**SANDIA REPORT**
SAND2012-10095
Unlimited Release
Printed November 2012

# Digital Attenuator Design

Perry J. Robertson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

Unlimited Release
SAND2012-10095
November 2012

# Digital Attenuator Design

Perry J. Robertson
RF and Opto Microsystems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1082

Abstract

This document describes the design and operation of a electronic digital step attenuator. This attenuator was designed with a 1 dB step resolution and 0.1 dB accuracy with a bandwidth from 1 to 10 MHz. The attenuator dynamic range provides attenuation of input signals from zero to 63 dB in 1 dB steps using a rotary control. A linear array of LEDs is used to display the attenuation value as a 6-bit binary value.

# CONTENTS

# FIGURES

# NOMENCLATURE

| | |
|---|---|
| A | Amperes |
| DOE | Department of Energy |
| SNL | Sandia National Laboratories |
| mV | milli-Volt, 1 V = 1000 mV |
| mA | milli-Amp, 1 A = 1000 mA |
| V | Volts |

# 1. INTRODUCTION

This paper documents the design of a digitally controlled step attenuator capable of operating over a frequency range from 1 to 10 MHz. Initially, a web search of available commercial offerings turned up a number of candidates, however they all either lacked the precision required for the target application or were prohibitively expensive. At the time of the search, commercial step attenuators were running around $5,000 each and there were typically specified to operate at over 1 GHz. Little information was available on their operation at lower frequencies. Additionally, due to the number of units that would be needed (>24) the purchase of such a large number would be prohibitively expensive.

A low cost "hobby grade" binary digital attenuator kit was initially discovered. This attenuator was based on pi attenuator segments connected by DPDT switches. It appeared that was very affordable given the large number of units required. A sample unit was purchased, built and tested. This unit appeared to be suitable and a dozen units were built. An aluminum case was designed to house the attenuator and fabricated. During full scale testing, the switches began to fail and after careful consideration many were found to be faulty. A better solution was needed and the decision was made to design and build our own digital step attenuators.

A digital attenuator chip from Mini-Circuits (DAT-31-PP+) was identified that would be suitable for the overall attenuator. This chip is specified to have a usable frequency range from DC-2400 MHz with an input and output impedance matched to 50 Ohms. However, there was little data from the vendor on its operating characteristics at frequencies below 2 MHz. A prototype unit was needed for further evaluation for its suitability to our application.

A prototype of the digital attenuator was built and tested. The units were built with the ability to program the digital attenuator chip using switches or a microcontroller board. The switches could be set in a binary manner and then a load switch could be used to load the new attenuator value. An Arduino Nano board could also be used to load the attenuator value into the chip. However, there was some risk that the software for the controller would prove difficult to get to work right. In the end, the software for the microcontroller board worked so well, no units utilizing switches were ever built. Overall, around two dozen units were completed.

# 2. STEP ATTENUATOR

When the need for attenuators was first identified, a search of possible candidates was undertaken. Initially, a couple of laboratory grade RF step attenuators were identified (Agilient being one). A set of requirements was generated (see information below). However, the cost to outfit as many as 16 channels with attenuators was cost prohibitive for an LDRD activity. Each attenuator cost approximately $5000. A less expensive alternative was sought.

An attenuator was needed to match return signal levels between channels. The requirements on this design were as follows.

| | |
|---|---|
| Bandwidth | 100 kHz to 10 MHz |
| Battery powered | 9V |
| Attenuation Factor | >60 dB attenuation |
| Power In | > (1Vpp) dBm |
| Controls | Easy to use, must be able to adjust smoothly with single knob |
| Insertion Loss | <3 dB |
| Single Bat. Op. | > 9 hours continuous |

The initial attempt at a simple step attenuator utilized a commercial off the shelf step attenuator sold to hobbyist. This attenuator relied upon multiple pi attenuators[1] which utilized Double Pole, Double Throw (DPDT) switches to increase the attenuation in unit steps. This attenuator had an insertion loss of around 1 dB and could increase attenuation in 1 dB steps to over a range of 80 dB. The unit steps were 20, 20, 20, 10, 5, 3, 2, and 1 dB. These boards were put into a custom designed aluminum housing which had two BNC connectors for signal in and out. The front panel design is shown in Figure 1. The construction of the housing is shown in Figure 2. A 3D view is shown in Figure 3. The overall design drawing is shown in Figure 4.

The initial performance of these attenuators was excellent and a full complement was built. However, with further testing, the switches were found to be faulty in several units. The switches were very inexpensive and it was determined that a better solution was needed if the project were to proceed successfully.

[1] A pi attenuator is constructed from three resistors; a resistor to ground at the input, a resistor to ground at the output and a series resistor between the input and the output. A pi attenuator can be designed to attenuate the incoming signal at any level while maintaining a 50 ohm input impedance and a 50 ohm output impedance.

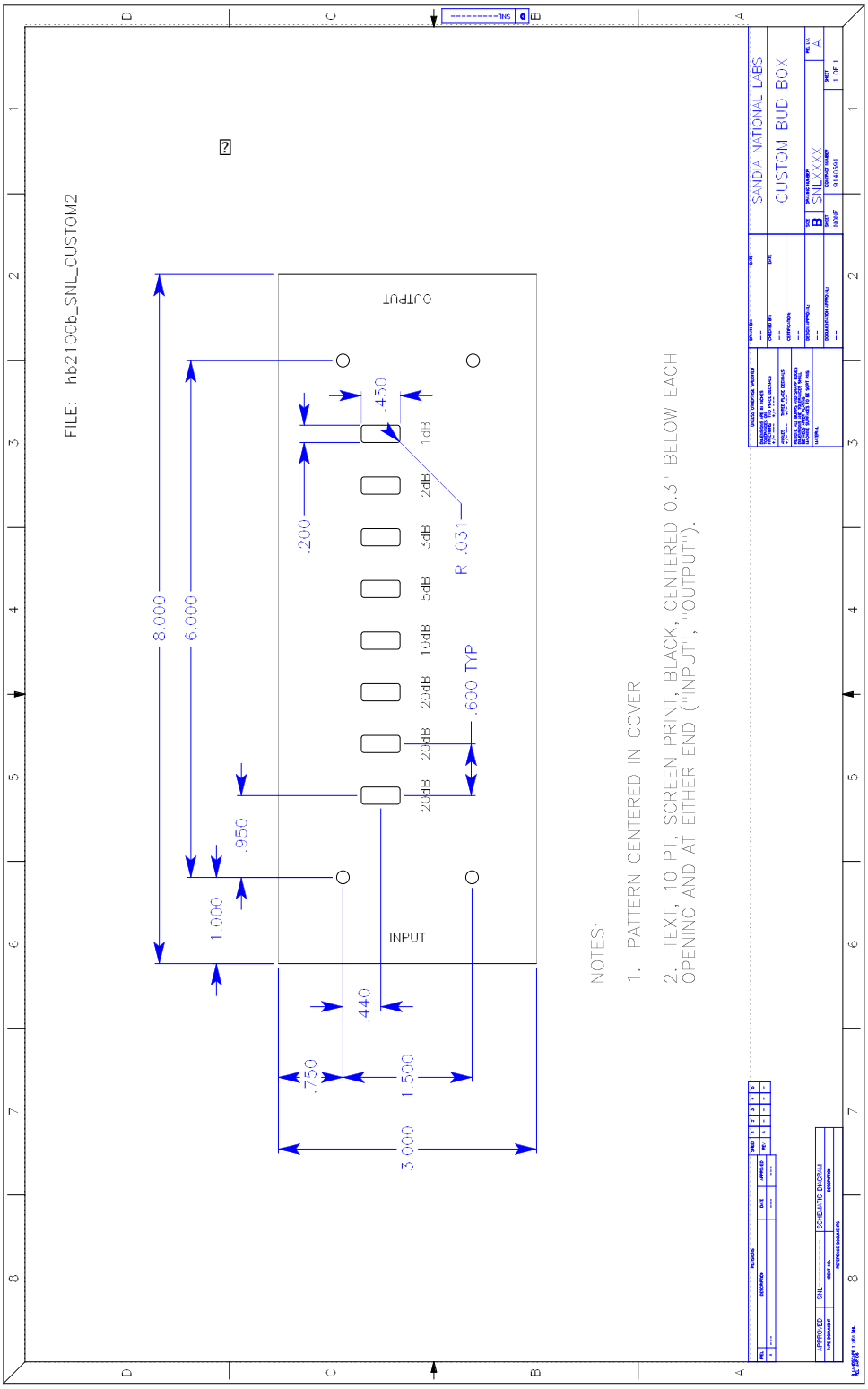



**Figure 1. Front Panel Manual Step Attenuator.**

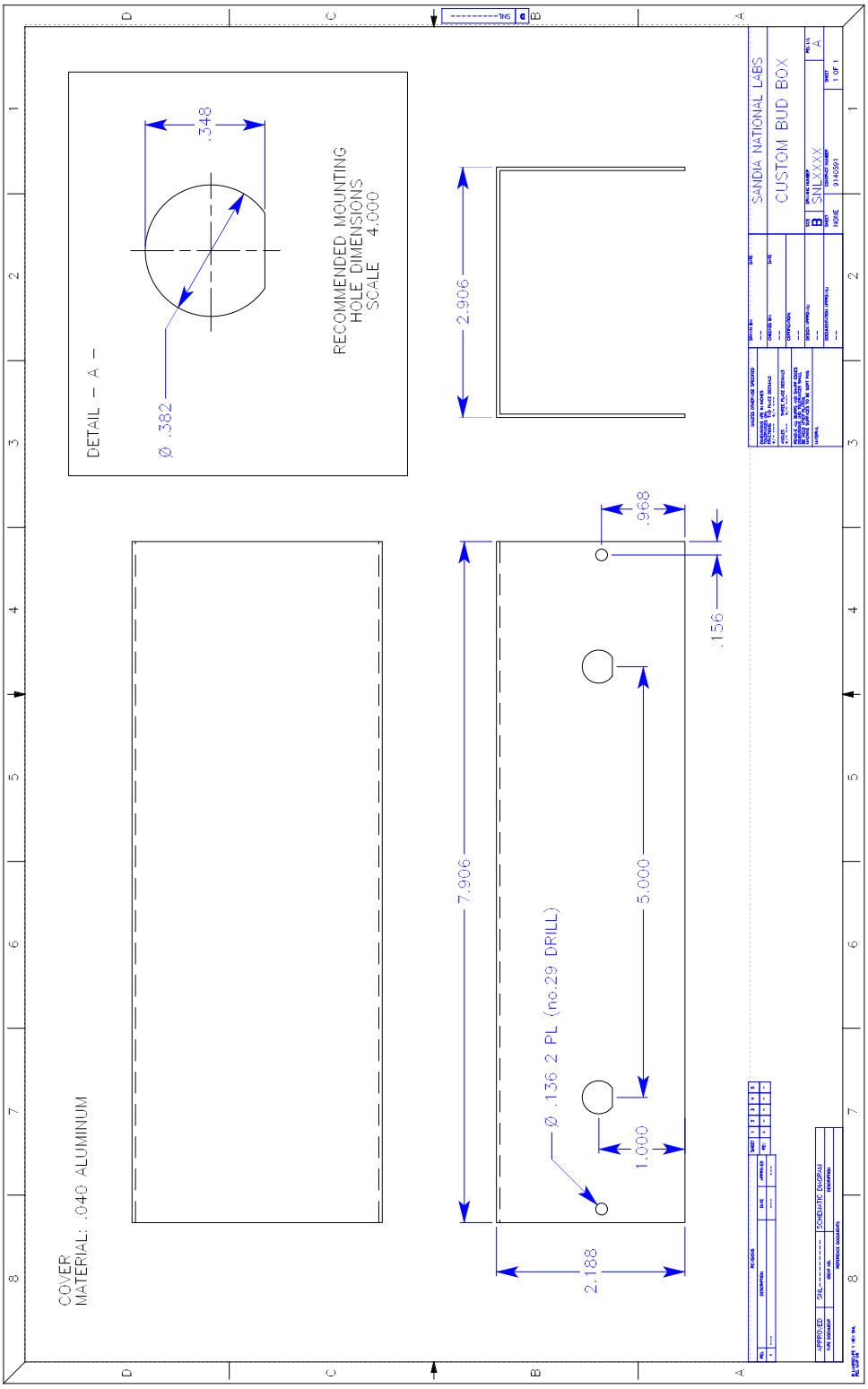



**Figure 2. Step Attenuator Construction.**

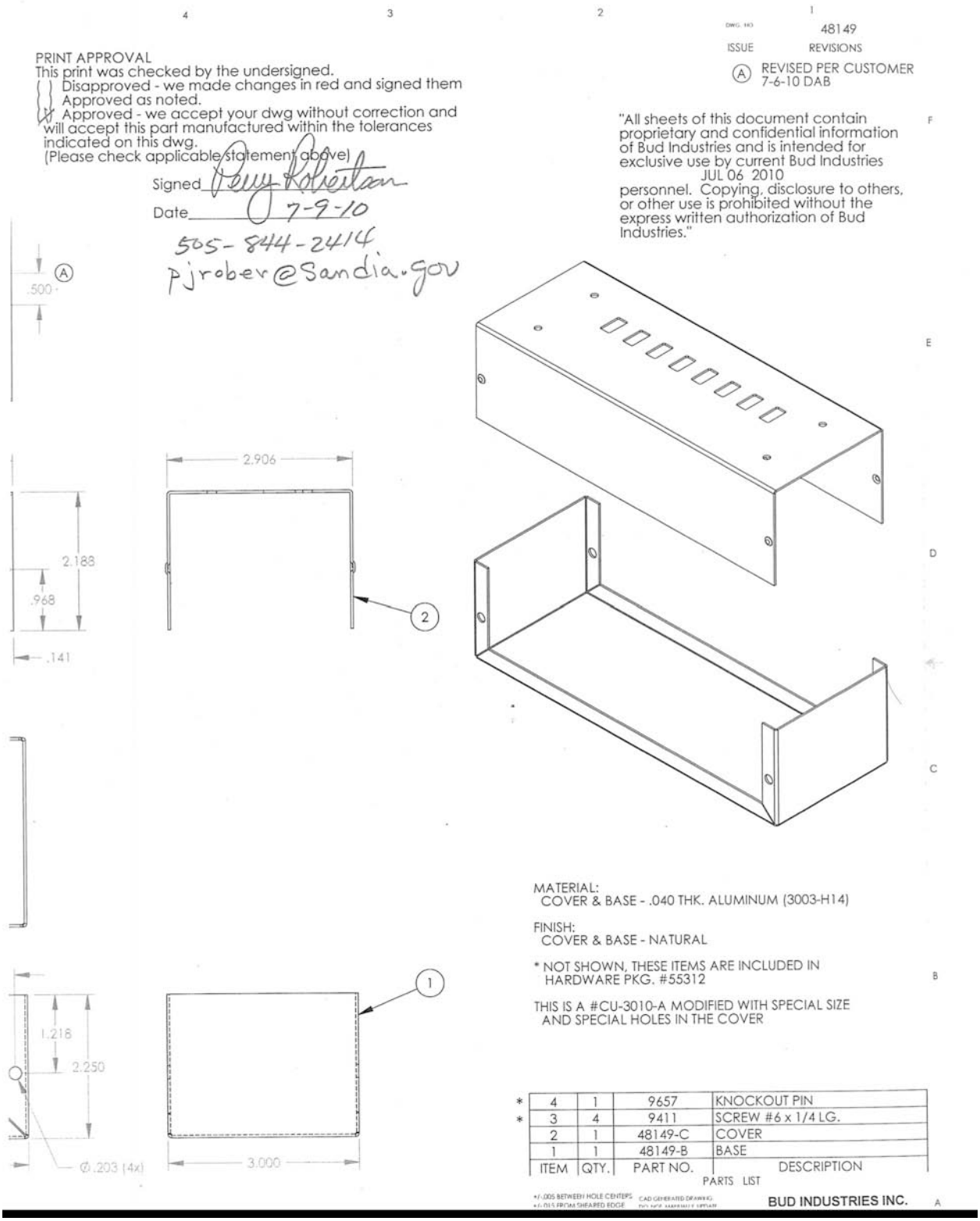



**Figure 3. Step Attenuator 3D View.**

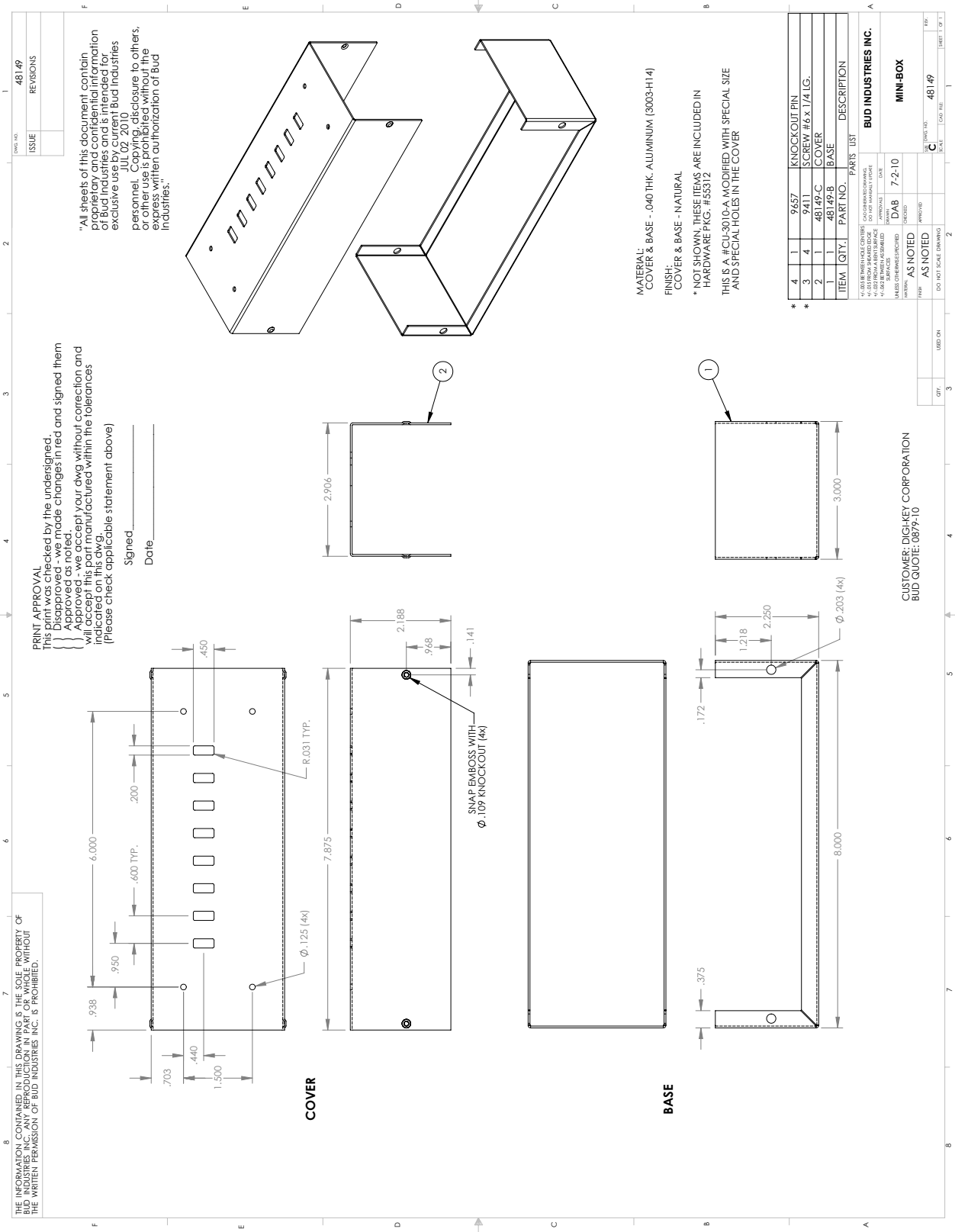



**Figure 4. Step Attenuator Design Drawing.**

# 3. DIGITAL ATTENUATOR CHIP

The electronic digital attenuator chip chosen for this project is a Mini-Circuits DAT-31-PP+. This chip met many of the requirements for the overall step attenuator design. It had electronic digital control via a 5-bit parallel interface that would permit control of the chip using a small microprocessor. Powered from a single +3V power supply requiring only 100 uA of current in the quiescent state this chip could operate for hours from a single 9V battery. It had the accuracy required of 0.1 dB and the ability to step through 31 dB overall attenuation in 1.0 dB steps. The chip was produced in a silicon CMOS process and comes in a 20 pin package.

**Pin Description**

| Function | Pin Number | Description |
|---|---|---|
| C16 | 1 | Control for Attenuation bit, 16dB (Note 3) |
| RF in | 2 | RF in port (Note 1) |
| N/C | 3 | Not connected (Note 4) |
| GND | 4 | Ground connection |
| LE | 5 | Latch Enable Input (Note 2) |
| $V_{DD}$ | 6 | Power Supply |
| PUP1 | 7 | Power-up selection |
| PUP2 | 8 | Power-up selection |
| $V_{DD}$ | 9 | Power Supply |
| GND | 10 | Ground connection |
| GND | 11 | Ground connection |
| GND | 12 | Ground connection (Note 6) |
| GND | 13 | Ground connection |
| RF out | 14 | RF out port (Note 1) |
| C8 | 15 | Control for attenuation bit, 8 dB |
| C4 | 16 | Control for attenuation bit, 4 dB |
| C2 | 17 | Control for attenuation bit, 2 dB |
| GND | 18 | Ground Connection |
| C1 | 19 | Control for attenuation bit, 1 dB |
| N/C | 20 | Not connected (Note 4) |
| GND | Paddle | Paddle ground (Note 5) |

Notes:
1. Both RF ports must be held at 0VDC or DC blocked with an external series capacitor.
2. Latch Enable (LE) has an internal 100KΩ resistor to $V_{DD}$.
3. Place a 10KΩ resistor in series, as close to pin as possible to avoid freq. resonance.
4. Place a shunt 10KΩ resistor to GND.
5. The exposed solder pad on the bottom of the package (See Pin Configuration) must be grounded for proper device operation.
6. Ground must be less than 80 mil (0.08") from Pin 12 for proper device operation.

**Pin Configuration (Top View)**

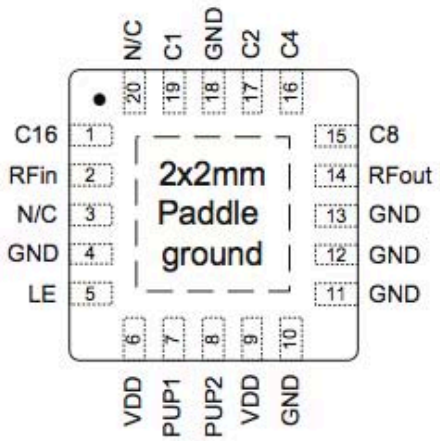



**Figure 5. Digital Attenuator Chip Pinout (from Mini-Circuits DAT-31-PP+ Datasheet).**

# 4. ATTENUATOR DESIGN

The attenuator was designed to fit into a Pomona Aluminum die cast box model 3602 with the four layer FR4 board acting as the top as shown in Figure 6. The 9V battery was mounted on the outside to make it easy to change the battery when required (see Figure 7). Two BNC connectors can be seen in the top view (Figure 8) and are the input and output connectors.

The digital attenuator chips are mounted on the back of the printed circuit board as shown in Figure 9. Two attenuator chips are used in series in order to achieve an overall attenuator factor of > 60 dB. The Arduino controller board can also be seen on this side of the board along with a host of surface mount components. The controller monitors the position of the rotational control (measuring a voltage) and sets the attenuation on the two chips accordingly. The controller also monitors when the knob is not being manipulated and goes into a sleep mode to save power. The final units were demonstrated to operate from a fresh 9V battery for at least an entire 9 hour day. A reference to the 9V battery lifetime is given in Appendix C.

There are three controls on the attenuator. The switch at the upper left is the on-off switch. The switch at the upper right latches data into the parallel inputs of the attenuator chips. This switch is utilized only if the board is being used in manual mode. Normally, the switch is left in the Enable position. The rotating knob in the center of the unit controls the overall attenuation.

LEDs at the bottom of the board indicate the attenuation setting as a binary number with weightings of 1, 2, 4, 8, 16, 31 as shown in Figure 6. The high bit is not 32 as one would expect because it represents one of the attenuator chips with all inputs high (the maximum setting for a single attenuator chip is 31 dB.

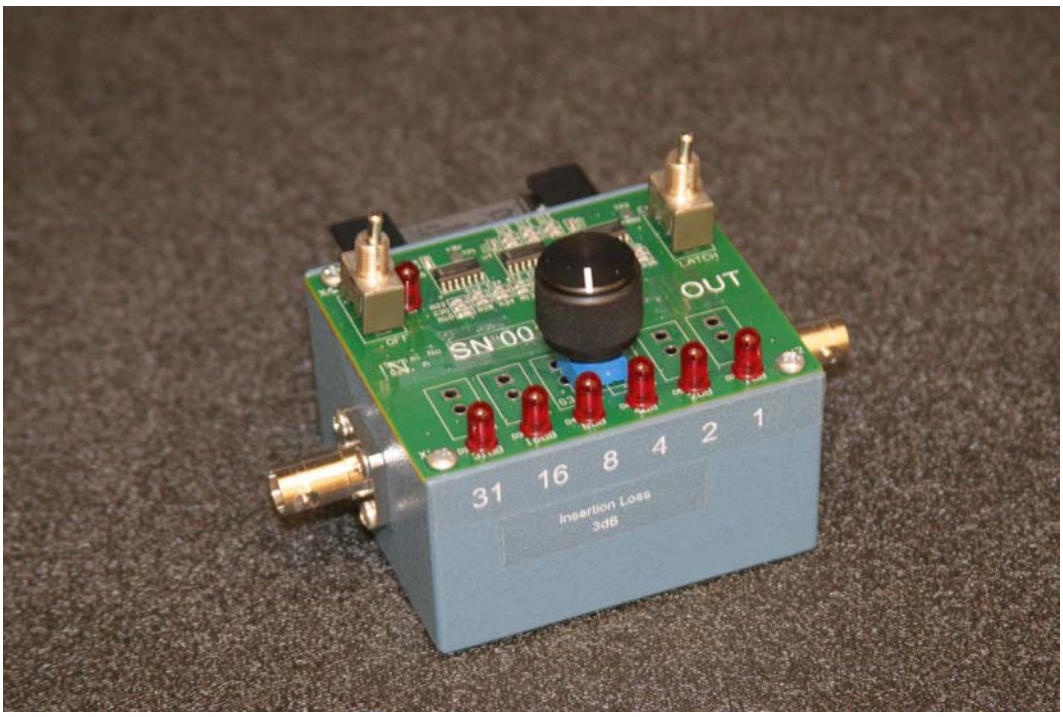



**Figure 6. Digital Attenuator.**

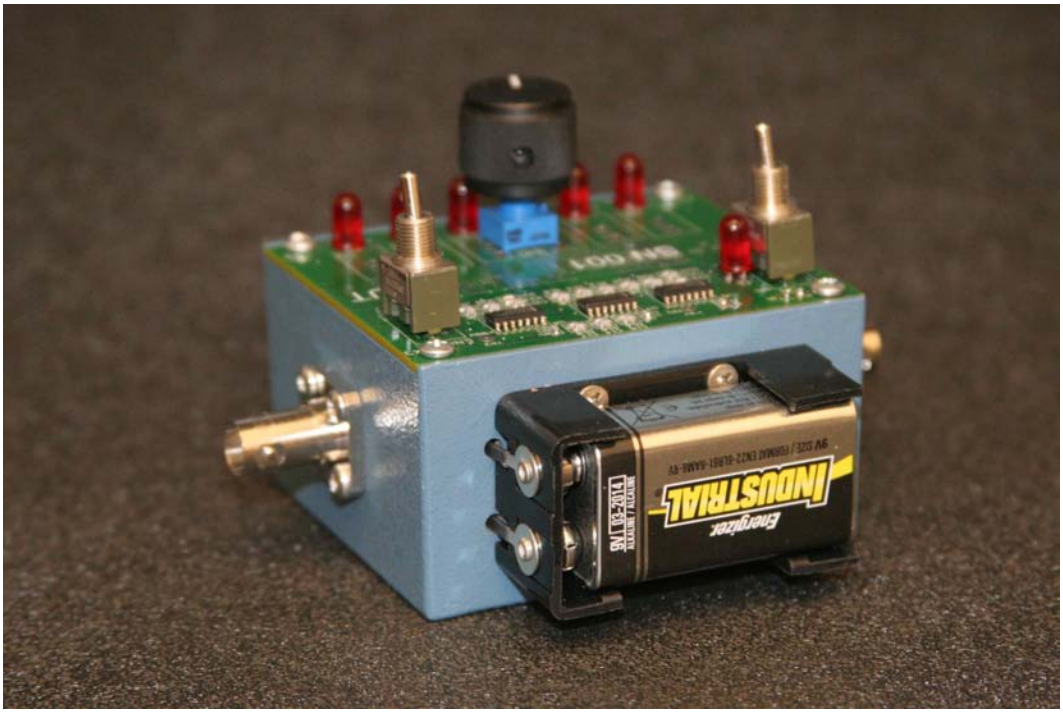



**Figure 7. Digital Attenuator Side View.**

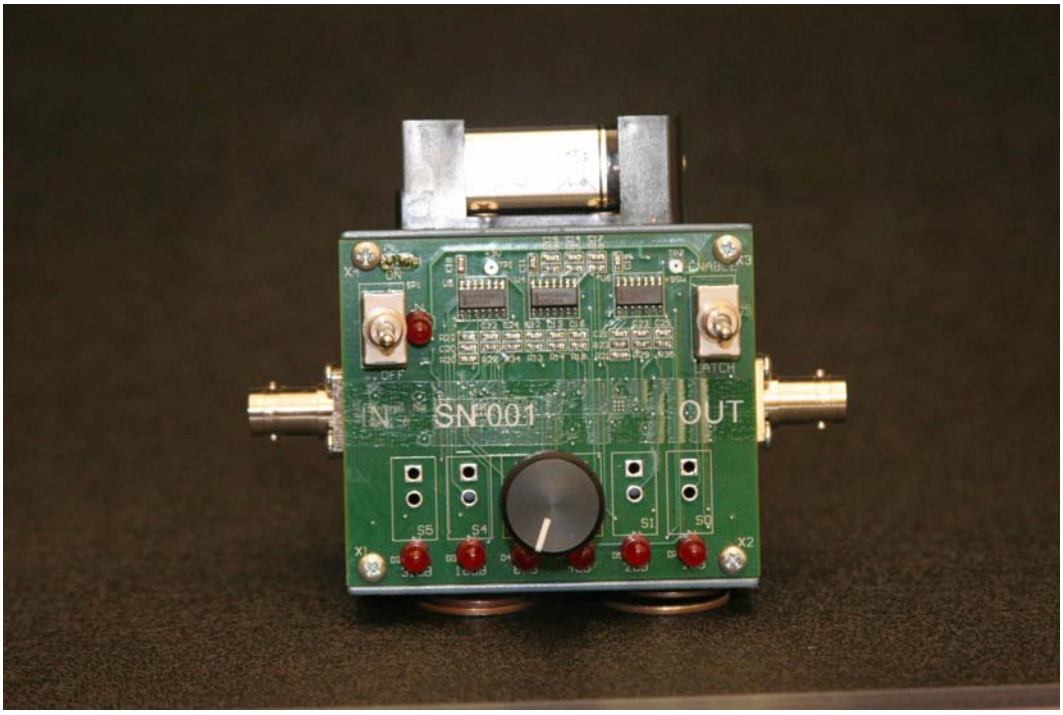



**Figure 8. Digital Attenuator Top Down View.**

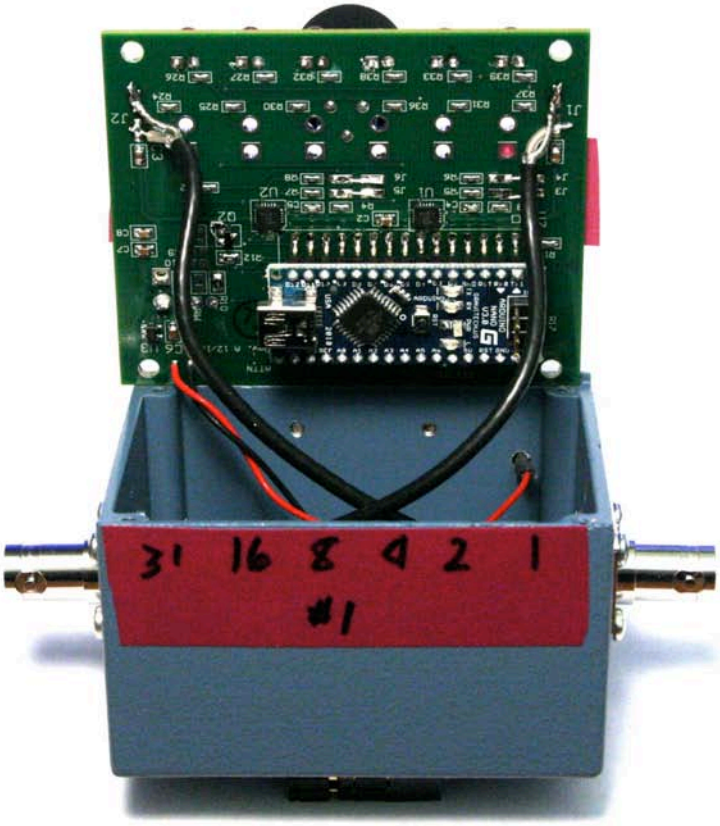



**Figure 9. Digital Attenuator Controller Installed.**

# 5. ATTENUATOR PERFORMANCE

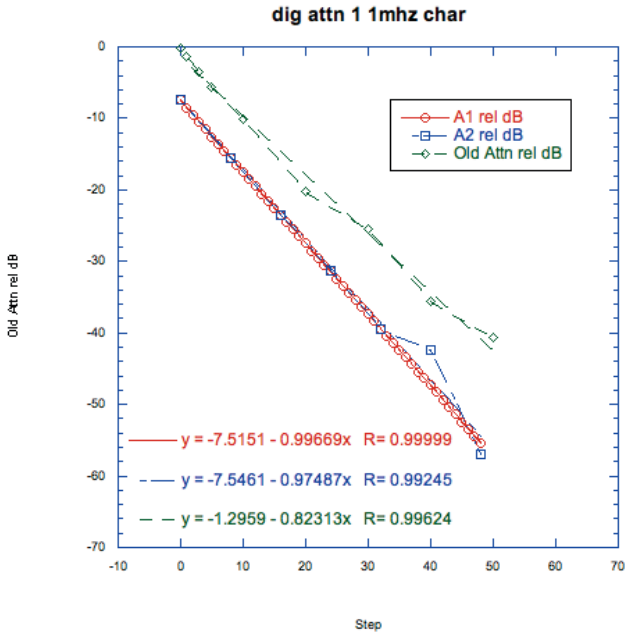



**Figure 10. Attenuator Performance.**

# 6. CONTROLLER BOARD

The controller board chosen for this project is the Arduino Nano 3.0 (contains a single ATmega328 microprocessor). This board has an extremely small footprint and has all the support components to quickly and efficiently start a design using it as the controller. It can be powered via the Mini-USB connector or from your motherboard. This board also contains a voltage regulator so that the power can vary from 7V to over 12V. It has 14 digital I/O pins (of which 6 can be programmed to provide PWM output). The board is 0.73" x 1.70". Complete specifications can be found at http://arduino.cc/en/Main/AuduinoBoardNano.

The Arduino Nano Board contains a USB (serial) port that is used to load the board and can be used for debugging as a serial interface to the board.

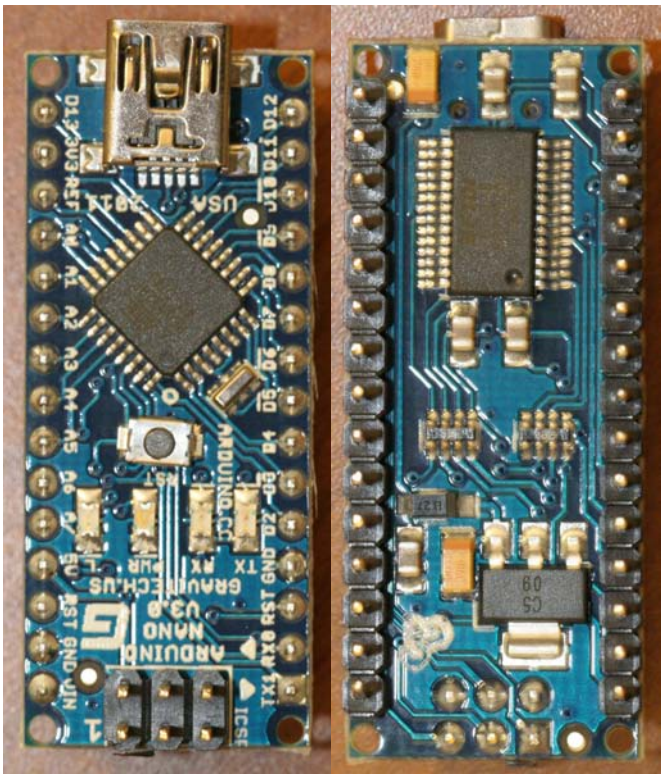


**Figure 11. Arduino Nano Development Board Top and Bottom Views.**

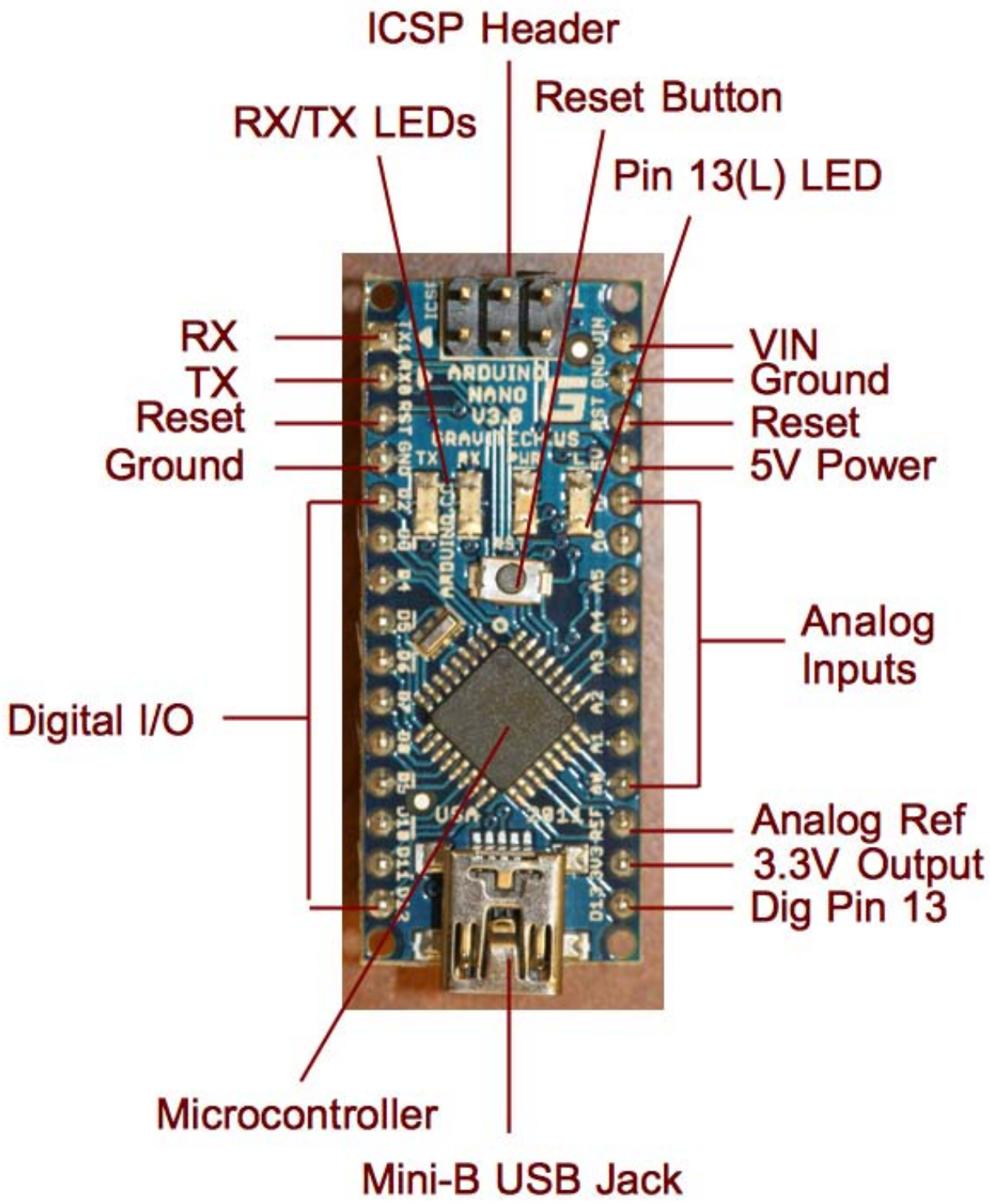



**Figure 12. Arduino Nano 3.0 with part descriptions (from ardunio.cc).**

# 7. PRINTED CIRCUIT BOARD DESIGN

The schematic of the printed circuit board is given in the Appendix A.

Page 1 of the schematic shows the 9V battery going to a toggle switch, SP1, which applies power to the rest of the board. There is a multi-vibrator circuit consisting of NPN transistors Q1 and Q2 that was designed to blink the power LED, LED0. This design was intended for stand-alone operation, i.e. not having the controller board. It was not known if the controller chip would be able to work, so the board was designed with a manual capability. The LED must blink or the power in the on state will quickly kill the battery. An assembly note details how to configure the board for Arduino control. Power is applied to the +3V regulator U3 which generates power for the rest of the board.

Page 2 of the schematic shows the control switches (which were not needed in the end) and Schmitt trigger inverting buffers, U5 and U6, that are used to drive the digital attenuator chip's digital inputs. 10K Ohm pull-up resistors are attached to each of the inputs. The Arduino controller drives each of the inputs producing one signal which lights the LED and a second signal that goes to the attenuator chip.

Page 3 of the schematic shows the high control bits that are branched out and drive the second attenuator chip (high bit is worth 31 dB) and the chip enables, LE0 and LE1. As can be seen, both chips are enabled at the same time.

Page 4 of the schematic shows the connections to the attenuator chips. Note that the chips are isolated from the input and output and each other via 2.2 uF capacitors.

Page 5 of the schematic shows the connections to the Arduino controller board. The Arduino controller board has its own internal +5V regulator and is therefore connected directly to the +9V switched power. R40 is a single turn 100K POT that generates a voltage between 0 and +5 volts. R41 and R41 create a voltage divider that allows the Arduino controller to monitor the battery voltage and signal the user via power LED flashes that the battery is nearing end of useful life.

# 8. SOFTWARE OPERATION

This section describes the operation of the Arduino Controller. The software is listed in Appendix B. The software contains two procedures: setup() and loop(). Setup is executed first and is responsible for initializing all the digital pins. Setup also starts the internal timers. After execution, Setup passes control to Loop. Loop consists of a single unending loop.

Setup initializes the digital I/O to a known startup state. It then turns on the power LED, sets the AREF output to 5V, starts the timer and captures the current time. Control then passes to Loop().

The first time through the Loop (the entirety of loop only really gets executed once) the power LED is used to signal what version of the software is loaded via a series of coded flashes. This is performed by the routine showversion(). Next the initial read is made of the battery state using the procedure blinkPwrLed(). This procedure blinks the LED at least once, given that the battery is fresh. As the battery drains, the number of flashes increases, hopefully getting the user's attention to change the battery. After flashing, the LED is turned off to save power.

The initial reading of the input POT is performed by the readPot() routine. This value is put into the variable newValue, which is used to compare the current and last settings of the POT. If there is a change, then the new POT setting is used to compute a new digital word for the attenuators and output a new setting which is done by the routine changeAttn(). The first time through, the newValue is just loaded into the attenuators without checking if there was a change.

The current time (currTime) is found and a future time (deltaTime) is calculated. When this time is exceeded, the LED is blinked 10 times, using the routine sleepBlink(), and the controller is put into a low power mode using the routine sleepNow(). The sleep mode is set to SLEEP_MODE_PWR_DOWN which has the most power savings and the unit enters sleep_mode(). An interrupt will occur which will wake the unit and execution continues with another time through the loop(). Assuming no changes have been detected in the POT setting, the controller goes back to sleep.

The unit will go in this power savings mode after a short time period where the user has

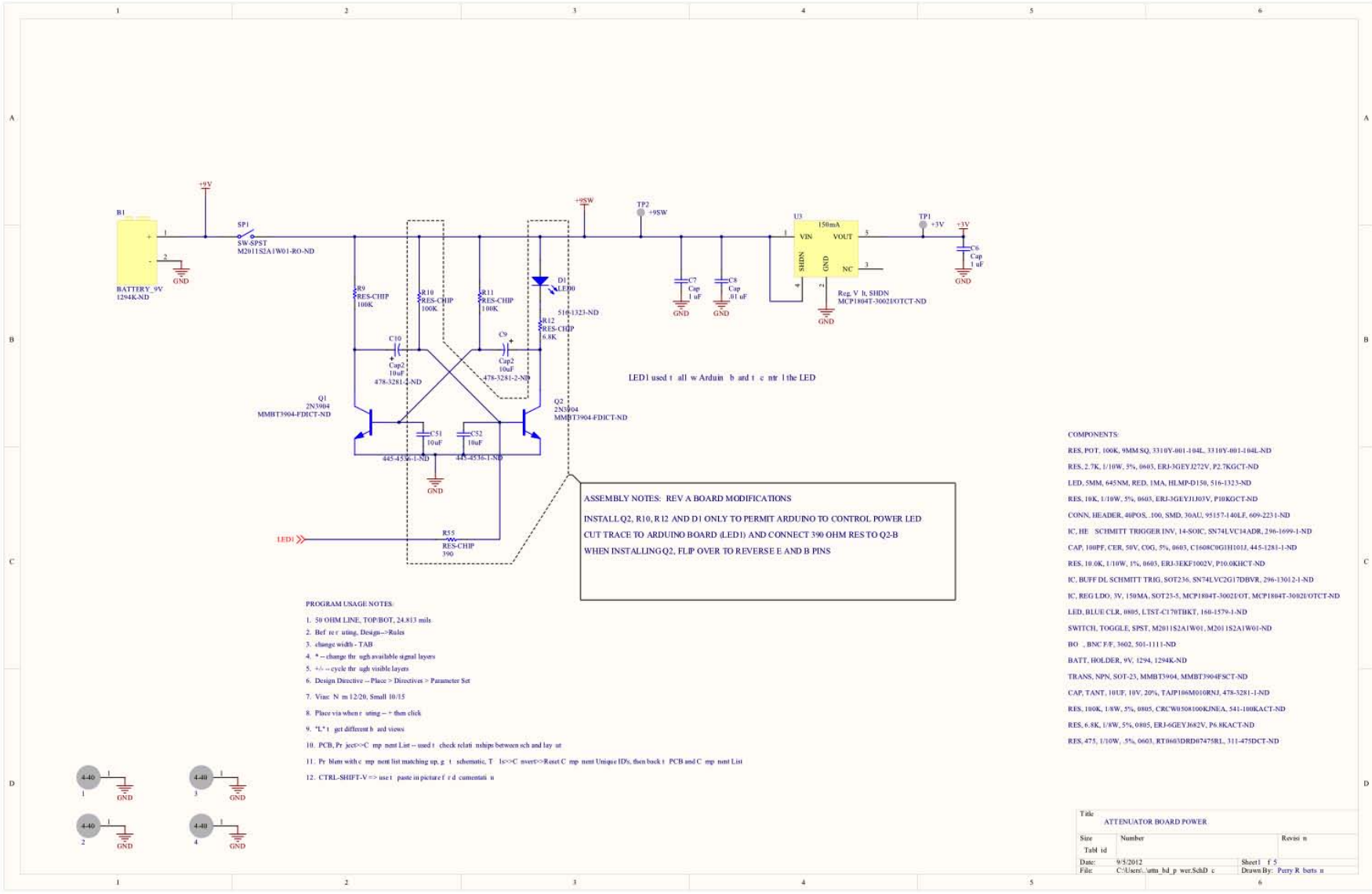
quit making changes to the attenuator setting. The user experiences a short delay when they again start to adjust the POT. The timing was adjusted to make the user experience as seamless as possible.

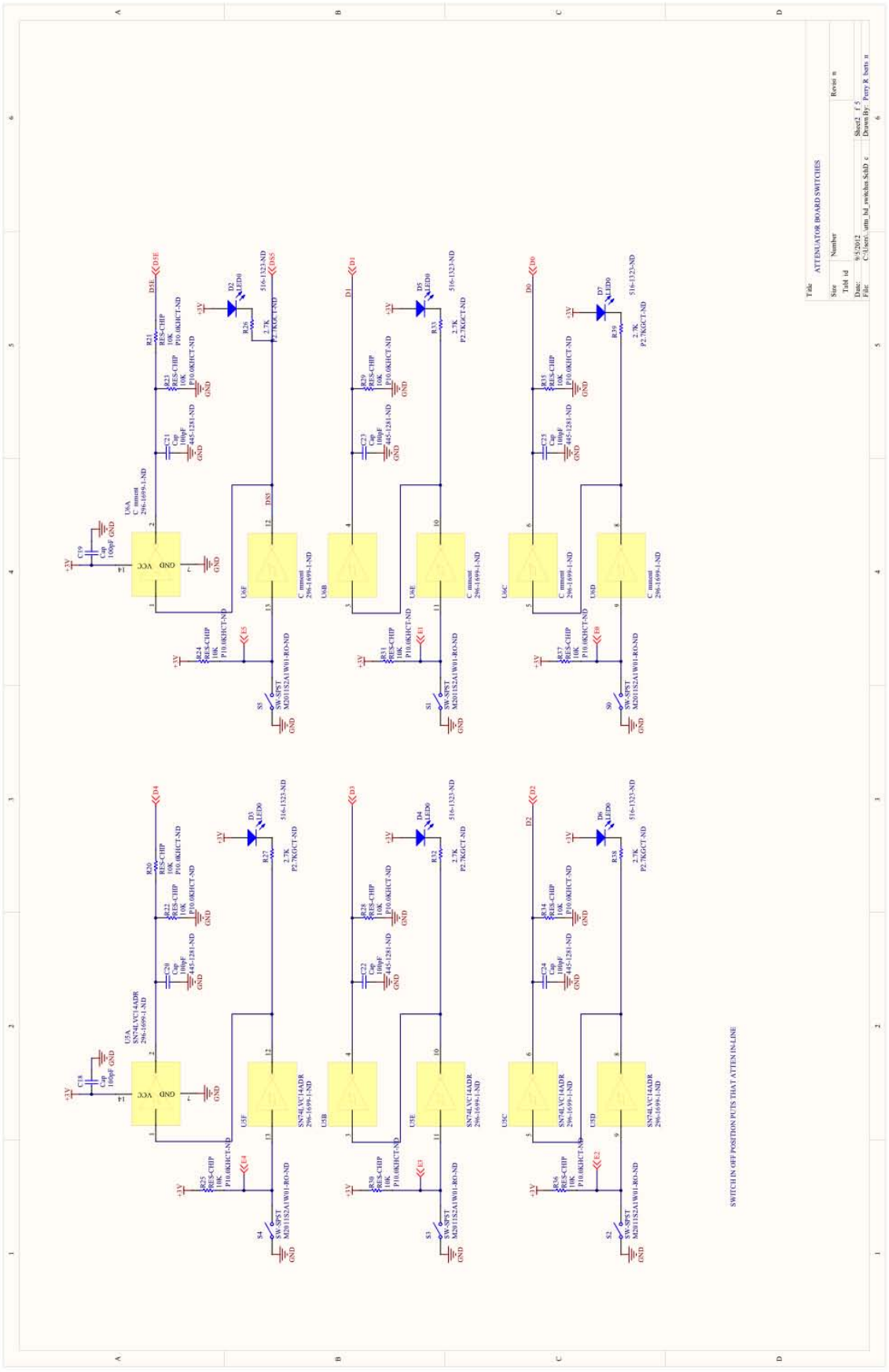
# 9. CONCLUSIONS

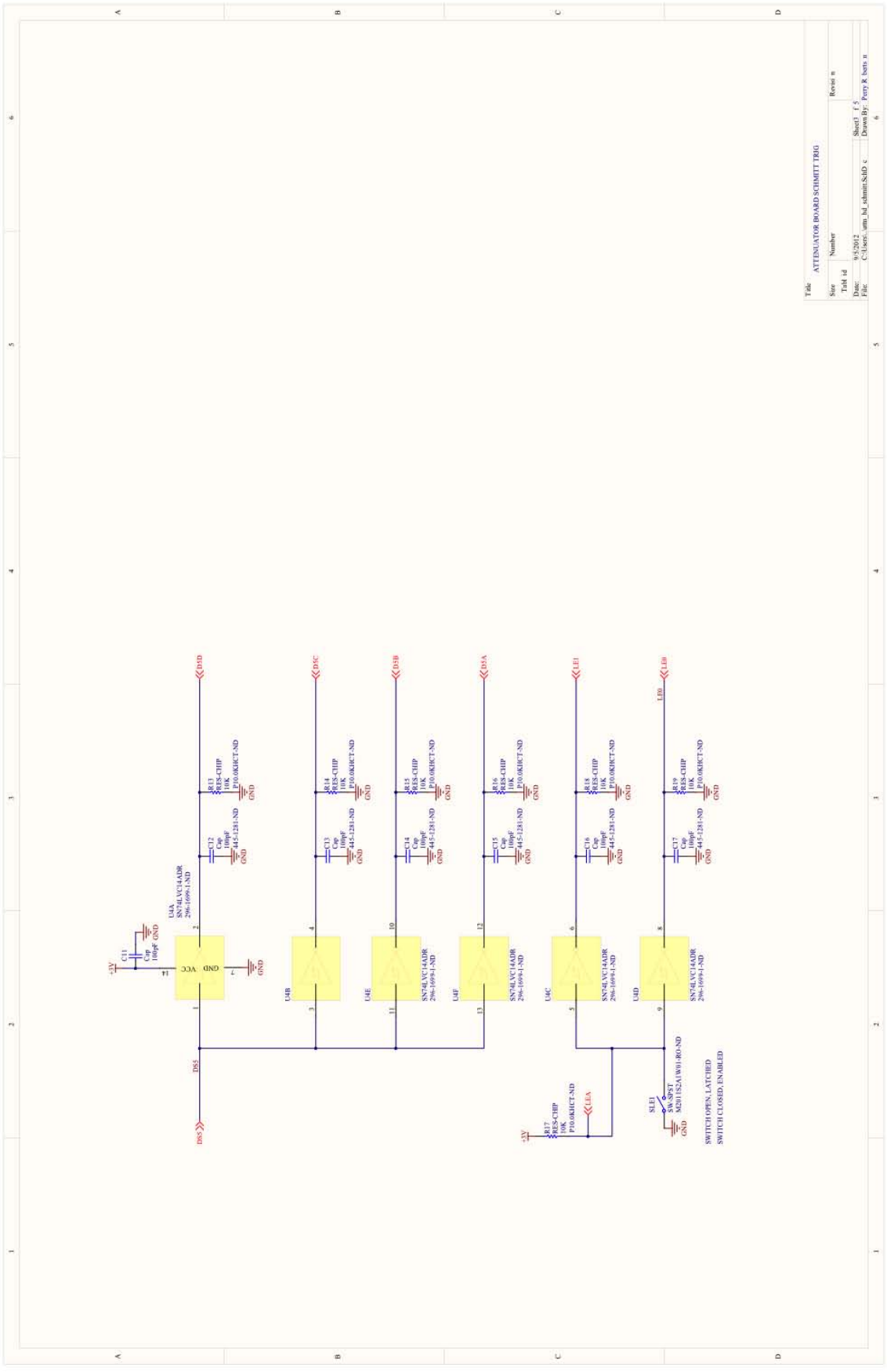
This document describes the design and operation of a digital step attenuator. This attenuator was designed to aid in the measurement of signals from 1 to 10 MHz. The attenuator was capable of attenuating the input signal from zero to 63 dB in 1 dB steps using a rotary control. A binary linear array of LEDs is used to display the attenuation value. The control of the attenuator chip was accomplished using an Arudino micro controller board. The software permitted the user to rotate a dial to change the attenuator value.

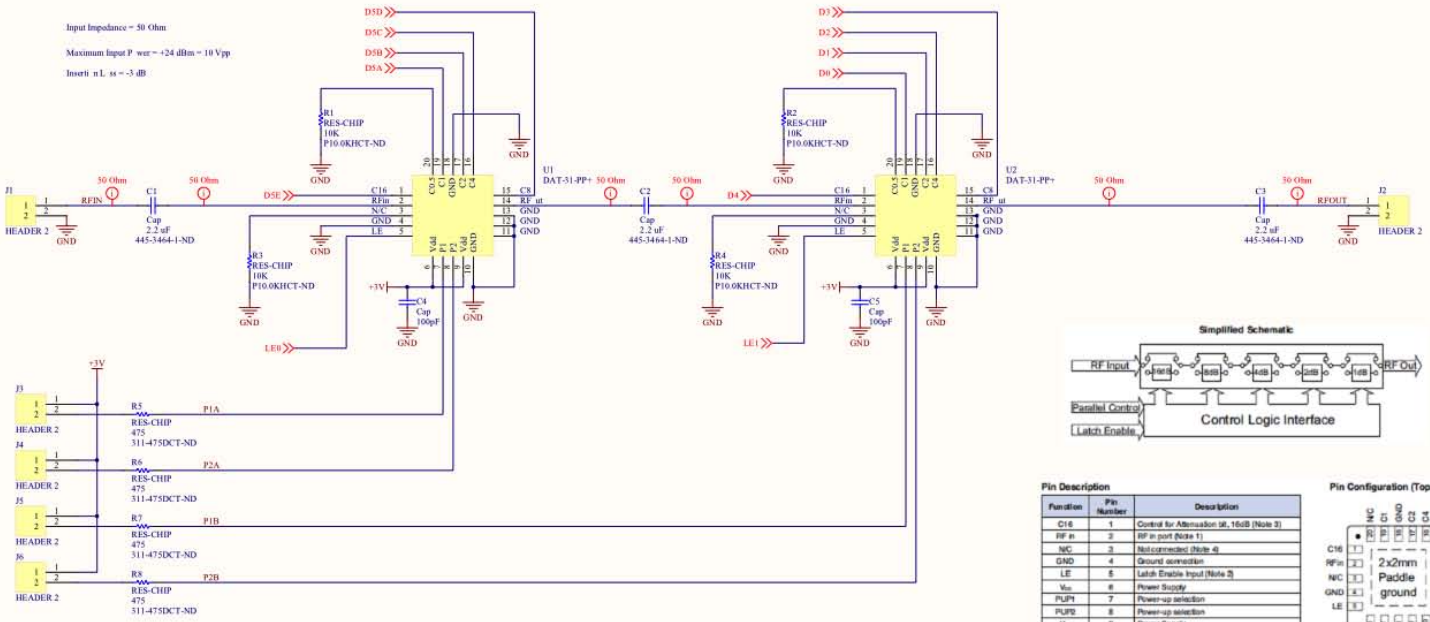
# APPENDIX A

**Schematics and Bill of Materials**


B1
BATTERY_9V
1294K-ND
+9V
SP1
SW-SPST
M2011S2A1W01-RO-ND
GND
R9
RES-CHIP
100K
R10
RES-CHIP
100K
R11
RES-CHIP
100K
D1
LED0
516-1323-ND
R12
RES-CHIP
6.8K
+9SW
TP2
+9SW
C7
Cap
1 uF
C8
Cap
.01 uF
U3
150mA
VIN
VOUT
SHDN
GND
NC
Reg. V lt, SHDN
MCP1804T-3002I/OTCT-ND
TP1
+3V
+3V
C6
Cap
1 uF
C10
Cap2
10uF
478-3281-2-ND
C9
Cap2
10uF
478-3281-2-ND
Q1
2N3904
MMBT3904-FDICT-ND
Q2
2N3904
MMBT3904-FDICT-ND
C51
10uF
445-4536-1-ND
C52
10uF
445-4536-1-ND
LED1 used t all w Arduin b ard t c ntr l the LED
LED1
R55
RES-CHIP
390
ASSEMBLY NOTES: REV A BOARD MODIFICATIONS
INSTALL Q2, R10, R12 AND D1 ONLY TO PERMIT ARDUINO TO CONTROL POWER LED
CUT TRACE TO ARDUINO BOARD (LED1) AND CONNECT 390 OHM RES TO Q2-B
WHEN INSTALLING Q2, FLIP OVER TO REVERSE E AND B PINS
PROGRAM USAGE NOTES:
1. 50 OHM LINE, TOP/BOT, 24.813 mils
2. Bef re uting, Design-->Rules
3. chinge width - TAB
4. * -- chinge thr ugh available signal layers
5. +/- -- cycle thr ugh visible layers
6. Design Directive -- Place > Directives > Parameter Set
7. Vias: N m 12/20, Small 10/15
8. Place via when r uting -- + then click
9. "L" t get different h ard views
10. PCB, Pr ject<>C mp nent List -- used t check relati nships between sch and lay ut
11. Pr blem with c mp nent list matching up, g t schematic, T ls<>C nvert<>Reset C mp nent Unique IDs, then back t PCB and C mp nent List
12. CTRL-SHIFT-V => use t paste in picture f r d cumentati n
4-40
GND
COMPONENTS:
RES, POT, 100K, 9MM SQ, 3310Y-001-104L, 3310Y-001-104L-ND
RES, 2.7K, 1/10W, 5%, 0603, ERJ-3GEYJ272V, P2.7KGCT-ND
LED, 5MM, 645NM, RED, 1MA, HLMP-D150, 516-1323-ND
RES, 10K, 1/10W, 5%, 0603, ERJ-3GEYJ103V, P10KGCT-ND
CONN, HEADER, 40POS, .100, SMD, 30AU, 95157-140LF, 609-2231-ND
IC, HE SCHMITT TRIGGER INV, 14-SOIC, SN74LVC14ADR, 296-1699-1-ND
CAP, 100PF, CER, 50V, C0G, 5%, 0603, C1608C0G1H101J, 445-1281-1-ND
RES, 10.0K, 1/10W, 1%, 0603, ERJ-3EKF1002V, P10.0KHCT-ND
IC, BUFF DL SCHMITT TRIG, SOT236, SN74LVC2G17DBVR, 296-13012-1-ND
IC, REG LDO, 3V, 150MA, SOT23-5, MCP1804T-3002I/OT, MCP1804T-3002I/OTCT-ND
LED, BLUE CLR, 0805, LTST-C170TBKT, 160-1579-1-ND
SWITCH, TOGGLE, SPST, M2011S2A1W01, M2011S2A1W01-ND
BO , BNC F/F, 3602, 501-1111-ND
BATT, HOLDER, 9V, 1294, 1294K-ND
TRANS, NPN, SOT-23, MMBT3904, MMBT3904FSCT-ND
CAP, TANT, 10UF, 10V, 20%, TAJP106M010RNJ, 478-3281-1-ND
RES, 100K, 1/8W, 5%, 0805, CRCW0508100KJNEA, 541-100KACT-ND
RES, 6.8K, 1/8W, 5%, 0805, ERJ-6GEYJ682V, P6.8KACT-ND
RES, 475, 1/10W, .5%, 0603, RT0603DRD07475RL, 311-475DCT-ND
Title
ATTENUATOR BOARD POWER
Size
Number
Revisi n
Tabl id
Date: 9/5/2012
Sheet1 f 5
File: C:\Users\...uttn_bd_p wer.SchD c
Drawn By: Parry R barts n


ATTENUATOR BOARD SWITCHES
SWITCH IN OFF POSITION PUTS THAT ATTEN IN-LINE


ATTENUATOR BOARD SCHMITT TRIG
Number
Revision
Sheet1 of 5
Drawn By: Perry R. Isern II
C:\Users\...utm_bd_schmitt.SchDoc
9/5/2012
SWITCH OPEN, LATCHED
SWITCH CLOSED, ENABLED
SN74LVC14ADR
296-16991-ND
RES-CHIP
10K
P10.0KHCT-ND
Cap
100pF
445-1281-ND
SW-SPST
M2015S2A1W01-R0-ND
GND
+3V





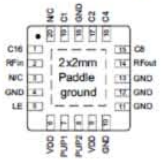


**Pin Description**

| Function | Pin Number | Description |
|---|---|---|
| C16 | 1 | Control for Attenuation bit, 16dB (Note 3) |
| RF in | 2 | RF in port (Note 1) |
| N/C | 3 | Not connected (Note 4) |
| GND | 4 | Ground connection |
| LE | 5 | Latch Enable Input (Note 2) |
| $V_{DD}$ | 6 | Power Supply |
| PUP1 | 7 | Power-up selection |
| PUP2 | 8 | Power-up selection |
| $V_{DD}$ | 9 | Power Supply |
| GND | 10 | Ground connection |
| GND | 11 | Ground connection |
| GND | 12 | Ground connection |
| GND | 13 | Ground connection |
| RF out | 14 | RF out port (Note 1) |
| C8 | 15 | Control for attenuation bit, 8 dB |
| C4 | 16 | Control for attenuation bit, 4 dB |
| C2 | 17 | Control for attenuation bit, 2 dB |
| GND | 18 | Ground Connection |
| C1 | 19 | Control for attenuation bit, 1 dB |
| N/C | 20 | Not connected (Note 4) |
| GND | Paddle | Paddle ground (Note 5) |

Notes:
1. Both RF ports must be held at 0VDC or DC blocked with an external series capacitor.
2. Latch Enable (LE) has an internal 100KΩ resistor to $V_{DD}$.
3. Place a 10KΩ resistor in series, as close to pin as possible to avoid freq. resonance.
4. Place a shunt 10KΩ resistor to GND.
5. The exposed solder pad on the bottom of the package (See Pin Configuration) must be grounded for proper device operation.

**Pin Configuration (Top View)**




**Power-up Control Settings**

The DAT-31-PP always assumes a specifiable attenuation setting on power-up, allowing a known attenuation state to be established before an initial parallel control word is provided.

When the attenuator powers up with LE=0, the control bits are automatically set to one of four possible values . These four values are selected by the two power-up control bits,PUP1 and PUP2 ,as shown in table 3: (Power-Up Truth Table, Parallel Mode).

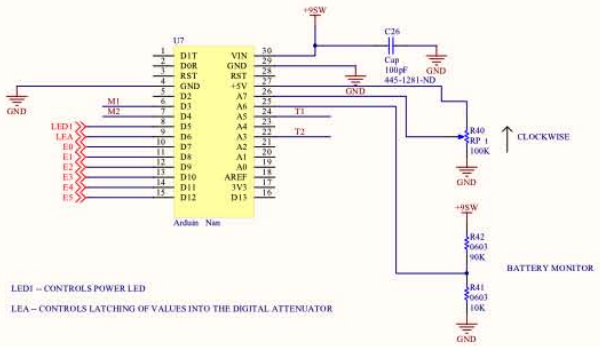
**Table 3. Power-Up Truth Table, Parallel Mode**

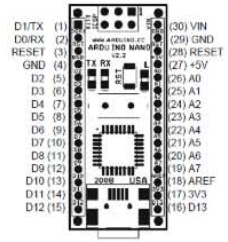
| Attenuation State | PUP1 | PUP2 | LE |
|---|---|---|---|
| Reference | 0 | 0 | 0 |
| 8 (dB) | 0 | 1 | 0 |
| 16 (dB) | 1 | 0 | 0 |
| 31 (dB) | 1 | 1 | 0 |
| Defined by C1-C16 (See Table 1-Truth Table) | X (Note 1) | X (Note 1) | 1 |

Note 1: PUP1 and PUP2 Connection may be 0, 1, GROUND, or not connect, without effect on attenuation state.

Power-Up with LE=1 provides normal parallel operation with C1-C16, and PUP1 and PUP2 are not active.

| | | | |
|---|---|---|---|
| Title | ATTENUATOR BOARD MC ATTENUATORS | | |
| Size | Number | | Revisi n |
| Tabl id | | | |
| Date: | 9/5/2012 | Sheet4 f 5 | |
| File: | C:\Users\...uttn_bd_attnuat rs.SchD c | Drawn By: | Perry R barts n |




| Pin No. | Name | Type | Description |
|---|---|---|---|
| 1-2, 5-16 | D0-D13 | I/O | Digital input/output port 0 to 13 |
| 3, 28 | RESET | Input | Reset (active low) |
| 4, 29 | GND | PWR | Supply ground |
| 17 | 3V3 | Output | +3.3V output (from FTDI) |
| 18 | AREF | Input | ADC reference |
| 19-26 | A7-A0 | Input | Analog input channel 0 to 7 |
| 27 | +5V | Output or Input | +5V output (from on-board regulator) or +5V (input from external power supply) |
| 30 | VIN | PWR | Supply voltage |

| | | | |
|---|---|---|---|
| Title | ATTENUATOR BOARD ARDUINO | | |
| Size | Number | | Revisi n |
| Tabl id | | | |
| Date: | 9/5/2012 | Sheet5 f 5 | |
| File: | C:\Users\...\attn_bd_arduin .SchD c | Drawn By: | Perry R barts n |

| Comment | Description | Designator | Footprint | LibRef | Quantity |
|---|---|---|---|---|---|
| BATTERY_9V | Battery 9V | B1 | HDR2 | BATTERY_9V | 1 |
| Cap | Capacitor, non-polar | C1, C2, C3, C6, C7, C8 | RC0805 | Cap | 6 |
| Cap | Capacitor, non-polar | C4, C5, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23, C24, C25, C26 | RC0603 | Cap | 18 |
| Cap2 | Capacitor, polar | C9, C10 | C0805 | Cap2 | 2 |
| Cap | Capacitor, non-polar | C51, C52 | RC1210 | Cap | 2 |
| LED0 | Typical INFRARED GaAs LED | D1, D2, D3, D4, D5, D6, D7 | LED-.1 | LED0 | 7 |
| HEADER 2 | Header, .2" | J1, J2, J3, J4, J5, J6 | HEADER 2 SMD | HEADER 2 | 6 |
| 2N3904 | 2N3904, SMD | Q1, Q2 | SOT23-3 | 2N3904 | 2 |
| RES-CHIP | CarbResistor, Chip | R1, R2, R3, R4, R5, R6, R7, R8, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R34, R35, R36, R37, R38, R39, R55 | RC0603 | RES-CHIP | 36 |
| RES-CHIP | CarbResistor, Chip | R9, R10, R11, R12 | RC0805 | RES-CHIP | 4 |
| RPot | Potentiometer | R40 | BOURNS-3310Y | RPot | 1 |
| 0603 | CarbResistor, Chip | R41, R42 | RC0603 | RES-CHIP | 2 |
| SW-SPST | Single-Pole, Single-Throw Switch | S0, S1, S2, S3, S4, S5, SLE1, SP1 | M2011S2 | SW-SPST | 8 |
| +3V | Test Point Thru Hole | TP1 | TP-THRU | TestPoint | 1 |
| +9SW | Test Point Thru Hole | TP2 | TP-THRU | TestPoint | 1 |
| Attenuator | Digital Atten, Parallel Control, Pos Volt | U1 | DG983-1 | DAT-31 | 1 |
| Comment | Digital Atten, Parallel Control, Pos Volt | U2 | DG983-1 | DAT-31 | 1 |
| Reg, Volt, SHDN | Voltage Regulator with Shutdown | U3 | SOT-23-CT5_N | VREG-SHDN | 1 |
| SN74LVC14ADR | Hex Schmitt-Trigger Inverter | U4, U5 | SOIC-14 | SN74LVC14ADR | 2 |
| Comment | Hex Schmitt-Trigger Inverter | U6 | SOIC-14 | SN74LVC14ADR | 1 |
| Arduino Nano | Arduino Nano | U7 | ARDUINO-NANO-SMD | Arduino-Nano | 1 |
| CREW HOLE, PLATED, 4 | Hole, Plated, 4-40 Screw | X1, X2, X3, X4 | HOLE-PLATE-4-40 | SCREW-HOLE-PLATE-4- | 4 |
| | | | | | |

# APPENDIX B

## Control Software Listing

The Ardunio board software was developed and compiled using the Arduino Development Environment (ADE). The ADE contains a text editor for writing code, a message area, a text console, a toolbar with common functions and other menus. The ADE also permits connections to be made with the microprocessor and transfers code into the same.

Software written for the Ardunio boards is called a sketch. Sketches are C like programs that enable the programmer to call prewritten routines easily. In fact, the boards are simply AVR development boards and one can program them using AVR C or C++ (using avr-gcc and avrdude or AVR Studio programming environments). More information on Arduino can be found on the web site http://arduino.cc.

Here is the listing of the final version of the software[2].

[2] The software listing was made by copying the original code into MacVim then printing as a pdf document. The pdf document was broken into 8 separate pdf pages and imported one by one into this Word document. This was the only way to preserve the formatting and color coding that MacVim provides.

```
/*
 * Name:  Attn_v3_11_1_2011
 * Desc:  Control a digital attenuator from POT
 * Engineer:  Perry Robertson
 *           Sandia National Labs
 * Date:  11/1/2011
 *
 * This code reads the voltage value on a POT from 0 to 5V
 * corresponding to integer values 0 to 1023
 *
 * The data values are digitized (top 6 bits) and output to the
 * attenuator.
 *
 * Battery monitor circuit blinks power light when low battery detected
 *
 * The attenuation, in this version, increases with clockwise rotation
 *
 * A timer is used to put the Arduino to sleep after defined interval
 *  using unsigned long millis() -- returns number of milliseconds since Arduino beg
an
 *  running the current program.  Overflow after approx 50 days.
 *
 * Modified 11-1-11 to show version for 2 sec after power up
 *   showversion(), make sure you change the routine when updating version
 */

#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/io.h>

//#define DEBUG

#define INIT_TIMER_COUNT 6
#define RESET_TIMER2 TCNT2 = INIT_TIMER_COUNT

// Set the pin numbers for the I/O on Deicimila Board
#define POT A7
#define PWR A6
#define LED 13   // on board LED always connected to D13
#define E5  12
#define E4  11
#define E3  10
#define E2   9
#define E1   8
#define E0   7
#define M1   3 // system clock for testing
#define M2   4
#define LED1 5
#define LEA  6

// Battery indicator state voltages
#define BatGreen 0     //  V > .8V
#define BatYellow 1    //  .6V < V < .8V
#define BatRed 2       //  V < .6V

// int - integer value 16-bit 32,767 to -32,768
// long - extended data type for long integer 32-bit 2,147,483,647 to -2,147,483,648
// float - floating-point 32-bit 3.4028235E+38 to -3.4028235E+38

int sleepStatus = 0;        // variable to store a request for sleep
int count=0;                // counter
int potValue=1;             // the last value of the input voltage
int newValue=60;             // the newest value of the pot
float pwrsupValue=0;        // voltage of power supply
int lowBat=0;               // low battery indicator
int delaycnt=750;            // default delay count (basic loop delay)

// variables for Interrupt Timer
int int_counter = 0;
long starttime = 0;
int clk = 0;
unsigned long currTime = 0;
unsigned long oldTime = 0;
```

```
unsigned long deltaTime = 0;      // difference between oldTime and currTime
unsigned long delayTime = 3600000;  // time before we sleep approx 1 hour (3600000),
 5000 for testing

void setup()
{
  // initialize all the digital pins as an output.
  // Set all the digital pins as inputs to emulate open collector outputs
  pinMode(LED, OUTPUT); // this is the LED on Arduino Nano Board
  pinMode(LED1, OUTPUT); // LED1
  pinMode(LEA, INPUT); // LEA
  pinMode(E5, INPUT); // E5
  pinMode(E4, INPUT); // E4
  pinMode(E3, INPUT); // E3
  pinMode(E2, INPUT); // E2
  pinMode(E1, INPUT); // E1
  pinMode(E0, INPUT); // E0
  pinMode(M1, OUTPUT);
  digitalWrite(M1, LOW);
  pinMode(M2, INPUT);
  //digitalWrite(M2, LOW);

  // Turn on the LED until first cycle completes
  digitalWrite(LED, HIGH);       // turn LED on at beginning
  digitalWrite(LED1, HIGH);      // do likewise to the power LED
  #ifdef DEBUG
    Serial.begin(9600); // start a serial port for test outputs
  #endif

  // can put interrupt code here from Sleep demo
  analogReference(DEFAULT);  // sets the AREF=5V
  count=0;

  // get the start time and set oldTime
  currTime = millis();
  oldTime = currTime;

  #ifdef DEBUG
      Serial.println("Setup Complete");
  #endif
} // setup()

void showversion()
{
  // Setup has already turned on all the leds, we need to
  // display the version number on the leds
  // Set all the digital pins as inputs to emulate open collector outputs
  // Current Version = 3
  pinMode(E5, OUTPUT); // E5
  digitalWrite(E5, LOW);
  pinMode(E4, OUTPUT); // E4
  digitalWrite(E4, LOW);
  pinMode(E3, OUTPUT); // E3
  digitalWrite(E3, LOW);
  pinMode(E2, OUTPUT); // E2
  digitalWrite(E2, LOW);
  pinMode(E1, INPUT); // E1
  pinMode(E0, INPUT); // E0
  // might need to cycle the LE to set the new value (all at once)
  // LEA normally an INPUT (attenuators digital inputs latched)
  // here we are going to enable the new data
  pinMode(LEA, OUTPUT);
  digitalWrite(LEA, LOW);  // pull low
  delay(2);                // wait just a little bit
  pinMode(LEA, INPUT);     // relatch the new inputs

} // showversion()

// this wake up routine is not really called because we never wake up!
void wakeUpNow()             // interrupt handler
{
  // execute code after wake-up, before returning to the loop()
}  // wakeUpNow()
```

```
// clock() creates a short pulse each cycle of the main loop()
void clock()             // create cycle clock for testing
{
  if (clk = 0)
  {
    clk = 1;
    digitalWrite(M1, HIGH);
  }
  else
  {
    clk = 0;
    digitalWrite(M1, LOW);
  }

} // clock()

// update the Attenuator digital values from potValue
//    they act like open collector outputs
//    as inputs, they will float high
//    as outputs, set to LOW, they will pull the line down
// potValue range 0 - 63

// stay in the while loop as long as changes are taking place
// when the counter exceeds limit, then return to main loop

void changeAttn()
{
  // set the bits one by one by doing a bit test

  int delaycnt; // temp var how long to take for each cycle
  int count; // number of cycles since last change

  delaycnt=25;    // set the faster delay count for the little loop
  count=0;

  while (count < 300) {

    // Read the Analog Input
    newValue = readPot();         // read the analog input represents attn setting 6
-bits

#ifdef DEBUG
    Serial.print("POT = ");
    Serial.print(newValue, DEC);
    //Serial.print(" = ");
    //Serial.print(newValue, BIN);
    Serial.print("  Vsup = ");
    Serial.print(pwrsupValue, DEC);

    Serial.print("  Count = ");
    Serial.print(count, DEC);

    Serial.print("  lowBat = ");
    Serial.println(lowBat, DEC);
#endif

    // update based on the POT Value
    if (newValue == potValue) {
      count++;
    }
    else {
      potValue=newValue;  // save the new value
      count=0;            // clear the cycle count, we are a changing things

      // make changes using the newest value

      if ( potValue & 1 ) {
        pinMode(E0, INPUT);       // set the output high
      }
      else {
        pinMode(E0, OUTPUT);      // set the output low
        digitalWrite(E0, LOW);
```

```
      }

      if ( potValue & 2 ) {
        pinMode(E1, INPUT);
      }
      else {
        pinMode(E1, OUTPUT);
        digitalWrite(E1, LOW);
      }

      if ( potValue & 4 ) {
        pinMode(E2, INPUT);
      }
      else {
        pinMode(E2, OUTPUT);
        digitalWrite(E2, LOW);
      }

      if ( potValue & 8 ) {
        pinMode(E3, INPUT);
      }
      else {
        pinMode(E3, OUTPUT);
        digitalWrite(E3, LOW);
      }

      if ( potValue & 16 ) {
        pinMode(E4, INPUT);
      }
      else {
        pinMode(E4, OUTPUT);
        digitalWrite(E4, LOW);
      }

      if ( potValue & 32 ) {
        pinMode(E5, INPUT);
      }
      else {
        pinMode(E5, OUTPUT);
        digitalWrite(E5, LOW);
      }

      // might need to cycle the LE to set the new value (all at once)
      // LEA normally an INPUT (attenuators digital inputs latched)
      // here we are going to enable the new data
      pinMode(LEA, OUTPUT);
      digitalWrite(LEA, LOW);  // pull low
      delay(2);                // wait just a little bit
      pinMode(LEA, INPUT);     // relatch the new inputs
    }

    // as long as we are making changes, keep looping here faster
    // than we normally go through the main loop
    //clock();
    //digitalWrite(LED, HIGH);   // set the LED on
    delay(delaycnt);              // wait for a second
    clock();
    //digitalWrite(LED, LOW);    // set the LED off
    delay(delaycnt);              // wait for a second

  }

  // the guy has quit twiddling on the knob, go back to main loop
} // changeAttn()


// read the POT ADC routine
int readPot() {
  int v;
  v = analogRead(POT);    // read the pot value (10 bits)
  v = 1023 - v;           // make attn increase with clockwise rotation
  v /= 16;                // convert to a 6 bit value
  if (v > 62) {
```

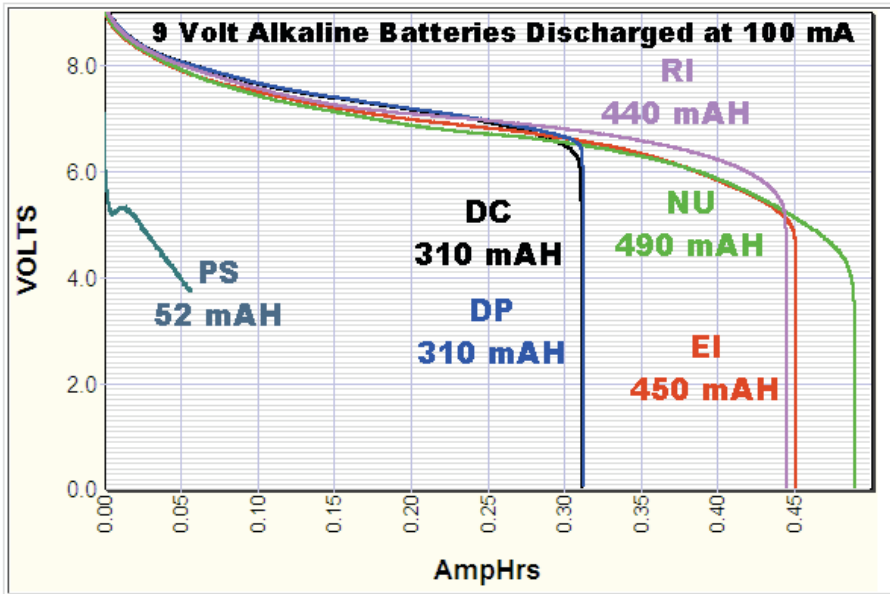
```
    v = 62;
  }
  return v;               // return value
} // readPot()

// read the Power Supply value
float readPwr()
{
  int v;
  float f;
  v = analogRead(PWR);    // read the power supply voltage (div 10)
  // reading will be scaled 5V=1023, scale .9*1023/5=184
  f = 5.0*v/1023;         // scale reading

  return f;               // return value
} // readPwr()

float batTest()
{
  float v;

  // read the power supply voltage
   v = readPwr();      // the 9V power supply Div 10 (0-.9V)
   if (v > .8) {
     lowBat = 0; // battery good
   }

   if ((v <=.8) && (v>=.6)) {
     lowBat = 1;
   }

   if (v < .6) {
     lowBat = 2;
   }

   #ifdef DEBUG
     Serial.print("Vsup=");
     Serial.print(v, DEC);
     Serial.print("  lowBat=");
     Serial.print(lowBat, DEC);
   #endif

   return v;

} // batTest()

// blink the power LED a number of times depending on the level of the
// battery.  lowBat is gloabal variable set in batTest()
void blinkPwrLed()
{
  int i;

  for(i=0; i<=lowBat; i++) {

  #ifdef DEBUG
     Serial.print(" lowBat=");
     Serial.print(lowBat, DEC);
     Serial.print(" BLINK ");
   #endif

  // one pulse no matter what
      digitalWrite(LED1, HIGH);        // turn on LED
      delay(100);
      digitalWrite(LED1, LOW);       // leave LED off
      delay(100);
  }

} // blinkPwrLed()

// blink the LED 10 times just before we go to sleep
void sleepBlink()
{
      int i;
```

```
      for(i = 0; i<10; i++) {
         digitalWrite(LED1, HIGH);       // turn on LED
         delay(100);
         digitalWrite(LED1, LOW);        // turn off LED
         delay(100);
      }

} // sleepBlink()

// go to sleep routine
void sleepNow() {
  /* Now is the time to set the sleep mode. In the Atmega8 datasheet
   * http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf on page 35
   * there is a list of sleep modes which explains which clocks and
   * wake up sources are available in which sleep mode.
   *
   * In the avr/sleep.h file, the call names of these sleep modes are to be found:
   *
   * The 5 different modes are:
   *     SLEEP_MODE_IDLE         -the least power savings
   *     SLEEP_MODE_ADC
   *     SLEEP_MODE_PWR_SAVE
   *     SLEEP_MODE_STANDBY
   *     SLEEP_MODE_PWR_DOWN     -the most power savings
   *
   * For now, we want as much power savings as possible, so we
   * choose the according
   * sleep mode: SLEEP_MODE_PWR_DOWN
   *
   */

   // shut off all the digital outputs (and the LEDS
   // by setting their modes to INPUT
   // this should leave the digital attenuators in last state and keep
   // it from changing (until power on the board is cycled
   pinMode(LED, INPUT); // this is the LED on Arduino Nano Board
   pinMode(LED1, OUTPUT); // LED1
   pinMode(LEA, INPUT); // LEA, leave high to keep from enabling board
   pinMode(E5, OUTPUT); // E5
   pinMode(E4, OUTPUT); // E4
   pinMode(E3, OUTPUT); // E3
   pinMode(E2, OUTPUT); // E2
   pinMode(E1, OUTPUT); // E1
   pinMode(E0, OUTPUT); // E0
   pinMode(M1, INPUT);
   pinMode(M2, INPUT);
   digitalWrite(E5, LOW); // write low turns off LEDs
   digitalWrite(E4, LOW);
   digitalWrite(E3, LOW);
   digitalWrite(E2, LOW);
   digitalWrite(E1, LOW);
   digitalWrite(E0, LOW);
   digitalWrite(LED1, LOW);

   set_sleep_mode(SLEEP_MODE_PWR_DOWN);   // sleep mode is set here

   sleep_enable();          // enables the sleep bit in the mcucr register
                            // so sleep is possible. just a safety pin

  /* Now it is time to enable an interrupt. We do it here so an
   * accidentally pushed interrupt button doesn't interrupt
   * our running program. if you want to be able to run
   * interrupt code besides the sleep function, place it in
   * setup() for example.
   *
   * In the function call attachInterrupt(A, B, C)
   * A   can be either 0 or 1 for interrupts on pin 2 or 3.
   *
   * B   Name of a function you want to execute at interrupt for A.
   *
   * C   Trigger mode of the interrupt pin. can be:
   *             LOW        a low level triggers
```

```
   *             CHANGE     a change in level triggers
   *             RISING     a rising edge of a level triggers
   *             FALLING    a falling edge of a level triggers
   *
   * In all but the IDLE sleep modes only LOW can be used.
   */
/*
  // Perry doesn't need an interrupt
  attachInterrupt(0,wakeUpNow, LOW); // use interrupt 0 (pin 2) and run function
  // wakeUpNow when pin 2 gets LOW
*/

    sleep_mode();            // here the device is actually put to sleep!!
                             // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP

    sleep_disable();         // first thing after waking from sleep:
                             // disable sleep...
 //   detachInterrupt(0);      // disables interrupt 0 on pin 2 so the
                             // wakeUpNow code will not be executed
                             // during normal running time.

} // sleepNow()

// Main loop -- enter here after setup()
void loop()
{
  delay(50);              // give time to see LED1 turn on
  digitalWrite(LED, LOW);  // turn it off and leave it off
  digitalWrite(LED1, LOW);
  showversion(); // show the version number on LEDs
  delay(50);

  // every time through the loop, read the battery voltage
  // may want to slow this down to save power later on
  pwrsupValue = batTest();
  // main loop delay
  delay(delaycnt);
  blinkPwrLed(); // flash the on board led with power state
  delay(delaycnt);
  //clock();

  // every time through loop, read the pot and see if we need to respond to user inp
ut
  // Read the Analog Input
  newValue = readPot();         // read the analog input represents attn setting 6-b
its
  count++;                      // increment each time through the loop
                                // not really doing much with this except when debug
ging

  if (newValue != potValue) {
    changeAttn();             // change the values to the attenuators
    count=0;
    currTime = millis();      // reset the timer values
    oldTime = currTime;
  }

  // see if enough time has passed since the guy twiddled my knob
  currTime = millis();
  deltaTime = currTime - oldTime;  // number of milliseconds
  #ifdef DEBUG
     Serial.print(" dtime=");
     Serial.print(deltaTime, DEC);
  #endif
  if (deltaTime > delayTime) {     // we are going to sleep and save battery cause s
omeone
                                   // has left me on for a long time

    // create a pulse so we can tell when we go to sleep
    clock();

    // blink the power led 10 times before going to sleep
```

```
    sleepBlink();

    sleepNow();

  }

#ifdef DEBUG
  Serial.print("  POT = ");
  Serial.print(newValue, DEC);

  Serial.print("  Count = ");
  Serial.println(count, DEC);
#endif

} // loop()
```

# APPENDIX C


9 Volt Alkaline Batteries Discharged at 100 mA
RI
440 mAH
NU
490 mAH
DC
310 mAH
DP
310 mAH
EI
450 mAH
PS
52 mAH
VOLTS
8.0
6.0
4.0
2.0
0.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
AmpHrs

- [illegible]

## Distribution

All Electronic Copies

| | | | |
|---|---|---|---|
| 1 | MS0341 | Perry J. Robertson | 1751 |
| 1 | MS1064 | Rick Kellogg | 2616 |
| 1 | MS0899 | Technical Library | 9536 |

Sandia National Laboratories