....

Sayeda Suraya Akter

.....

Anannya Ekram

ACCEPTANCE

This project is entitled “Microcontroller Based Continuous weight Measurement & Automatic Data Log System for a Vehicle” submitted by Sayeda Suraya Akter, ID No. 2011-3-60-006 & Anannya Ekram ID No. 2012-1-60-023 to the department of Computer Science & Engineering, East West University, Dhaka, Bangladesh is accepted as satisfactory for partial fulfillment of the requirement for the degree of Bachelor of Computer Science & Engineering on December 2016.

.....

Dr. Anisur Rahman

Assistant Professor

Department of Computer Science & Engineering

East West University, Dhaka, Bangladesh .

.....

Dr. Mozammel Huq Azad Khan

Chairperson and Professor

Department of Computer Science & Engineering

East West University, Dhaka, Bangladesh .

Abstract

Due to the expansion of the vehicle transportation system, misuse of vehicle is also increased & become a serious problem in recent years. Because lack of keeping track of the vehicles, it is essential to design & implement a modern system, which can monitor & control these vehicles. A system that is able to calculate the vehicle's weight continuously & automatically store the data in the microSD card memory as a file. So that owner can check the data any time & able to keep an eye on his/her driver. In our project, we have showed it by interfacing both the Load Cell & Arduino Uno microcontroller.

Acknowledgement :

Anisur Rahman, Assistant professor of the department of Computer Science and Engineering, East West University for his precious contribution, consistent advise and support for the completion of this project. Without his support it was not possible for us to complete this task.

We would like to express our gratitude to all the faculty members of the Computer Science and Engineering Department for the support they had given.

But above everything we like to thank almighty of Allah for giving s the ability, patience & stamina to complete this task successfully.

Contents

Declaration	i
Acceptance	ii
Abstract	iii
Acknowledgement	iv

Chapter 1 Introduction

1.1 Motivation	01
1.2 Objective	01
1.3 Overview of the Proposed System	02
1.4 Contribution	02
1.5 Outline of the thesis	02

Chapter 2 Description of Hardware System

2.1 Basic Introduction of Devices	03
2.1.1 Arduino Uno	03
2.1.2 Load Cell	09
2.1.3 Amplifier Module 24-bit	14
2.1.4 16×2 LCD Display	16
2.1.5 MicroSD Card Adapter	18
2.1.6 Read Time Clock (DS13072)	20

Chapter 3 Description of the Software System

3.1 Introduction	26
3.2 Structure	26
3.3 Variable Declaration	27
3.4 Arithmetic Operators	27

3.5 Comparison Operators	28
3.6 Logical Operator	28
3.7 Function	28
3.7.1 pinMode(pin, mode)	29
3.7.2 digitalRead(pin)	29
3.7.3 digitalWrite(pin, value)	30
3.7.4 delay(ms)	30
3.7.5 Serial.begin(rate)	31
3.7.6 Serial.println(data)	31
3.8 Code Generation	32
Chapter 4 Conclusion	
4.1 Contribution of the propose system	40
4.2 Limitations of the system	40
References	41

Chapter 1

Introduction

Since the creation of the first human on Earth, business and transportation of goods through land and on the road has been a usual way. With developing the highway, transportation and business trade, vehicle weigh-in-motion technology has become a key technology and trend of measuring weight of the loads. Moreover, because of the strong competition between transport modes and companies, transportation management was improved, which has led to an increase in the numbers of fully loaded trucks and their gross weights. Recently, there have been a significant number of open access to vehicles illegally overloaded and the damage vehicles cause on the road is in direct proportion to the axle weight by 4th power. The overloaded transportation would greatly increase the cost for the pavement maintenance and repair, shorten the service life of pavement, even affect the traffic safety and capability. So it is imperative to build a weigh station to solve these problems. We are mainly working for the private vehicles. A weighting scale is a device for measuring weight. Many weight measurement machine have been made for different purpose. Our Machine measures the weight & also automatically save data of the weight as well as keeping track of current time & date.

This chapter introduces the proposed weighing measurement system. The motivation behind this system is described in section 1.1 Objective of the proposed system discussed in section 1.2 while section 1.3 gives a short overview of the proposed system. Section 1.4 presents the contribution of proposed system. Finally section 1.5 is about the outline for the rest of the document.

1.1 Motivation

There is a need for an automatic vehicle load monitoring system and navigation monitoring system that can measure the weight of the vehicle at every moment. For this purpose, this system consists of two main parts: hardware and software. Hardware part is in control of measuring changes of suspension system and processing these data in order to gain vehicle loading weight. Concluded weight could be displayed to the driver on the LCD in vehicle cabin. The system comprises a weight sending device attached to a base of a vehicle, and wherein the weight sensing, device is load cell, a compression spring attached to the weight sensing device and to a suspension spring of the vehicle, a voltage conversion unit attached to the weight sensing device to convert and output resistance of the load cell into a voltage.

The amount of transferred load by vehicles and calculating the value of load has been of concerns of the managers of transportation system. There has always been a need for load sensor with a reasonable price that can calculate the amount of load on the vehicle and provide it for the employers.

1.2 Objective

Our project purpose is focus on private vehicles' (car) security assurance. If the User set up machine, according to our project mechanism, Owner can keep tracking on his/her car, during driver driving it , by checking microSD card which logged the data in file. This will ensure weather the driver taking any extra passenger or stuff from the road, which is heavy than its capacity. So any time owner of the vehicle simply can check the microsd card via connecting with his/her computer or mobile phone.

1.3 Overview of the proposed system

This document presents, microcontroller board design. Microcontrollers use inputs and outputs like any computer. Inputs capture information from the user or the environment while outputs do something with the information that has been captured. A switch and a sensor could be a digital and an analog input respectively into the Arduino. Any object we want to turn on and off and control could be an output. Strain gauge load cells are the most common in industry. These load cells are particularly stiff, have very good resonance values, and tend to have long life cycles in application. Strain gauge load cells work on the principle that the strain gauge (a planar resistor) deforms / stretches/contracts when the material of the load cells deform appropriately.

1.4 Contribution

In this work, we present arduino Uno microcontroller interfacing with computer by getting instruction given in the arduino IDE, which is written in the programming language. In this work we showed , how load sensor taking input from the objects (weight) & based on that how the amplifier module -24bit interfacing directly with a bridge sensor.

1.5 Outline of this project

Chapter 2 describes the entire hardware system of this project & necessary steps of interfacing.

Chapter 3 describes the entire software system of this project & necessary functions.

Chapter 2

Description of the Hardware System

In this chapter we describe the entire proposed system of our project where all the necessary components name, their significance in the project are described elaborately.

2.1 Basic Introduction of the Devices

2.1.1 Arduino Uno

The **Arduino Uno** is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button.

The first Arduino was introduced in 2005, based on 8-bit Atmel AVR, aiming to provide a low cost, easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots , thermo-stats and motion detectors.

An Arduino board consists of an Atmel 8-, 16- or 32-bit AVR microcontroller (although since 2015 other makers' microcontrollers have been used) with complementary components that facilitate programming and incorporation into other circuits. An important aspect of the Arduino is its standard connectors, which let users connect the CPU board to a variety of interchangeable add-on modules termed *shields*. Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via an I²C serial bus—so many shields can be stacked and used in parallel. Before 2015, Official Arduinos had used the Atmel megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. In 2015, units by other producers were added. A handful of other processors have also been used by Arduino compatible devices. Most boards include a 5 V linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the Lily Pad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external chip programmer. This makes using an Arduino more straightforward by allowing the use of an ordinary computer as the programmer. Currently, opt boot boot loader is the default boot loader installed on Arduino UNO.

At a conceptual level, when using the Arduino integrated development environment, all boards are programmed over a serial connection. Its implementation varies with the hardware version. Some serial Arduino boards contain a level shifter circuit to convert between RS-232 logic levels and transistor–transistor logic (TTL) level signals. Current Arduino boards are programmed via Universal Serial Bus (USB), implemented using USB-to-serial adapter chips such as the FTDI FT232. Some boards, such as later-model Uno boards, substitute the FTDI chip with a separate AVR chip containing USB-to-serial firmware, which is reprogrammable via its own ICSP header. Other variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods, when used with traditional microcontroller tools instead of the Arduino IDE, standard AVR in-system programming (ISP) programming is used.

Arduino and Arduino-compatible boards use printed circuit expansion boards called *shields*, which plug into the normally supplied Arduino pin headers. Shields can provide motor controls for 3D printing and other applications, Global Positioning System (GPS), Ethernet, liquid crystal display (LCD), or breadboarding (prototyping). Several shields can also be made do it yourself (DIY).

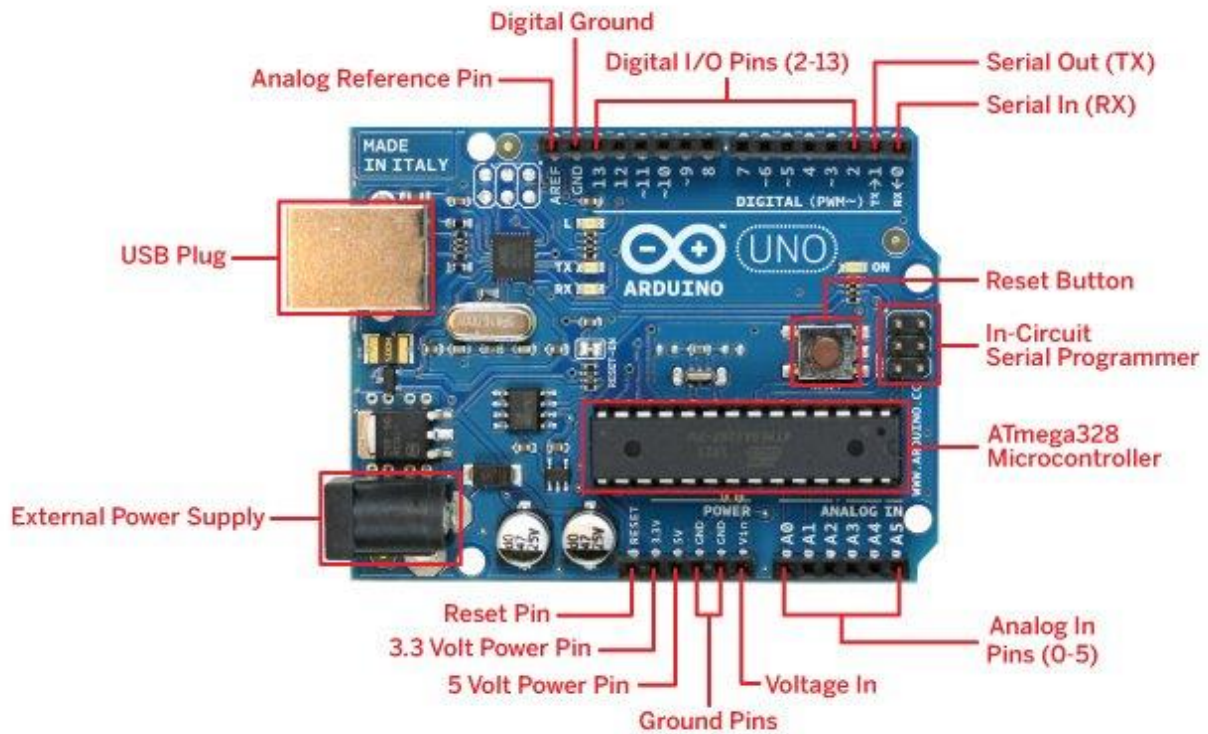


Figure 2.1 : Arduino Uno

Technical Specification

1) Microcontroller	ATmega328
2) Operating	Voltage 5V
3) Input Voltage (recommended)	7-12V
4) Input Voltage (limits)	6-20V
5) Digital I/O Pins	14 (of which 6 provide PWM output)
6) Analog Input Pins	6
7) DC Current per I/O Pin	40 mA
8) DC Current for 3.3V Pin	50 mA
9) Flash Memory	32 KB of which 0.5 KB used by bootloader

10)SRAM	2 KB
11)EEPROM	1 KB
12)Clock Speed	16 MHz

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter

can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a

battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V

pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage

regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the boot loader; It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the [EEPROM Library](#)).

Input & Output

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode\(\)](#), [digitalWrite\(\)](#), and [digitalRead\(\)](#) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the [attachInterrupt\(\)](#) function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite\(\)](#) function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off. The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the [analogReference\(\)](#) function. Additionally, some pins have specialized functionality:
- **I2C: 4 (SDA) and 5 (SCL).** Support I2C (TWI) communication using the [Wire library](#). There are a couple of other pins on the board:
- **AREF.** Reference voltage for the analog inputs. Used with [analogReference\(\)](#).
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Arduino LEDs

Likewise, the Arduino has **four** LEDs: **L**, **RX**, **TX**, and **ON** .On the UNO, three are in the middle and one is to the right .

ON LED - this LED will shine green whenever the Arduino is powered. Always check this LED if your Arduino is not acting right, if its flickering or off then you should check your power supply.

RX and **TX** LEDs - these are like the 'send' and 'receive' LEDs on your cable modem. They blink whenever information is sent from or to the Arduino through the USB connection. The **TX** LED lights up yellow whenever data is sent **from the Arduino to the computer** USB port . The **RX** LED lights up yellow whenever data is sent **to the Arduino from the computer** USB port.

L LED - this is the one LED that you can control. The ON, RX and TX LEDs all light up automatically no matter what. The **L** LED, however, is connected to the Arduino main chip and you can turn it on or off when you start writing code.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, an *.inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A [SoftwareSerial library](#) allows for serial communication on any of the Uno's digital pins. The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the [documentation](#) for details. To use the SPI communication, please see the ATmega328 datasheet.

The Arduino IDE (integrated development environment) supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consist of two functions that are compiled.

2.1.2 Load Cell

A **load cell** is a transducer that is used to create an electrical signal whose magnitude is directly proportional to the force being measured. The various types of load cells include hydraulic load cells, pneumatic load cells and strain gauge load cells. In our Project we use straight gauge load cell.

Through a mechanical construction, the force being sensed deforms a strain gauge. The strain gauge measures the deformation (strain) as a change in electrical resistance, which is a measure of the strain and hence the applied forces. A load cell usually consists of four strain gauges in a Wheatstone bridge configuration. Load cells of one strain gauge (quarter bridge) or two strain gauges (half bridge) are also available.^[1] The electrical signal output is typically in the order of a few millivolts and requires amplification by an instrumentation amplifier before it can be used. The output of the transducer can be scaled to calculate the force applied to the transducer. Sometimes a high resolution ADC, typically 24-bit, can be used directly.

Strain gauge load cells are the most common in industry. These load cells are particularly stiff, have very good resonance values, and tend to have long life cycles in application. Strain gauge load cells work on the principle that the strain gauge (a planar resistor) deforms/stretches/contracts when the material of the load cells deforms appropriately. These values are extremely small and are relational to the stress and/or strain that the material load cell is undergoing at the time. The change in resistance of the strain gauge provides an electrical value change that is calibrated to the load placed on the load cell.

Strain gauge load cells convert the load acting on them into electrical signals. The gauges themselves are bonded onto a beam or structural member that deforms when weight is applied. In most cases, four strain gauges are used to obtain maximum sensitivity and temperature compensation. Two of the gauges are usually in tension can be represented as T1 and T2, and two in compression can be represented as C1 and C2, and are wired with compensation adjustments. The strain gauge load cell is fundamentally a spring optimized for strain measurement. Gauges are mounted in areas that exhibit strain in compression or tension. When weight is applied to the load cell, gauges C1 and C2 compress decreasing their resistances. Simultaneously, gauges T1 and T2 are stretched increasing their resistances. The change in resistances causes more current to flow through C1 and C2 and less current to flow through T1 and T2. Thus a potential difference is felt between the output or signal leads of the load cell. The gauges are mounted in a differential bridge to enhance measurement accuracy.^[2] When weight is applied, the strain changes the electrical resistance of the gauges in proportion to the load.^[3] Other load cells are fading into obscurity, as strain gauge load cells continue to increase their accuracy and lower their unit costs.



Figure 2.1 : Load Cell

A strain gauge is a device that measures electrical resistance changes in response to, and proportional of, strain (or pressure or force or whatever you so desire to call it) applied to the device. The most common strain gauge is made up of very fine wire, or foil, set up in a grid pattern in such a way that there is a linear change in electrical resistance when strain is applied in one specific direction, most commonly found with a base resistance of 120Ω , 350Ω , and $1,000\Omega$.

Wiring

The full-bridge cells come typically in four-wire configuration. The wires to the top and bottom end of the bridge are the excitation (often labelled $E+$ and $E-$, or $Ex+$ and $Ex-$), the wires to its sides are the signal (labelled $S+$ and $S-$). Ideally, the voltage difference between $S+$ and $S-$ is zero under zero load, and grows proportionally to the load cell's mechanical load.

Sometimes a six-wire configuration is used. The two additional wires are "sense" (Sen+ and Sen-), and are connected to the bridge with the Ex+ and Ex- wires, in a fashion similar to [four-terminal sensing](#). With these additional signals, the controller can compensate for the change in wire resistance due to e.g. temperature fluctuations.

The individual resistors on the bridge usually have resistance of 350 Ω . Sometimes other values (typically 120 Ω , 1,000 Ω) can be encountered.

The bridge is typically electrically insulated from the substrate. The sensing elements are in close proximity and in good mutual thermal contact, to avoid differential signals caused by temperature differences.

Wiring Colors

The most common color assignment is red for Ex+, black for Ex-, green for S+, and white for S-.

Less common assignments are red for Ex+, white for Ex-, green for S+, and blue for S-, or red for Ex+, blue for Ex-, green for S+, and yellow for S-.^[4] Other values are also possible, e.g. red for Ex+, green for Ex-, yellow for S+ and blue for S-.

Straight Gauge Amplifier

Each strain gauge has a different sensitivity to strain, which is expressed quantitatively as the gauge factor (GF). The gauge factor is defined as the ratio of fractional change in electrical resistance to the fractional change in length (strain).

(The gauge factor for metallic strain gauges is typically around 2.

We set up a strain gauge load cell and measure that change in resistance and all is good, right?

Not so fast. Strain measurements rarely involve quantities larger than a few mill strain (fancy units for strain, but still very small). So let's take an example: suppose you put a strain of 500me. A strain gauge with a gauge factor of 2 will have a change in electrical resistance of only for a 120 Ω gauge, this is a change of only 0.12 Ω .

0.12 Ω is a very small change, and, for most devices, couldn't actually be detected, let alone detected accurately. So we are going to need another device that can either accurately measure super small changes in resistance (spoiler: they are very expensive) or a device that can take that very small change in resistance and turn it into something that we can measure accurately.

This is where an amplifier, such as the HX711 comes in handy.

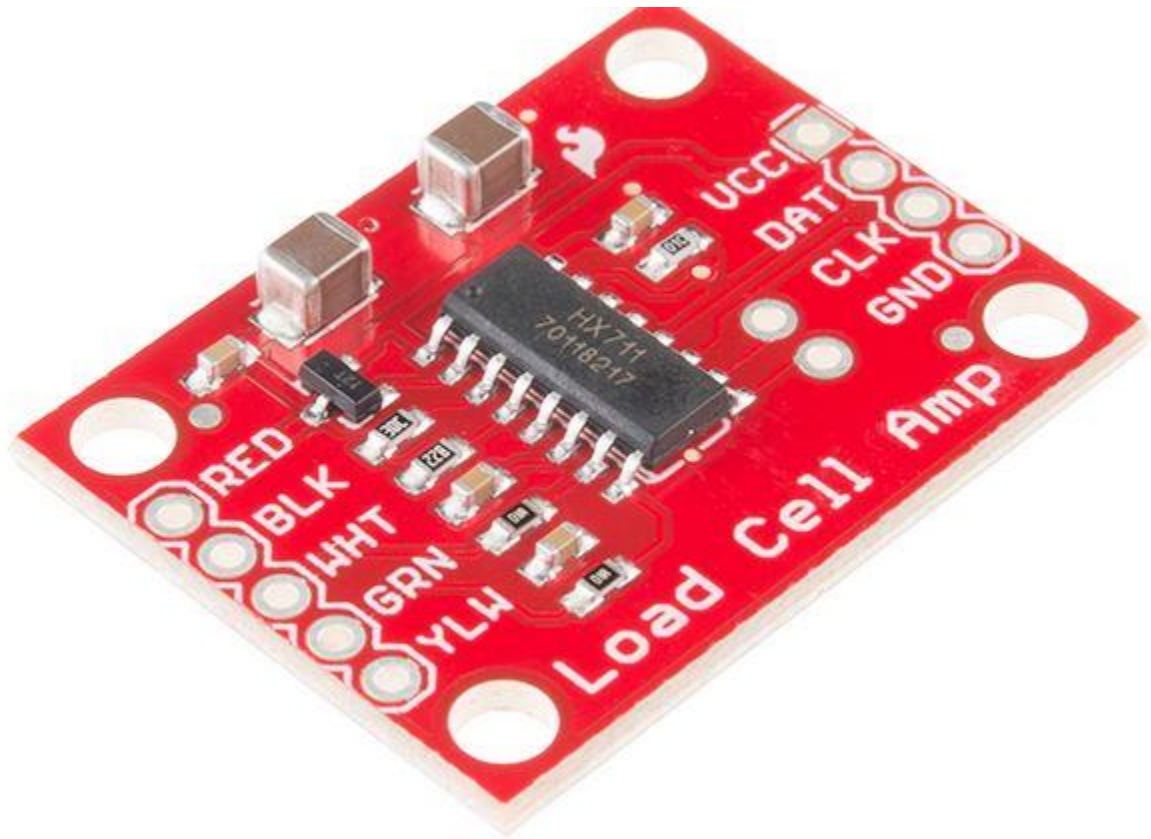


Figure : HX711 Amplifier Breakout Board

A good way of taking small changes in resistance and turning it into something more measurable is using a Wheatstone bridge. A Wheatstone bridge is a configuration of four resistors with a known voltage applied like this:

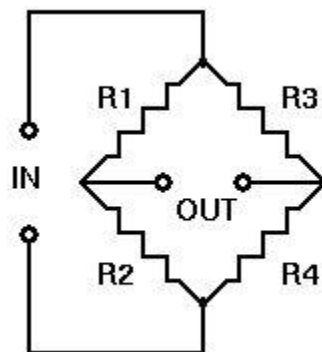


Figure 2.1 : Schematic Diagram of Strain Gauge Load Cell

where V_{in} is a known constant voltage, and the resulting V_{out} is measured. If V_{in} then V_{out} is 0, but if there is a change to the value of one of the resistors, V_{out} will have a resulting change that can be measured and is governed by the following equation using ohms law:

$$V_{out} = [(R3/(R3 + R4) - R2/(R1 + R2))] * V_{in}$$

By replacing one of the resistors in a wheatstone bridge with a strain gauge, we can easily measure the change in V_{out} and use that to assess the force applied.

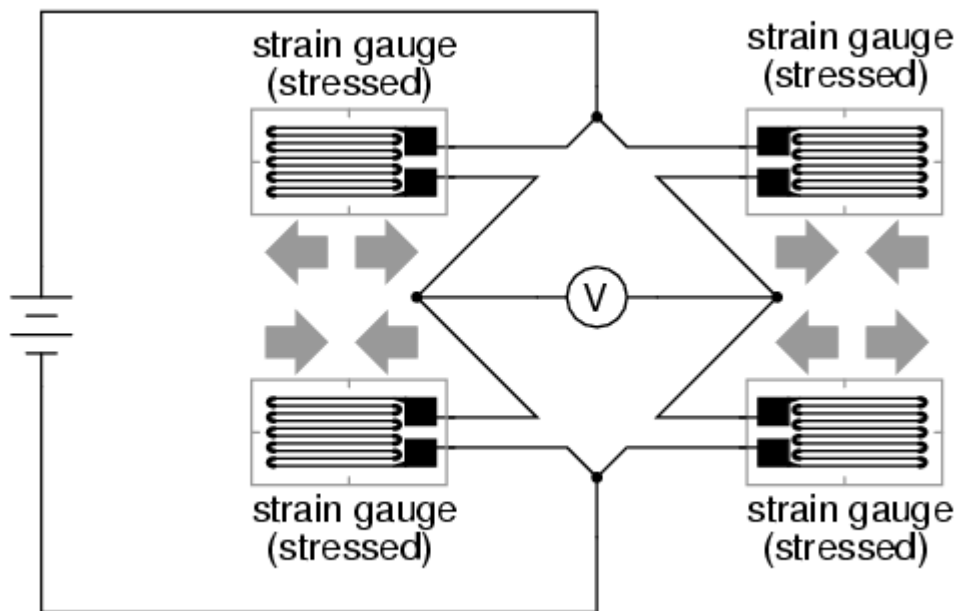


Figure 2.1 : Full Bridge Strain Gauge Circuit

Excitation and Rated Output

The bridge is excited with stabilized voltage (usually 10V, but can be 20V, 5V, or less for battery powered instrumentation). The difference voltage proportional to the load then appears on the signal outputs.

The cell output is rated in millivolts per volt (mV/V) of the difference voltage at full rated mechanical load. So a 2.96 mV/V load cell will provide 29.6 millivolt signal at full load when excited with 10 volts.

Typical sensitivity values are 1 to 3 mV/V. Typical maximum excitation voltage is around 15 volts.

2.1.3 Amplifier Module -24 bit

This Weight Sensor amplifier is based on HX711, which consists of an amplifier and a precision 24-bit analog-to-digital converter designed for weigh scale and industrial control applications to interface directly with a bridge sensor. Compared with other chips, HX711 not only has a few basic functions, also contains high integration, fast response, immunity, and other features. The chip lowered the cost of the electronic scale, at the same time, improving the performance and reliability. The input interface of this weight sensor module is used sensor interface, which is compatible with Arduino I/O ports. The output adopts compact terminal that makes weight sensor module easier to connect the weight sensor. It's the best choice for electronic enthusiasts to do some tiny home scale.

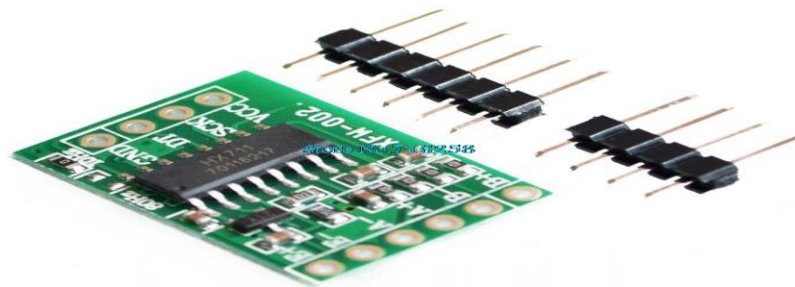


Figure 2.1: Amplifier Module -24bit

The Load Cell Amplifier and ADC Module is a small breakout board for the HX711 IC that allows you to easily read load cells to measure weight. By connecting the module to your microcontroller you will be able to read the changes in the resistance of the load cell and with some calibration you'll be able to get very accurate weight measurements. This can be handy for creating your own industrial scale, process control, or simple presence detection.

The HX711 uses a two wire interface (Clock and Data) for communication. Any microcontroller's GPIO pins should work and numerous libraries have been written making it easy to read data from the HX711. Check the hookup guide below for more information.

Load cells use a four wire wheatstone bridge to connect to the HX711. These are commonly colored RED, BLK, WHT, GRN, and YLW.

Each color corresponds to the conventional color coding of load cells:

- Red (Excitation+ or VCC)
- Black (Excitation- or GND)
- White (Amplifier+, Signal+, or Output+)
- Green (A-, S-, or O-)
- Yellow (Shield)

The YLW pin acts as an optional input that is not hooked up to the strain gauge but is utilized to ground and shield against outside EMI (electromagnetic interference). Please keep in mind that some load cells might have variations in color coding.

Features

- Two selectable differential input channels
- On-chip active low noise PGA with selectable gain of 32, 64 and 128
- On-chip power supply regulator for load-cell and ADC analog power supply
- On-chip oscillator requiring no external component with optional external crystal
- On-chip power-on-reset
- Simple digital control and serial interface: pin-driven controls, no programming needed
- Measurement Resolution: 24 bit
- Selectable 10SPS or 80SPS output data rate
- Simultaneous 50 and 60Hz supply rejection
- Current consumption including on-chip analog power supply regulator:
 - normal operation < 1.5mA,
 - power down < 1uA
- Operation supply voltage range: 2.7V ~ 5.5V
- Operation temperature range: -40 ~ +85°C

Pinouts

Analog Side

- E+ : Excitation positive
- E- : Excitation negative
- A- : Channel A Negative Input
- A+ : Channel A positive Input
- B- : Channel B Negative Input
- B+ : Channel B positive Input

Digital Side

- GND: 0V / Ground Power Connection
- DT: Data IO Connection
- SCK: Serial Clock Input
- VCC: Power Input

Package Includes

- 1x Load Cell Amplifier and 24-Bit ADC Module based on HX711 chip
- 10 pin (6+4) Male to Male Header pins in 0.1 inch pitch

2.1.4 16x2 LCD Display

The Liquid Crystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface.

This example sketch prints "Hello World!" to the LCD and shows the time in seconds since the Arduino was reset :



Figure 2.1 : 16x2 LCD Display

Pin No	Symbol	Level	Description
1	VSS	0V	Ground
2	VDD	5V	Supply Voltage for logic
3	VO	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA, L: Instruction code
5	R/W	H/L	H: Read(MPU?Module) L: Write(MPU?Module)
6	E	H,H->L	Chip enable signal
7	DB0	H/L	Data bus line
8	DB1	H/L	Data bus line
9	DB2	H/L	Data bus line
10	DB3	H/L	Data bus line
11	DB4	H/L	Data bus line
12	DB5	H/L	Data bus line
13	DB6	H/L	Data bus line
14	DB7	H/L	Data bus line
15	A	5V	LED +
16	K	0V	LED-

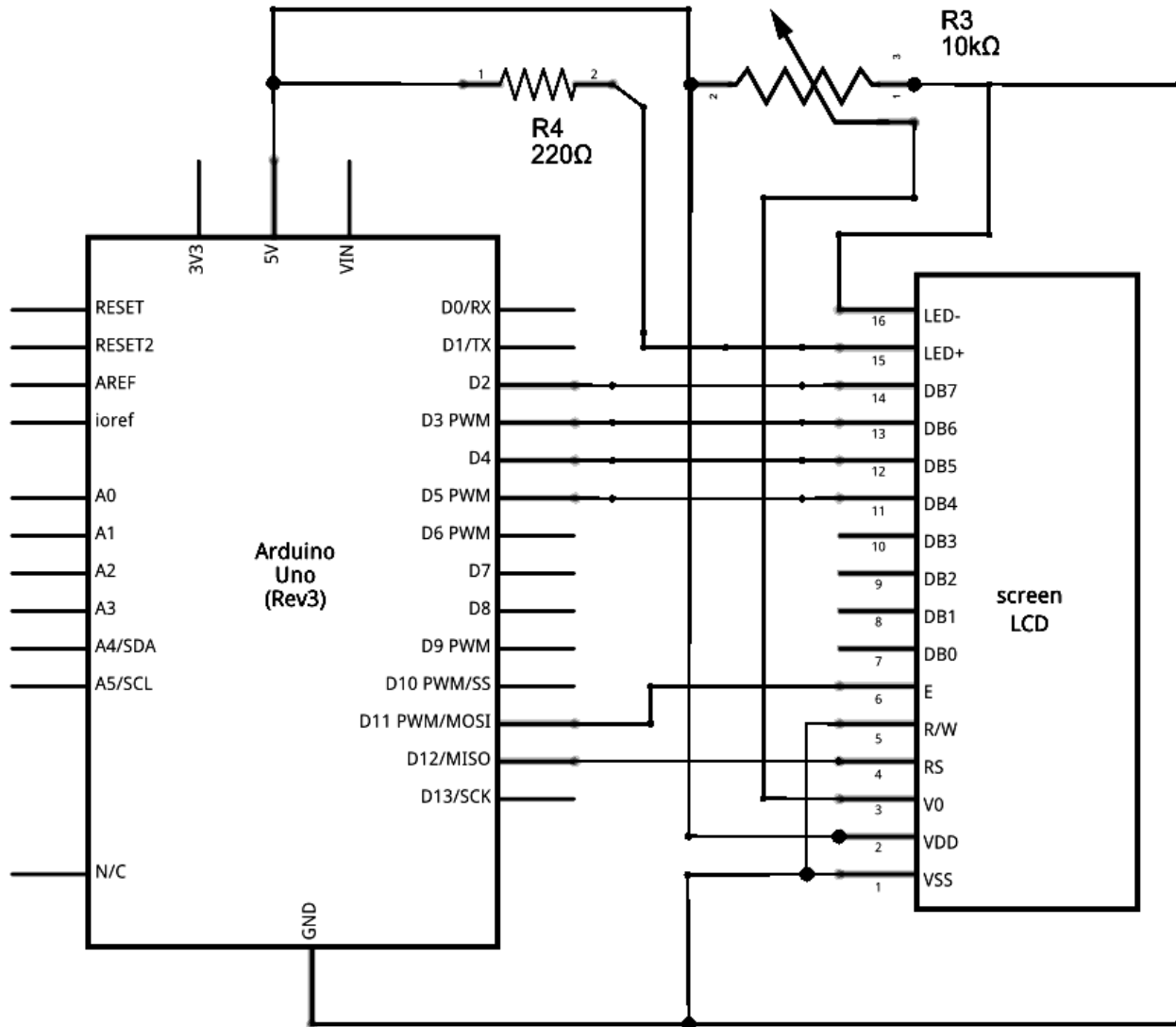


Figure :Schamatic diafram of Arduino Uno & LCD Display

2.1.5 SPI MicroSD Card Adapter

The module is a Micro SD card read and write module; SCM system can read and write file on micro SD card through SPI interface; Can initialize, read and write card by using SD card library of Arduino IDE; A product for Arduino that works with official Arduino boards.

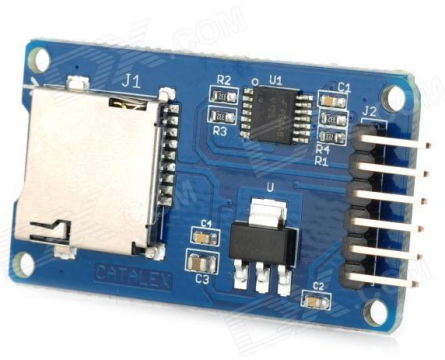


Figure 2.1 :SPI MicroSD Card

General	
Brand	N/A
Model	N/A
Quantity	1Piece
Color	Blue+Silvery
Material	PCB
Specification	
Chipset	N/A
English Manual / Spec	Yes
Download Link	http://pan.baidu.com/s/1hqj1PxM
Other Features	
Other	Power supply: VCC 4.5~5.5V; Current: 0.2~200mA; Interface electrical level: 3.3V

Features	/ 5V; Onboard 3.3V voltage regulator circuit; Supports Micro SD up to 2GB; Micro SDHC up to 32GB; Leads: GND, VCC, MISO, MOSI, SCK, CS; SPI standard interface; M2 2.2mm screw installation holes
Dimensions & Weight	
Dimensions	1.77 in x 0.94 in x 0.28 in (4.5 cm x 2.4 cm x 0.7 cm)
Weight	0.18 oz (5 g)

Packing List

1 x Module

All packages from DX.com are sent without DX logo or any information indicating DX.com. Due to package variations from suppliers, the product packaging customers receive may be different from the images displayed.

To enable volume discounts on this site, use coupon code: **BULKRATE** during checkout. You will see a discount applied at the bottom of the shopping cart. Competitive pricing is available. Contact us for details.

Bulk Rate is a semi-wholesale system with items priced separately from retail. When you use bulk rates, a flat \$1.70 registered air mail fee will automatically be added to your cart to ensure delivery of package. While BulkRate's intention is to offer cheaper prices when you buy in bulk, because it is priced separately it on occasions show a higher than retail price. That's why we ask you to enter BULKRATE as a coupon code to manually activate the rates.

2.1.6 Real Time Clock

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

1) Completely Manages All Timekeeping Functions

- i) Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year with Leap-Year Compensation Valid Up to 2100
 - ii) 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
 - iii) Programmable Square-Wave Output Signal
- 2) Simple Serial Port Interfaces to Most Microcontrollers
- i) I2C Serial Interface
- 3) Low Power Operation Extends Battery Backup Run Time
- i) Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
 - ii) Automatic Power-Fail Detect and Switch Circuitry
- 4) 8-Pin DIP and 8-Pin SO Minimizes Required Space
- 5) Optional Industrial Temperature Range: -40°C to +85°C Supports Operation in a Wide Range of Applications
- 6) Underwriters Laboratories® (UL) Recognized

Ordering Information

PART	TEMP RANGE	VOLTAGE (V)	PIN-PACKAGE	TOP MARK*
DS1307+	0°C to +70°C	5.0	8 PDIP (300 mils)	DS1307
DS1307N+	-40°C to +85°C	5.0	8 PDIP (300 mils)	DS1307N
DS1307Z+	0°C to +70°C	5.0	8 SO (150 mils)	DS1307
DS1307ZN+	-40°C to +85°C	5.0	8 SO (150 mils)	DS1307N
DS1307Z+T&R	0°C to +70°C	5.0	8 SO (150 mils) Tape and Reel	DS1307
DS1307ZN+T&R	-40°C to +85°C	5.0	8 SO (150 mils) Tape and Reel	DS1307N

+Denotes a lead-free/RoHS-compliant package.

*A “+” anywhere on the top mark indicates a lead-free package. An “N” anywhere on the top mark indicates an industrial temperature range device.

Underwriters Laboratories, Inc. is a registered certification mark of Underwriters Laboratories, Inc.

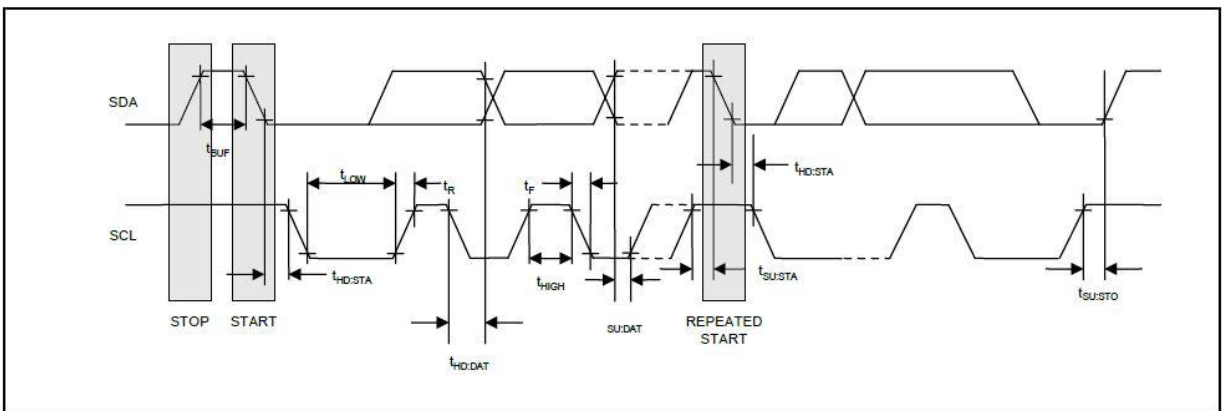
Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

Capacitance

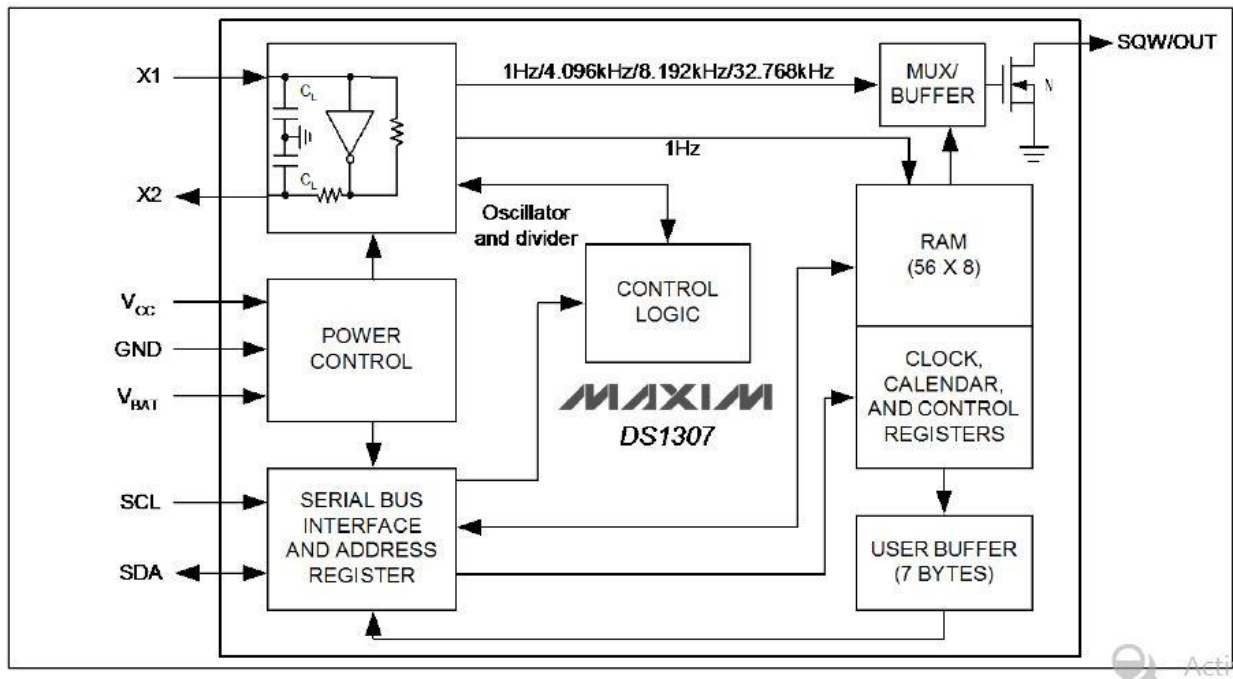
T_A = +25°C

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Pin Capacitance (SDA, SCL)	C _{I/O}				10	pF
Capacitance Load for Each Bus Line	C _B	(Note 7)			400	pF

Timing Diagram



Block Diagram



CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table 2 shows the RTC registers. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on.) Illogical time and date entries result in undefined operation. Bit 7 of Register 0 is the clock halt (CH) bit. When this bit is set to 1, the oscillator is disabled. When cleared to 0, the oscillator is enabled. On first application of power to the device the time and date registers are typically reset to 01/01/00 01 00:00:00 (MM/DD/YY DOW HH:MM:SS). The CH bit in the seconds register will be set to a 1. The clock can be halted whenever the timekeeping functions are not required, which minimizes current (IBATDR).

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20 to 23 hours). The hours value must be re-entered whenever the 12/24-hour mode bit is changed.

When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any I2C START. The time information is read from these secondary registers while the clock continues to run. This eliminates the need to re-read the registers in case the internal registers update during a read. The divider chain is reset whenever the seconds register is written. Write transfers occur on the I2C acknowledge from the DS1307. Once the divider chain is reset, to avoid rollover issues, the remaining time and date registers must be written within one second.

Time Keeper register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds		Seconds				Seconds	00–59	
01h	0	10 Minutes		Minutes				Minutes	00–59	
02h	0	12	10 Hour	10 Hour	Hours			Hours	1–12 +AM/PM 00–23	
		24	PM/ AM							
03h	0	0	0	0	DAY			Day	01–07	
04h	0	0	10 Date		Date			Date	01–31	
05h	0	0	0	10 Month	Month			Month	01–12	
06h	10 Year			Year				Year	00–99	
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh								RAM 56 x 8	00h–FFh	

Control Register

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

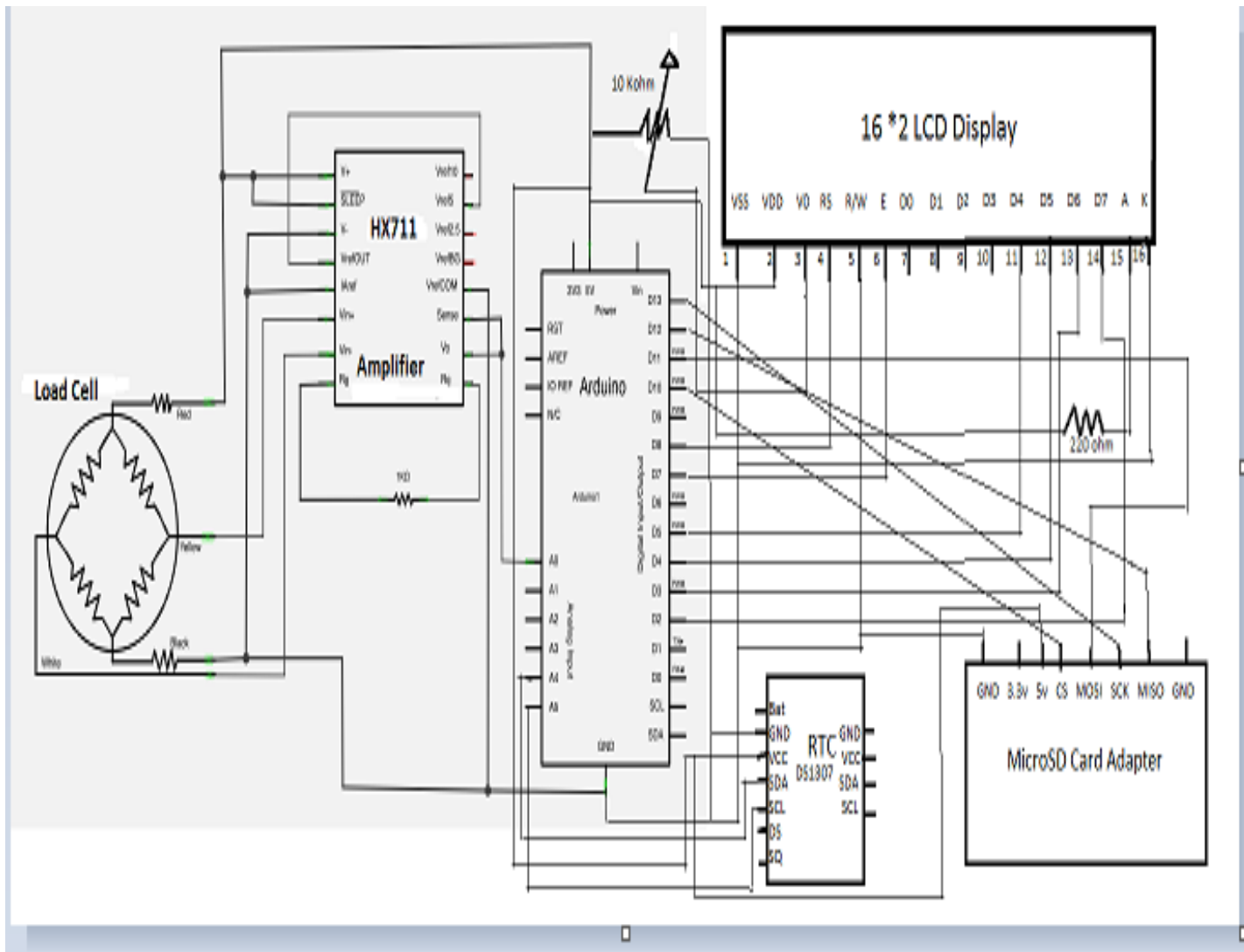
Bit 7: Output Control (OUT). This bit controls the output level of the SQW/OUT pin when the square-wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0. On initial application of power to the device, this bit is typically set to a 0.

Bit 4: Square-Wave Enable (SQWE). This bit, when set to logic 1, enables the oscillator output. The frequency of the square-wave output depends upon the value of the RS0 and RS1 bits. With the square-wave output set to 1Hz, the clock registers update on the falling edge of the square wave. On initial application of power to the device, this bit is typically set to a 0.

Bits 1 and 0: Rate Select (RS[1:0]). These bits control the frequency of the square-wave output when the square-wave output has been enabled. The following table lists the square-wave frequencies that can be selected with the RS bits. On initial application of power to the device, these bits are typically set to a 1.

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

Schematic Diagram of the weight measurement system is given below :



Chapter 3

Description of The Software System

3.1 Introduction

In this project we use programming reference for the command structure and the basic syntax of the arduino microcontroller. This project continues on to describe the syntax of the most common elements of the language and illustrates their usage with examples and code fragments. This includes many functions of the core library followed by an appendix with sample schematics and started program. Beginning with the basic structure of Arduino's C & C++ derived programming language.

The Arduino Uno can be programmed with the Arduino software ([download](#)). Select "Arduino Uno w/ATmega328" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the [reference](#) and [tutorials](#). The ATmega328 on the Arduino Uno comes preburned with a [bootloader](#) that allows you to upload new code to it without the use of an external hardware programmer.

It communicates using the original STK500 protocol ([reference](#), [C header files](#)). You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see [these instructions](#) for details.

The ATmega8U2 firmware source code is available . The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. we can then use [Atmel's FLIP software](#) (Windows) or the [DFU programmer](#) (MacOS X and Linux) to load a new firmware. Or you can use the ISP header with an external programme(overwriting the DFU bootloader).

3.2 Structure

The basic structure of the arduino language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
Void setup()
{
    statements ;
}

Void loop()
```

```
{  
    statements ;  
}
```

Where setup() is the preparation, loop() is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program & it runs only once , and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously - reading inputs, triggering outputs etc. This function is the core of all arduino programs and does the bulk of the work.

3.3 Variable Declaration

All variable have to be declared before they can be used. Declaring a variable means defining its value type as in int, long, float, setting a specified name and optionally assigned an initial value. This only needs to be done once in a program but the value can be changed at any time using the arithmetic and various assignments.

The following example declares that input variable is an int or integer type and that its initial value equals zero. This is called a simple assignment.

```
Int inputVariable = 0 ;
```

A variable can be declared in a number of locations throughout the program & where its definition takes place determines what parts of the program can use the variable.

3.4 Arithmetic Operator

Arithmetic operators include addition, subtraction, multiplication and division . They return the sum, difference, product and quotient of two operands.

```
y = y+3 ;
```

```
x = x-7 ;
```

```
i = j* 6 ;
```

```
r = r/5;
```

3.5 Compound Assignment

Compound assignment combine an arithmetic operation with a variable assignment. These are commonly found in for loops as described later . The most common compound assignments include :

```
X++ ; // increment x by 1
```

```
x-- ; // decrement x by 1
```

```
x+ = y ; //increment x by +y
```

```
x- = y; // decrement x by -y
```

```
x * = y; // multiplies x by y
```

```
x /= y; //divides x by y
```

3.6 Logical operator

Logical operations are way to compare two expressions and return a TRUE or FALSE depending on the operator. There are three logical operators AND, OR and NOT , that are often used in if statements .

Logical AND :

```
If (x> 0 && x<5) // true only if both expressions are true
```

Logical OR :

```
If ( x> 0 || y >0) // true if either expression is true
```

Logical Not :

```
If(!x >0) // true on if expression is false
```

3.7 pinMode(pin,mode)

Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

```
pinMode(pin, Output); //sets pin to output
```

Arduino digital pins default to inputs , so they don't need to be explicitly declared as inputs with `pinMode()`. Pin configured as input are said to be in a high impedance state.

There are also convenient 20 K ohm pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner.

```
digitalRead(pin)

pinMode(pin,INPUT);    //set 'pin' to input

digitalWrite(pin, High); //turn on pullup resistors
```

Pullup resistors would normally be used for connecting inputs like switches. Notice in the above example it does not convert pin to an output, it is merely a method for achieving the internal pull ups.

3.8 digitalRead(pin)

Reads the value from the specified digital pin with the result either high or low. The pin can be specified as either a variable or constant

```
Value = digitalRead(pin) ; //sets value equal to the input pin
```

3.9 digitalWrite(pin, Value)

Outputs either logical level High or Low at a specified digital pins. The pin can be specified as either a variable or constant.

```
Digitalwrite(pin, High) ;
```

The following example reads a pushbutton connected to a digital input and turns on an LED connected to a digital output when the button has been pressed :

```
Int led =13;    //connect LED to pin 13

Int pin = 7;    //connect pushbutton to pin 7

Int value = 0;  //variable to store the read value

Void setup()

{
```

```

        pinMode(led, Output);

        pinMode(pin, Output);
    }

    void loop()
    {
        Value= digitalRead(pin);

        digitalWrite(led, value);
    }

```

3.10 Delay(ms)

Pauses program for the amount of time as specified in milliseconds , where 1000 equals 1 second.

```
delay(1000); //wait for 1 second
```

3.11 Serial.begin(rate)

Opens Serial port and sets the baud serial data transmission. The typical baud rate for communicating with the computer is 9600 although other speeds are supported.

```

void setup()
{
    Serial.begin(9600); // opens serial port

                        //sets data rate to 9600 bps
}

```

When using serial communication, digital pins 0(RX) and 1(TX) cannot be used at the same time.

3.12 Serial.println(data)

Prints data to the serial port, followed by an automatic carriage return and line feed. This command take the same form as Serial.Print(), bt easier for reading data on the serial monitor.

```
Serial.println(analogValue); // sends the value of the analog value
```

Note : For more information on the various permutations of the Serial.println() and Serial.print() functions please refer to the Arduino website.

The following simple example takes a reading from the analog pin0 and sends this data to the Computer every 1 second.

```
Void setup()
```

```
{
```

```
    Serial.begin(9600); //sets serial to 9600 bps
```

```
}
```

```
Void loop()
```

```
{
```

```
    Serial.println(analogRead(0)); //sends analog value
```

```
    delay(1000);           //pauses for 1 second
```

```
}
```

Code Generation

Here load cell sensor before set up the scale with read average , get value and get units .after the runing the scale set up new scale value with read average, get value and get units that get from load cell.

The get units value is weight measurement value.

```
#include <SD.h> //Load Sd card library
```

```
#include <SPI.h> // Load SPI Library
```

```
#include "HX711.h"
```

```
#include <LiquidCrystal.h>
```

```
#include <DS1307RTC.h>
```

```

#include <Time.h>

#include <TimeLib.h>

#include <Wire.h>

LiquidCrystal lcd(8, 7, 5, 4, 3, 2);

HX711 scale(A1, A0); // HX711.DOUT - pin #A1
                    // HX711.PD_SCK - pin #A0

int chipSelect =10; //chipSelect pin for the SD card Reader

char timedatebuf[65];

int year4digit;

File weightData;//Data object write sensor data

void setup() {

  Serial.begin(9600);

  Serial.println(" Intializing SD card.....");

  pinMode(chipSelect,OUTPUT); //must declare 10 an output and reserve it

  if (SD.begin(10))

  {

    Serial.println("SD card is ready to use.");

  }
}

```

```
else
{
  Serial.println("SD card initialization failed");
  return;
}

lcd.begin(16, 2);

lcd.setCursor(4, 0);

lcd.print("Arduino");

lcd.setCursor(0, 1);

lcd.print("load cell sensor");

delay(5000);

lcd.clear();

//lcd.setCursor(2, 0);

//lcd.print("-Setting.");

// Serial.println("Arduino load cell sensor");

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

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

Serial.println(scale.read());

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

Serial.println(scale.read_average(20));
```



```
//Serial.print("get value: \t\t");  
Serial.println(scale.get_value(5));  
  
//Serial.print("get units: \t\t");  
Serial.println(scale.get_units(5), 1);  
  
scale.set_scale(2280.f);  
scale.tare();  
delay(5000);  
  
Serial.println("After setting up the scale:");  
  
Serial.print("read: \t\t");  
Serial.println(scale.read());  
  
Serial.print("read average: \t\t");  
Serial.println(scale.read_average(20));  
Serial.print("get value: \t\t");  
Serial.println(scale.get_value(5));  
  
Serial.print("get units: \t\t");  
Serial.println(scale.get_units(5), 1);  
// Serial.println("Readings:");  
lcd.clear();
```

```
}
```

```
void loop() {
```

```
    tmElements_t tm;
```

```
float A=scale.get_units(10);
```

```
Serial.print("one reading:\t");
```

```
//Serial.print(A, 1);
```

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

```
Serial.println((A*1.046), 1);
```

```
//Serial.println("\t Positive : \t");
```

```
scale.power_down();
```

```
delay(100);
```

```
scale.power_up();
```

```
if(A<1000){
```

```
    Serial.println((A*1.046),1);
```

```
lcd.setCursor(1, 0);
```

```
lcd.print("Weight Average");
```

```
lcd.setCursor(4, 1);
```

```

lcd.print((A*1.046),1);

lcd.print(" g ");

if(RTC.read(tm)){

//tm.Year = CalendarYrToTm(Year);

year4digit=tmYearToCalendar(tm.Year);

sprintf(timedatebuf," Time : %02d:%02d:%02d Date : %02d/%02d/%02d
",tm.Hour,tm.Minute+7,tm.Second,tm.Day,tm.Month,year4digit);

}

else{

// Serial.print("overload : \t");

lcd.setCursor(1,0);

lcd.print(" Overload.. ");

}

delay(5000);

weightData = SD.open("DataLog.txt",FILE_WRITE);

if( weightData){

weightData.print(timedatebuf);

weightData.print(" ");

weightData.print((A*1.046),1);

weightData.print(" ");

weightData.println(" g ");

// Serial.print(timedatebuf);

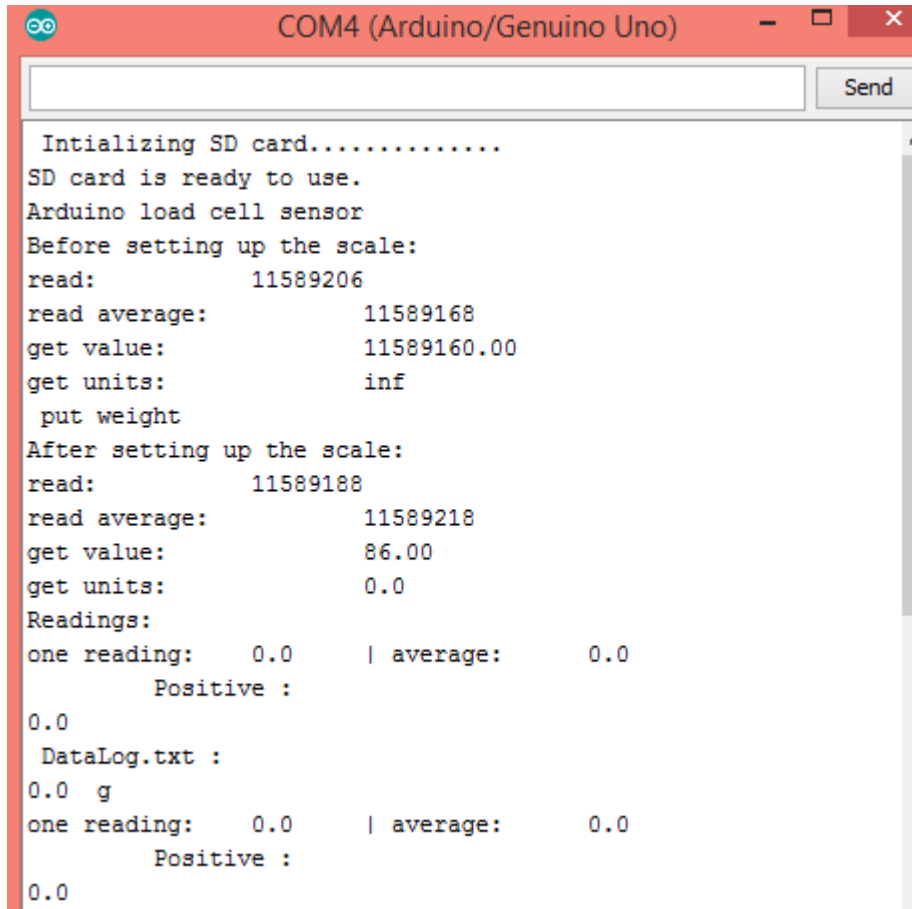
weightData.close();

```

```
}

weightData = SD.open("DataLog.txt");
if(weightData){
  Serial.println(" DataLog.txt : ");
  while(weightData.available()){
    Serial.write(weightData.read());
  }
  weightData.close();
}
else{
  Serial.println("error Opening DataLog.txt");
}
}
delay(5000);
}
```

Output :



```
Intializing SD card.....
SD card is ready to use.
Arduino load cell sensor
Before setting up the scale:
read:      11589206
read average: 11589168
get value:  11589160.00
get units:  inf
put weight
After setting up the scale:
read:      11589188
read average: 11589218
get value:  86.00
get units:  0.0
Readings:
one reading: 0.0 | average: 0.0
           Positive :
0.0
DataLog.txt :
0.0 g
one reading: 0.0 | average: 0.0
           Positive :
0.0
```

Here read the weight and get the average weight .it is positively measure the weight and also save the data in a Datalog.txt that contains Datalog .txt file.

After 5s it measure the data and save the data into file.

Time : 17:30:51	Date : 06/12/2016	weight	0.2	KG
Time : 17:31:02	Date : 06/12/2016	weight	0.0	KG
Time : 17:31:13	Date : 06/12/2016	weight	0.7	KG
Time : 17:31:24	Date : 06/12/2016	weight	0.4	KG
Time : 17:31:36	Date : 06/12/2016	weight	0.2	KG
Time : 17:31:47	Date : 06/12/2016	weight	0.2	KG
Time : 17:31:59	Date : 06/12/2016	weight	0.2	KG
Time : 17:32:10	Date : 06/12/2016	weight	0.0	KG
Time : 17:32:22	Date : 06/12/2016	weight	0.0	KG
Time : 17:32:34	Date : 06/12/2016	weight	0.0	KG
Time : 17:32:46	Date : 06/12/2016	weight	0.7	KG
Time : 17:32:57	Date : 06/12/2016	weight	0.5	KG
Time : 17:33:09	Date : 06/12/2016	weight	0.0	KG
Time : 17:33:21	Date : 06/12/2016	weight	0.0	KG
Time : 17:33:33	Date : 06/12/2016	weight	0.0	KG
Time : 17:33:45	Date : 06/12/2016	weight	0.0	KG
Time : 17:33:57	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:09	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:22	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:34	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:46	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:59	Date : 06/12/2016	weight	0.0	KG
Time : 17:35:11	Date : 06/12/2016	weight	0.0	KG
Time : 17:35:24	Date : 06/12/2016	weight	0.0	KG
Time : 17:35:37	Date : 06/12/2016	weight	0.0	KG
Time : 17:35:49	Date : 06/12/2016	weight	0.2	KG
Time : 17:36:02	Date : 06/12/2016	weight	0.1	KG
Time : 17:34:45	Date : 06/12/2016	weight	0.0	KG
Time : 17:34:58	Date : 06/12/2016	weight	0.0	KG
Time : 17:35:11	Date : 06/12/2016	weight	0.5	KG
Time : 17:35:24	Date : 06/12/2016	weight	0.0	KG

Figure : Data Logging

Chapter 4

Conclusion

In this Chapter we summarize the current work. Section 5.1 presents the contributions of our proposed system. The limitations of the proposed system discussed in section 5.2

4.1 Contribution of the Proposed System

The biggest advantage of this system is, it is ready to use its structure. As Arduino comes in a complete package form which includes the 5V regulator, a burner, an oscillator, a micro-controller, serial communication interface, LED and headers for the connections. one don't have to think about programmer connections for programming or any other interface. Just plug it into USB port of personal computer and that's it. The revolutionary idea is going to change the world after just few words of coding.

There are many forums present on the internet in which people are talking about the Arduino. Engineers, hobbyists and professionals are making their projects through Arduino. You can easily find help about everything. Moreover the Arduino website itself explains each and every functions of Arduino.

So, We should conclude the advantage of Arduino by saying that during working on different projects one just have to worry about one's innovative idea. The remaining will handle by Arduino itself.

4.2 Limitation of the propose system

The most important factor which we cannot deny is cost. This is the problem which every hobbyist, Engineer or Professional has to face. Now, we must consider that the Arduino is cost effective or not.

During building a project one has to make its size as small as possible. But with the big structures of Arduino we have to stick with big sized PCB's. If any one is working on a small micro-controller like ATmega8 one can easily make one's PCB as small as possible.

References

- 1) <http://www.myarduino.net/product/60/hx711-weight-sensor-amplifier-module-dual-channel-hx711-for-load-cell>
- 2) <http://www.loadstarsensors.com/what-is-a-load-cell.html>
- 3) <https://video.search.yahoo.com/yhs/search?fr=yhs-Lkry-newtab&hsimp=yhsnewtab&hspart=Lkry&p=how+to+set+time+and+date+sd+card+with+ardui+no+code#id=2&vid=15bc02c481ab08e4936773aa47da6d4b&action=click>
- 4) <https://www.arduino.cc>
- 5) <https://www.arduino.cc/en/Reference>
- 6) <https://create.arduino.cc/projecthub/arjun/programming-attiny85-with-arduino-uno-afb829>
- 7) <http://airtripper.com/1626/arduino-load-cell-circuit-sketch-for-calibration-test/>
- 8) <https://youtu.be/OQqZYxSIRhA>
- 9) <https://youtu.be/nGUpzwEa4vg>