Contents

[A. Basic Concepts 2](#_Toc398810442)

[1. Design Advantages 2](#_Toc398810443)

[2. Design Tools 3](#_Toc398810444)

[3. The 4 Major Components 4](#_Toc398810445)

[4. Additional Components 5](#_Toc398810446)

[5. Adding Sensors 6](#_Toc398810447)

[B. ‘Simple’ Version: 1.0 (Thru-hole mounting only) 7](#_Toc398810448)

[1. Block Diagram 1.0 7](#_Toc398810449)

[2. Wiring Schematic 1.0 8](#_Toc398810450)

[3. Board Layout 1.0 9](#_Toc398810451)

[4. Bill of Materials 1.0 10](#_Toc398810452)

[5. Code 1.0 11](#_Toc398810453)

[C. More Precise Version (±0.02% or better) 16](#_Toc398810454)

[1. A Few Changes 16](#_Toc398810455)

[2. Block Diagram 2.0 17](#_Toc398810456)

[3. Wiring Schematic 2.0 18](#_Toc398810457)

[4. Board Layout 2.0 19](#_Toc398810458)

[5. Code 2.0 20](#_Toc398810459)

[6. Bill of Materials 2.0 21](#_Toc398810460)

# Basic Concepts

## Design Advantages

This Arduino module is designed to provide the platform for a very low power consumption data logger while retaining the functionality and convenience of an Arduino Uno (or Micro in this case). Its major design advantages include

* Easy to Program
* Modular Components
* Low Power Consumption (~30 uA in Sleep Mode)
* Flexible Power Source (Anywhere between 1.8-5.5 V)
* Accurate, temperature adjusted clock with alarm
* Storage directly to microSD card (up to 32, 64+ GB storage if desired)
* Single platform for multiple digital and/or analog sensors
* Low Cost (~$75 plus cost of sensors)
* Option to include 16-bit ADC for ± .02% baseline accuracy (additional ~$10)

## Design Tools

The design here can be built and/or modified using all free software. The following tools and sites make design very simple and easy.

* **EAGLE PCB**: Software for designing Board Layouts and Schematics plus producing Gerber Files for manufacturing. Watch Youtube tutorials from Jeremy Blum to learn the how to create your own PCBs in a few days. (The provided board files and schematics have been made with Eagle “Light”.) <http://www.cadsoftusa.com/eagle-pcb-design-software/product-overview/>
* **My4pcb.com**: Free error check on your PCBs for DFM (Design for Manufacturing). This site lets you know in a few minutes if your board design is valid and provides .pdf files for all board layers. <https://www.my4pcb.com/net35/FreeDFMNet/FreeDFMHome.aspx>
* **SeeedStudio.com**: Very cheap PCB manufacturer. Lead times are long (~3 weeks), but very cost effective for prototyping and small batch creation ($2.00-$5.00 per PCB). <http://www.seeedstudio.com/service/index.php?r=pcb>
* **Adafruit.com**: Online merchant selling Arduino boards and breakout modules
* **Sparkfun.com**: Online merchant selling Arduino boards and breakout modules
* **GitHub.com**: Site for obtaining libraries and code for Arduino programming
* **Digikey.com**: Online merchant selling electronics components

## The 4 Major Components

The module uses four “breakout boards” which may be hard wired or attached to the base PCB with headers (plug-in). The modular design allows for interchangeability of parts, simplifies circuit design, and minimizes assembly cost by utilizing smaller pre-wired circuits. The four basic modules used here are as follows:

1. **Arduino Micro** with Headers - 5V 16MHz - (ATmega32u4 - assembled)  
   <https://www.adafruit.com/products/1086>  
   The Arduino Micro provides an easily programmable micro-controller as its brain. The Arduino Micro was selected over an Arduino Uno for its smaller footprint, though both possess the same basic functionality. (The only changes that need be made for an Arduino Uno are to re-wire the SPI pins from MOSI, MISO, and SCK on the Micro to pins 11, 12, and 13 respectively on the Uno)
2. **PowerBoost 500 Basic** - 5V USB Boost @ 500mA from 1.8V+  
   <https://www.adafruit.com/products/1903?gclid=CMS71dH52b8CFc5afgodggIAIg>  
   The PowerBoost Card utilizes a power converter with shutdown option to provide power to the module. The power converter can accept any voltage from 1.8-5.5V, so the battery can be changed as desired with ~90% efficiency in most cases. The “Enable” (Shutdown) pin on the card allows power to be completely cut off from the Arduino board, so it uses only ~30uA when not actively taking measurements.
3. **DeadOn RTC** - DS3234 Breakout  
   <https://www.sparkfun.com/products/10160>  
   The DS3234 Real Time Clock provides a clock to the unit with alarms set using control registers. The alarm runs only on the small watch battery when the unit is off, conserving the main power source. The DS3234 has a built-in temperature adjustment which should minimize drift over long-term deployments
4. Breakout Board for **microSD Transflash**  
   <https://www.sparkfun.com/products/544>  
   The microSD card is straightforward. It provides data storage up to 128GB if desired using SDHC cards, though even 2GB microSD Cards should provide adequate storage in most cases and can be acquired for under $5.

## Additional Components

In addition to the 4 modules listed above, only a few components need to be added:

* A basic **Signal Inverter** converts the voltage from the Real Time Clock to the Enable Pin on the Boost Converter to the proper value (0V when sleeping, VBat when alarm trips). The Inverter uses transistors rather than active switching so uses only ~1uA of power when the unit is sleeping.
* 1 ea. **pull-up/down resistors** for the Signal Inverter
* **Bypass capacitors** between battery-ground and 5V-ground to reduce signal noise.
* **2 Diodes** to act as valves for power converter “shutdown”
* A **Voltage Regulator** to provide more precise voltage for analog readings. The Arduino’s built in 5V output is not very precise. A basic LM4040 diode and 560 Ohm resistor can be used to provide accuracy of ± 0.25% (<http://www.skillbank.co.uk/arduino/measure.htm>) . This accuracy is also similar to that of the Arduino’s built in 10-bit ADC. If using the LM4040, the 4.096 voltage output must be wired to the Arduino’s ‘VRef’ pin and all our calculations must be based off the 4.096V value. A design for applications requiring more precise measurements (less than the 0.25% error here) is provided later on.

The above components can be placed on a simple PCB, requiring only thru-hole mounting if one wishes to keep assembly simple. Such a design is provided below.

## Adding Sensors

The only remaining task is to add the sensor(s) to the module. Almost any sensor(s) can be added to the module, whether analog or digital. Proper care must only be taken that enough analog/digital pins exist on the Arduino to measure the sensors and that they are switched on and off at the appropriate times so that signals and voltages do not mix.

A very simple design for a temperature sensor is provided below. It uses a basic voltage divider with a 10K resistor and a 10K thermistor to take a voltage reading at the junction. Here we used low tolerance (0.01%) resistor for the 10K voltage divider to increase baseline accuracy. The Arduino measures the voltage, which can be converted into a temperature in accordance with the thermistor’s characteristics. In this case, we chose a **PS103J2 Thermistor** from US Sensors. All PS103J2 thermistors are interchangeable within ± 0.1° C, so we should be able to obtain that degree of accuracy without any calibration using the resistor-temperature conversion chart.

# ‘Simple’ Version: 1.0 (Thru-hole mounting only)

## Block Diagram 1.0

A block diagram of the above described module is provided below. It should give you the basic idea before we get too complicated with the wiring schematic

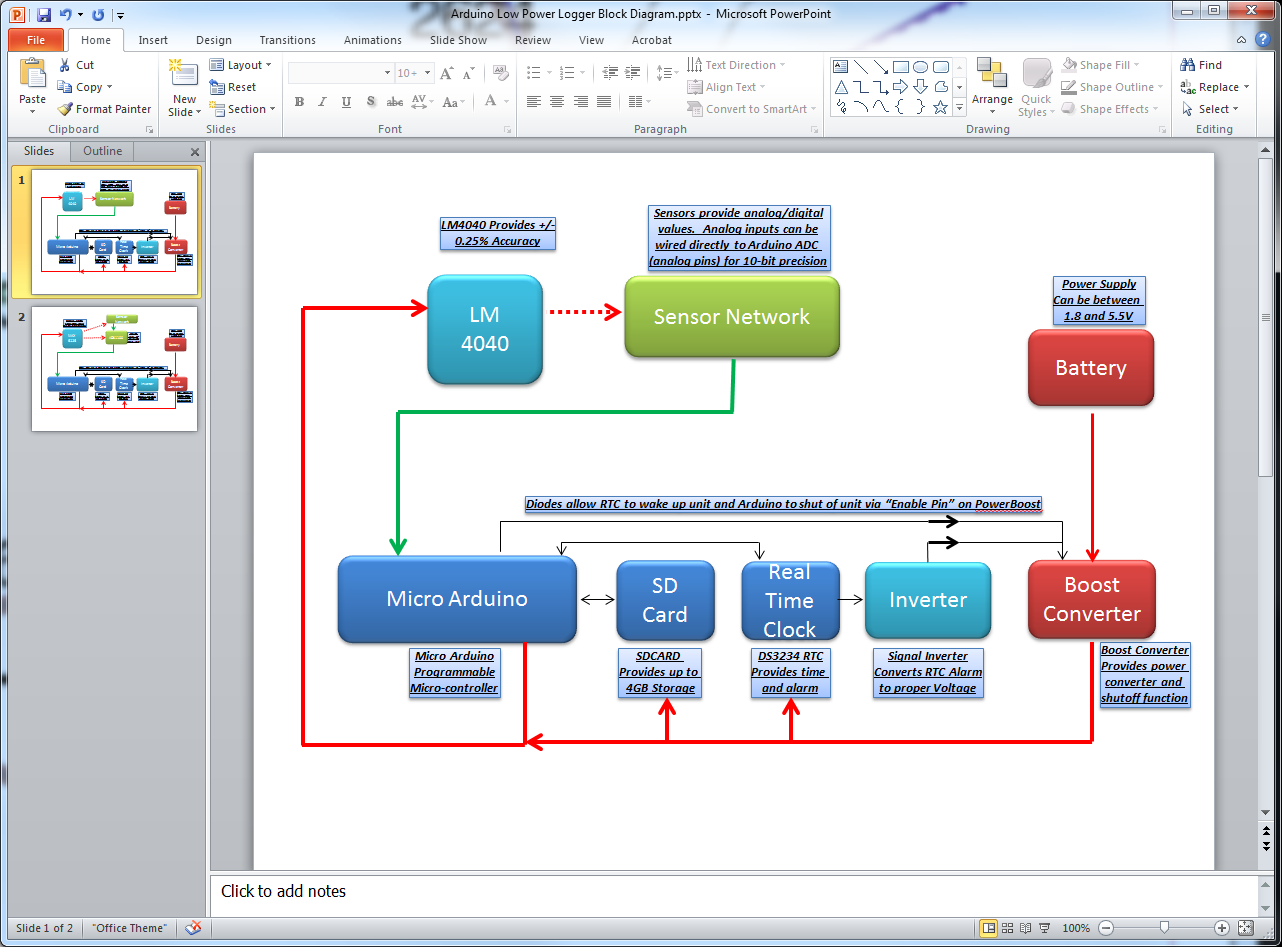


Figure1: Block Diagram (Simple Version)

## Wiring Schematic 1.0

If the above block diagram makes sense, now we’re ready for the actual wiring of the device. This includes all necessary components.

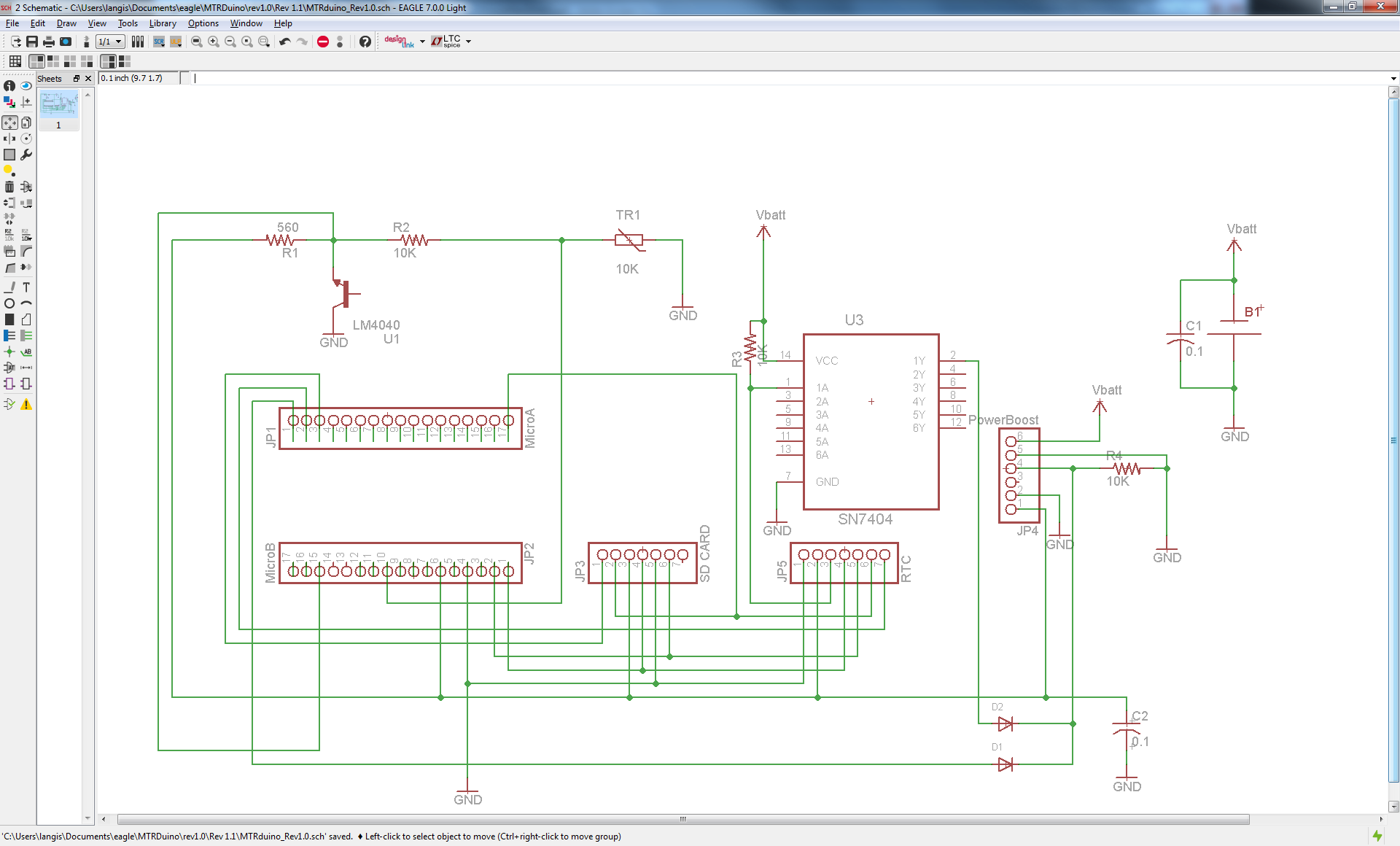


Figure 2: Wiring Schematic (Simple Version)

## Board Layout 1.0

A sample board layout is provided below. This design uses both sides of the board for mounting components to minimize the footprint and enable it to fit in a very small case. A BOM is also provided with components which will fit the board layout. It also uses only Thru-Hole components to make in-house assembly very easy.

The only additional item of note is that the thermistor ground plane is isolated from the ground plane of all other components to reduce ground noise effects. The two planes are connected by a single point of contact.

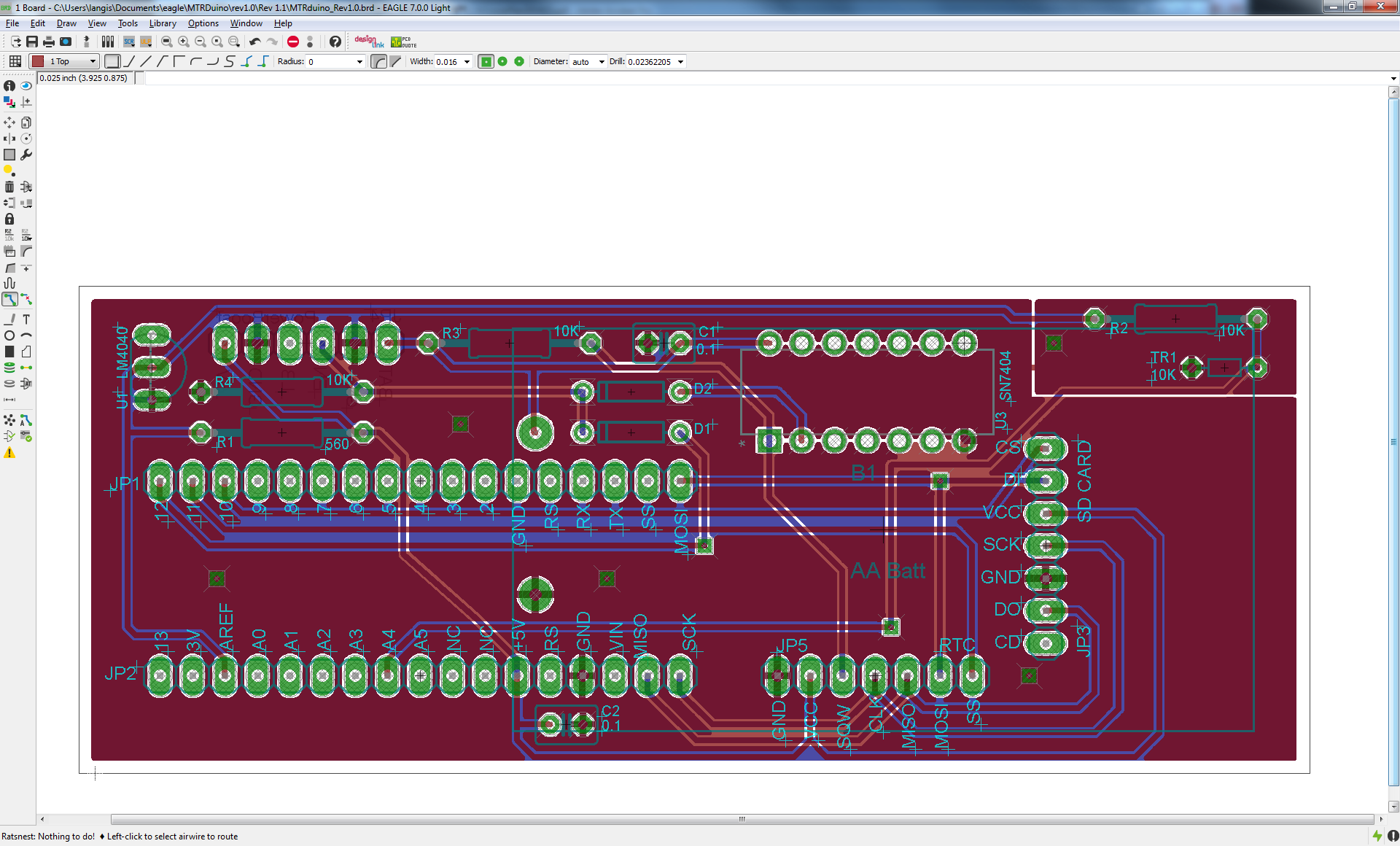


Figure 3: Board Schematic (Simple Version)

## Bill of Materials 1.0

Yep. (It’s also included as an excel file so you can work with it).



## Code 1.0

The following code can be used for the above wiring diagram. It is heavily commented so it should be mostly self-explanatory. There are still a few areas of note:

* The code is written in two parts so that users are able to “program” the device when initially uploading the code to the Arduino. The two parts are conveniently broken up into the Setup{} and Loop{} routines.
  1. The first section of code (Setup {}) boots up the Arduino, resets the alarm, takes a number of measurement from the sensor(s), averages the measurements, stored the data on the SDCard, and then sends a command to shut down power to the device.
  2. The second section of code (Loop {}) only runs IF the Arduino is connected to a serial/USB port. Then power will still be applied and allow users to check functionality, adjust the unit’s time/date, alarm, etc. before deployment. The user should upload the code as normal with an Arduino board and press “Ctrl+Shift+M” to open the Serial Communication window.
* The code to program the RTC is fairly cryptic. It involves setting a number of registers in the DS3234 chip using SPI communication and some bit-math to do so. If you wish to modify how the alarms function, refer to the DS3234 datasheet and expect some trial-and-error.
* The ANALOGREFERENCE command is used because we are utilizing the LM4040 here as our reference. All our measurements and calculations are based off its 4.096 output.
* A few variables are included near the start of the “logging” variables in the code just before the Setup{} routine
  1. Time Interval between samples, in seconds
  2. Filename, and
  3. Number of Measurements to average for each sample

//include required header files for Arduino funtions, SPI functionality, DS3234 clock, and SD Card

#include <ctype.h>

#include <Wire.h>

#include <ArduinoPins.h>

#include <SPI.h>

#include <SD.h>

#include <RTClib.h>

#include <RTC\_DS3234.h>

// Avoid spurious warnings

#undef PROGMEM

#define PROGMEM \_\_attribute\_\_(( section(".progmem.data") ))

#undef PSTR

#define PSTR(s) (\_\_extension\_\_({static prog\_char \_\_c[] PROGMEM = (s); &\_\_c[0];}))

#define BUFF\_MAX 256

#define RTCPIN 10 //Define Output Pin for RTC

#define SDPIN 9 //Define Output Pin for SDCARD

#define POWERPIN 12 //Define Digital Output Pin for Power Valve

#define THERMPIN 0 //Define Analog Input Pin for Thermistor Reading

#define CREG B00000101 //Define Control Register byte for DS3234 RTC

const float VREF **=** 4.096**;** //Define Voltage Reference from LM4040 Diode

RTC\_DS3234 RTC**(**RTCPIN**);** // Create a RTC instance, using the chip select pin it's connected to

//\*\*\*\*\*\*\*\*\*\*\* SET LOGGING VARIABLES HERE: \*\*\*\*\*\*\*\*\*\*\*

const int sampleinterval **=** 60**;**

const char filename**[]** **=** "data.txt"**;**

const int numSamples **=** 10**;**

//\*\*\*\*\*\*\*\*\*\*\* SET LOGGING VARIABLES HERE: \*\*\*\*\*\*\*\*\*\*\*

void setup **()** **{**

Wire**.**begin**();** // Start Arduino

pinMode**(**POWERPIN**,**OUTPUT**);** // setup Power Valve Pin

digitalWrite**(**POWERPIN**,** HIGH**);** // Set Power Pin High so Arduino stays on after RTC alarm is reset

pinMode**(**THERMPIN**,**INPUT**);** // setup Thermistor Pin

pinMode**(**RTCPIN**,**OUTPUT**);** // setup Real Time Clock Chip

SPI**.**begin**();** // Start SPI functionality to talk to SD Card and RTC

SPI**.**setBitOrder**(**MSBFIRST**);** // Set bitorder for SPI (required for setting registers on DS3234 RTC)

RTC**.**begin**();** // Start Real Time Clock

analogReference**(**EXTERNAL**);** // set analog reference to 4.096V from LM4040 voltage regulator diode

DateTime now **=** RTC**.**now**();** //Get time from RTC

DateTime alarmtime **=** **(**now**.**unixtime**()** **+** sampleinterval**);**

setAlarmTime**(**alarmtime**);**

float therm\_val **=** 0**;**

**for** **(**int j **=** 0**;**j **<** numSamples**;**j**++){** //Take a number of samples and everage them to reduce noise

delay**(**10**);**

therm\_val **+=** analogRead**(**THERMPIN**)** **;** //read value from analog input for thermistor

**}**

float therm\_voltage **=** therm\_val **/** 1023.0 **\*** VREF **/** numSamples**;** //divide accumulated thermistor reading by 1024 (10bit ADC), multiply by 4.096V reference, and average

float therm\_resistance **=** 10000.0**/(**VREF**/**therm\_voltage**-**1.0**);** //Calculate value of thermistor resistance based on voltage divider circuit

SDCardWrite**(**therm\_resistance**,**now**);**

//Drive output to power pin low (go to sleep and wake up on RTC alarm only)

digitalWrite**(**POWERPIN**,** LOW**);**

delay**(**2000**);**

**}**

void loop **()** **{**

//\*\*\*\*\*\*\*This Section of code will only run if arduino is plugged into a serial port (it will have shut off already otherwise);

//Measure value and reset Alarm

//Do this every time we change settings (when the loop runs) just to be safe.

DateTime now **=** RTC**.**now**();** //Get time from RTC

DateTime alarmtime **=** **(**now**.**unixtime**()** **+** sampleinterval**);**

setAlarmTime**(**alarmtime**);**

const int len **=** 32**;** //buffer length for RTC display on Serial comm

static char buf**[**len**];** //string for RTC display on Serial comm

float therm\_val **=** analogRead**(**THERMPIN**);** //read value from analog input for thermistor

float therm\_voltage **=** therm\_val **/** 1023.0 **\*** VREF**;**

float therm\_resistance **=** 10000.0**/(**VREF**/**therm\_voltage**-**1.0**);** //Calculate value of thermistor resistance based on voltage divider circuit

Serial**.**begin**(**9600**);** // Turn the Serial Protocol ON

**while(!**Serial**){}** //Wait till serial connection is established

//"OPTIONS" Menu

Serial**.**println**(**"'S' to display Status"**);**

Serial**.**println**(**"'T' to sync time"**);**

Serial**.**println**(**"'A' to set first alarm date"**);**

Serial**.**println**();**

byte byteRead**;** //variable to read Serial input from user

boolean waiting**=**true**;** //variable to wait for user input

**while(**waiting**==**true**){** //Run loop to set unit status as long as serial connection exists

byteRead **=** Serial**.**read**();** //read serial input (keystroke from user)

**switch** **(**byteRead**){**

**case** 84 **:** //if user enters 'T', sync the RTC

//code

RTC**.**adjust**(**DateTime**(**\_\_DATE\_\_**,** \_\_TIME\_\_**));**

Serial**.**println**(**"\*\*Time Synchronized with System Clock!"**);**

waiting**=**false**;**

**case** 83 **:** //if user enters 'S', display logger "Status"

now **=** RTC**.**now**();**

Serial**.**print**(**"Current RTC time: "**);**

Serial**.**println**(**now**.**toString**(**buf**,**len**));**

Serial**.**print**(**"First alarm time: "**);**

Serial**.**println**(**alarmtime**.**toString**(**buf**,**len**));**

Serial**.**print**(**"Sample Interval: "**);**

Serial**.**print**(**sampleinterval**);**

Serial**.**println**(**" seconds"**);**

Serial**.**print**(**"Measurements per Sample : "**);**

Serial**.**print**(**sampleinterval**);**

Serial**.**println**(**" (Change Sample Interval or # of Measurements in code if needed)"**);**

Serial**.**println**(**"SD Card Status: "**);**

SDCardCheck**();**

Serial**.**print**(**"Thermistor Value: "**);**

Serial**.**println**(**therm\_resistance**,**2**);**

Serial**.**println**();**

waiting**=**false**;**

**break;**

**case** 65 **:** //'A'

Serial**.**println**(**"Enter Desired Alarm Date:"**);**

Serial**.**println**(**"MMDDYY"**);**

char inputbuffer**[**6**];**

boolean checkdigit **=** true**;** //variable to check if user input is actually digits

boolean waiting2**=**true**;** //another variable to wait for user input

byte readLen **=**0**;**

byte MM**;** //variables to store time and date values

byte DD**;**

byte YY**;**

byte hh**;**

byte mm**;**

byte ss**;**

**while(**waiting2**==**true**){**

readLen **=** Serial**.**readBytes**(**inputbuffer**,** 6**);** //read serial input (6 keystrokes from user)

**if** **(**readLen**==**6**){** //check if user has put in 6 digits correctly

waiting2**=**false**;**

**for(**int i **=** 0**;** i**<**6**;**i**++){**

**if(**inputbuffer**[**i**]<**48**||**inputbuffer**[**i**]>**57**){**

checkdigit**=**false**;**

**}**

**}**

**}**

**}**

**if(**checkdigit**==**true**){**

MM**=**10**\*(**inputbuffer**[**0**]-**48**)+(**inputbuffer**[**1**]-**48**);** //parse input data into vaiables

DD**=**10**\*(**inputbuffer**[**2**]-**48**)+(**inputbuffer**[**3**]-**48**);**

YY**=**10**\*(**inputbuffer**[**4**]-**48**)+(**inputbuffer**[**5**]-**48**);**

Serial**.**print**(**MM**);**

Serial**.**print**(**DD**);**

Serial**.**println**(**YY**);**

**}**

**else{**

Serial**.**println**(**"Input Error"**);**

**}**

waiting2**=**true**;** //reset variable waiting for user input

**if(**checkdigit**==**true**){**

Serial**.**println**(**"Enter Desired Alarm Time:"**);**

Serial**.**println**(**"hhmmss"**);**

**while(**waiting2**==**true**){**

readLen **=** Serial**.**readBytes**(**inputbuffer**,** 6**);** //read serial input (6 keystrokes from user)

**if** **(**readLen**==**6**){** //check if user has put in 6 digits correctly

waiting2**=**false**;**

**for(**int i **=** 0**;** i**<**6**;**i**++){**

**if(**inputbuffer**[**i**]<**48**||**inputbuffer**[**i**]>**57**){**

checkdigit**=**false**;**

**}**

**}**

**}**

**}**

**if(**checkdigit**==**true**){**

hh**=**10**\*(**inputbuffer**[**0**]-**48**)+(**inputbuffer**[**1**]-**48**);** //parse input data into vaiables

mm**=**10**\*(**inputbuffer**[**2**]-**48**)+(**inputbuffer**[**3**]-**48**);**

ss**=**10**\*(**inputbuffer**[**4**]-**48**)+(**inputbuffer**[**5**]-**48**);**

Serial**.**print**(**hh**);**

Serial**.**print**(**mm**);**

Serial**.**println**(**ss**);**

**}**

**else{**

Serial**.**println**(**"Input Error"**);**

**}**

**}**

**if(**checkdigit**==**true**){** //if both inputs were six digits, then reset the alarm with the right values

int YEAR **=** **(**int**)**YY **+** 1900**;**

alarmtime **=** DateTime**(**ss**,**mm**,**hh**,**DD**,**MM**,**YEAR**);**

setAlarmTime**(**alarmtime**);**

Serial**.**println**(**"Alarm Set Successfully!"**);**

**}**

waiting**=**false**;**

**break;**

**}**

**}**

delay**(**2000**);** //wait a few seconds before running the loop again

**}**

//Routine to Write data to RTC Register VIA Arduino SPI

void RTCWrite**(**char reg**,** char val**){**

noInterrupts**();** //make sure transfew doesn't get interrupted

digitalWrite**(**RTCPIN**,** LOW**);** //enable SPI read/write for chip

SPI**.**transfer**(**reg**);** //define memory register location

SPI**.**transfer**(**bin2bcd**(**val**));** //write value

digitalWrite**(**RTCPIN**,** HIGH**);** //disable SPI read/write for chip

delay**(**10**);** //delay 10 ms to make sure chip is off

interrupts**();** //resume normal operation

**}**

//Routine to Read data From RTC Register VIA Arduino SPI

char RTCRead**(**char reg**){**

noInterrupts**();** //make sure transfew doesn't get interrupted

char val **=** 0x00**;** //initialize return variable

digitalWrite**(**RTCPIN**,** LOW**);** //enable SPI read/write for chip

SPI**.**transfer**(**reg**);** //define memory register location

val **=** bcd2bin**(**SPI**.**transfer**(-**1**));** //retrieve value

digitalWrite**(**RTCPIN**,** HIGH**);** //disable SPI read/write for chip

delay**(**10**);** //delay 10 ms to make sure chip is off

interrupts**();** //resume normal operation

**return** val**;**

**}**

void setAlarmTime**(**DateTime alarmtime**){**

//set alarm time: seconds

//87=write to location for alarm seconds

//binary & second with 0x7F required to turn alarm second "on"

RTCWrite**(**0x87**,**alarmtime**.**second**()** **&** 0x7F**);**

//set alarm time: minutes

//88=write to location for alarm minutes

//binary & minute with 0x7F required to turn alarm minute "on"

RTCWrite**(**0x88**,**alarmtime**.**minute**()** **&** 0x7F**);**

//set alarm time: hour

//89=write to location for alarm hour

//binary & hour with 0x7F required to turn alarm hour "on"

RTCWrite**(**0x89**,**alarmtime**.**hour**()** **&** 0x7F**);**

//set alarm time: day

//8A=write to location for alarm day

//binary & day with 0x3F required to turn alarm day "on" (not dayofWeek)

RTCWrite**(**0x8A**,**alarmtime**.**day**()** **&** 0x3F**);**

//Set Alarm #2 to zll zeroes (disable)

RTCWrite**(**0x8B**,**0**);**

RTCWrite**(**0x8C**,**0**);**

RTCWrite**(**0x8D**,**0**);**

//reset flags

//8F=write to location for control/status flags

//B00000000=Ocillator Stop Flag 0, No Batt Backed 32 KHz Output, Keep Temp CONV Rate at 64 sec (may change later), disable 32 KHz output, temp Not Busy, alarm 2 not tripped, alarm 1 not tripped

RTCWrite**(**0x8F**,**B00000000**);**

//set control register

//8E=write to location for control register

//B01100101=Oscillator always on, SQW on, Convert Temp off, SQW freq@ 1Hz, Interrupt enabled, Alarm 2 off, Alarm 1 on

RTCWrite**(**0x8E**,**CREG**);**

**}**

//simple routine to check if SD card is functioning properly

void SDCardCheck**(){**

pinMode**(**10**,**OUTPUT**);** //prevent errors

**if** **(!**SD**.**begin**(**SDPIN**))** **{**

Serial**.**println**(**" \*\*SD Card Initialization failed! Check connections and if SD Card is Present"**);**

**return;**

**}**

Serial**.**print**(**" \*\*SD Card Initialized successfully. Saving records to "**);**

Serial**.**println**(**filename**);**

**}**

//routine to write a sensor value and a formatted time/date string to SD Card

void SDCardWrite**(**int val**,**DateTime timestamp**){**

pinMode**(**10**,**OUTPUT**);** //prevent errors

SD**.**begin**(**SDPIN**);**

File dataFile **=** SD**.**open**(**filename**,** FILE\_WRITE**);**

**if** **(**dataFile**)** **{** // if the file is available, write to it:

dataFile**.**print**(**val**);**

dataFile**.**print**(**", "**);**

dataFile**.**print**(**timestamp**.**month**());**

dataFile**.**print**(**"/"**);**

dataFile**.**print**(**timestamp**.**day**());**

dataFile**.**print**(**"/"**);**

dataFile**.**print**(**timestamp**.**year**());**

dataFile**.**print**(**", "**);**

dataFile**.**print**(**timestamp**.**hour**());**

dataFile**.**print**(**": "**);**

dataFile**.**print**(**timestamp**.**minute**());**

dataFile**.**print**(**": "**);**

dataFile**.**println**(**timestamp**.**second**());**

dataFile**.**close**();**

**}**

// if the file isn't open, do nothing

**}**

# More Precise Version (±0.02% or better)

## A Few Changes

By adding a few more components, we can output use a more precise ADC (Analog-to-Digital-Converter), giving us much higher resolution for our analog measurements (0.0001 V increments for a 16-bit ADC with a 5V source). There are many ways of doing this. To get the most precision we will need:

1. **An ADC.** The **ADS1100** is a very simple converter which communicates using I2C communication. To take measurements, we wire the output to the Arduino I2C pins, rather than the analog inputs since we are bypassing the Arduino 10-bit ADC.
2. **A Precise Voltage Regulator.** Some ADCs come with built-in voltage regulators, but the ADS1100 does not. Built-in regulators are simpler for wiring, but can limit overall accuracy. (I couldn’t find a simple ADC with a regulator precise enough for my liking.) Also, this way we can also hook up our sensor network to the precision output to ensure our sensors are getting a reliable source as well. We can select something like the **MAX6126**, which provides 4.096V ±0.02%. It also only requires a minimum 4.3 volts to operate – good news for our board which gives out an only semi-reliable 5V.
3. **A few more resistors and capacitors.** Weshould include a few bypass capacitors to reduce noise and we might use something like a “bridge” network to measure our thermistor rather than the simpler voltage divide circuit used in the above example. Here we used low tolerance (0.1%) resistors for the bridge network to increase baseline accuracy.
4. **A better thermistor.**  We’re trying to get as much accuracy we can, so we move from a “precision” thermistor (± 0.1° C interchangeable) to an “ultra-precision” thermistor (± 0.05° C interchangeable). Again, we chose a **PS103J2 Thermistor** from US Sensors works nicely – it costs a few dollars more, but it’s worth the marginal increase. We should be able to obtain ± 0.05° C degree of accuracy without any calibration simply by using the resistor-temperature conversion chart.

## Block Diagram 2.0

A block diagram of the above described module is provided below. It should give you the basic idea before we get too complicated with the wiring schematic

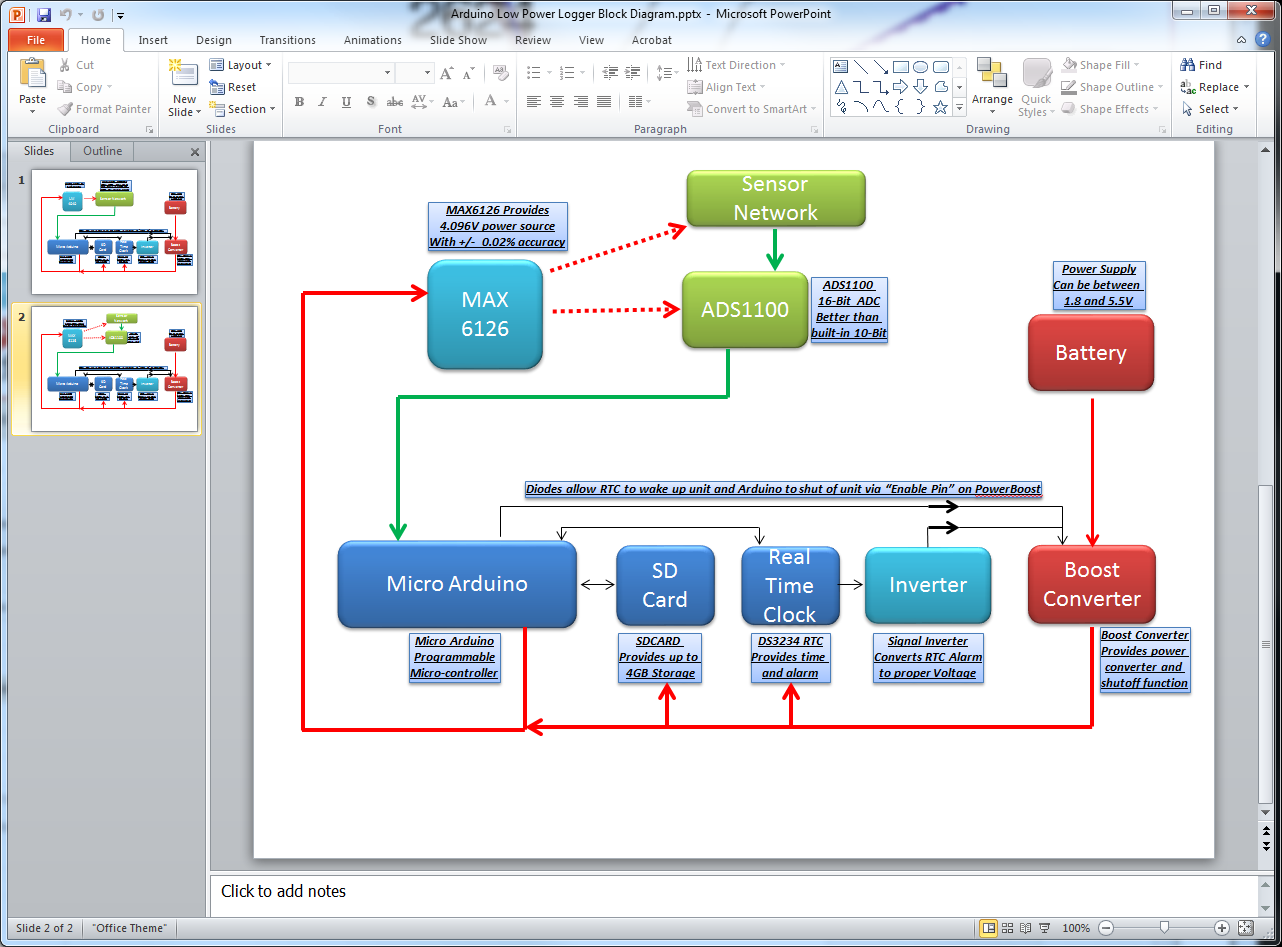


Figure 4: Block Diagram (More Precise Version)

## Wiring Schematic 2.0

If the above block diagram makes sense, now we’re ready for the actual wiring of the device. This includes all necessary components.

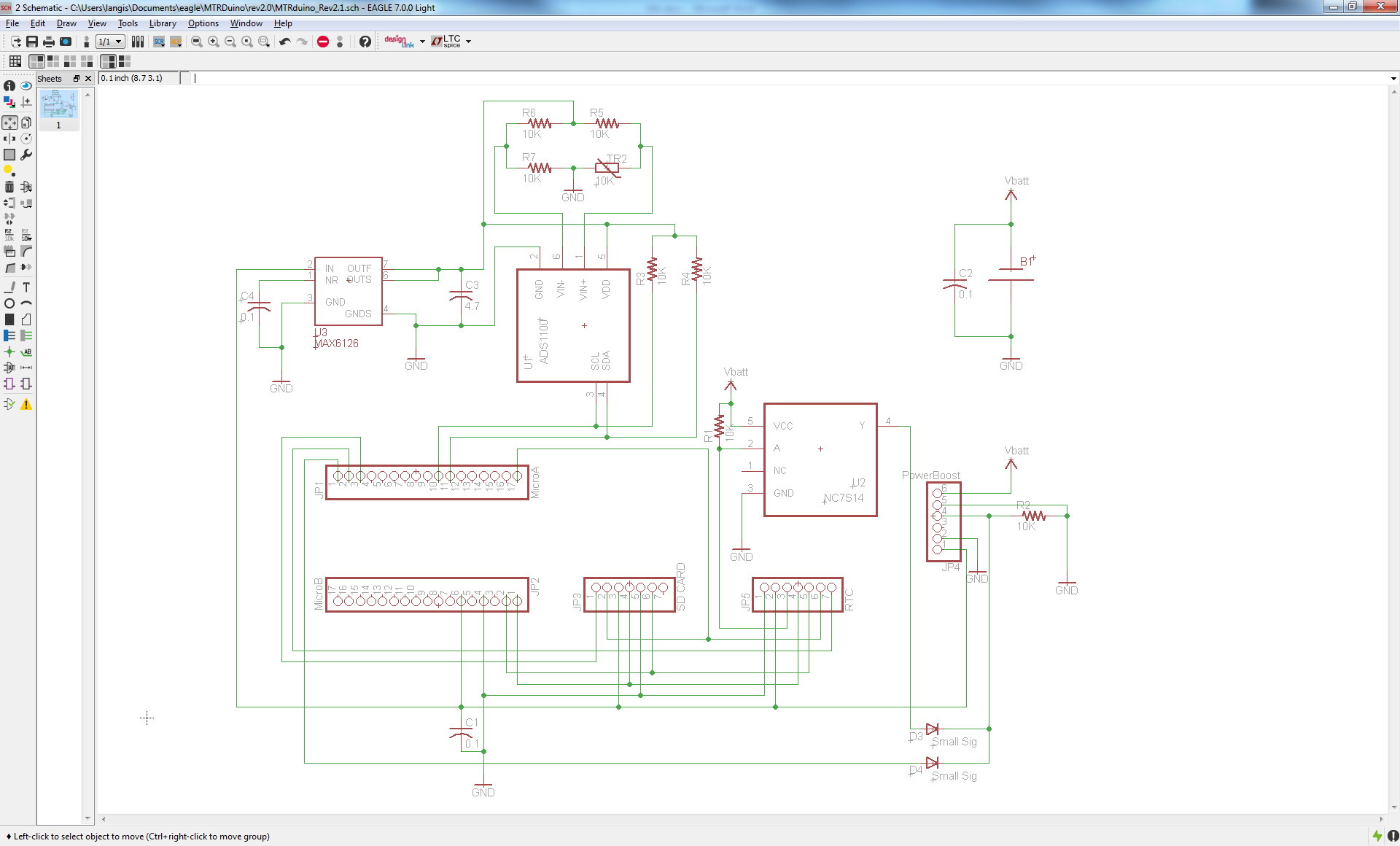


Figure 5: Wiring Schematic (More Precise Version)

## Board Layout 2.0

At this point with all the modifications and the assumption that we are using the device for very precise measurements, it makes sense to use Surface Mount components, which will likely be done by a professional/third party vendor. The resistors, capacitors, diodes, and ICs are all replaced by SMD components in this situation.

As before, the only additional item of note is that the thermistor ground plane is isolated from the ground plane of all other components to reduce ground noise effects. The two planes are connected by a single point of contact.

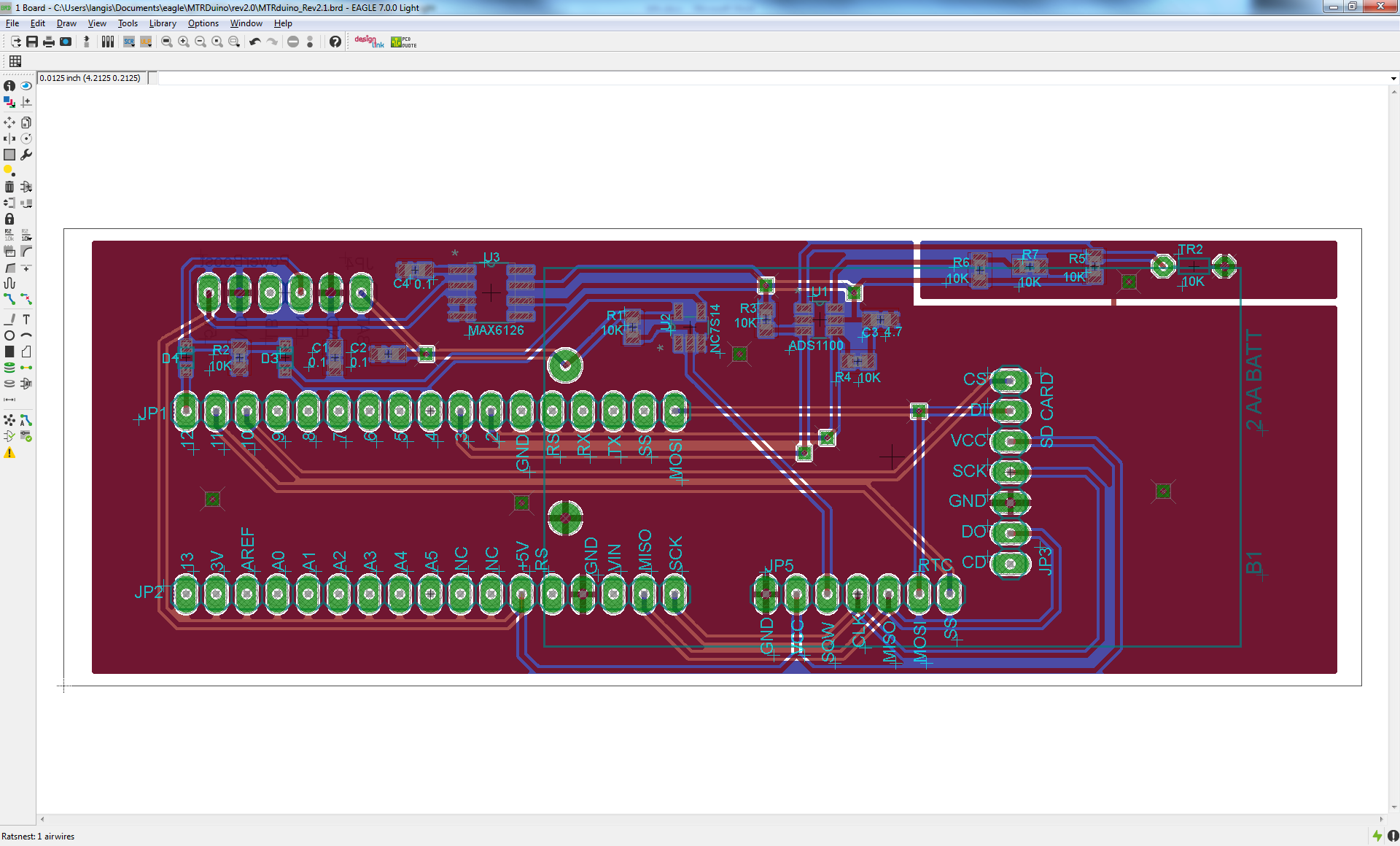


Figure 6: Board Layout (More Precise Version)

## Code 2.0

The following code can be used for the above wiring diagram. It is heavily commented so it should be mostly self-explanatory. Refer to the section on the simpler version for some explanations. The only difference here is we need some additional code to interface with the ADS1100 and take samples. We no longer use the analogRead() or analogReference() functions because we aren’t using the Arduino’s built in ADC.

## Bill of Materials 2.0

Yep. You guessed it. (It’s also included as an excel file so you can work with it).

