

Project Abstract

(VI)sualizer: A Smart Electronic Load

An instrument for profiling solar, chemical, and grid-powered energy delivery devices



Entry # NXP3767

Submitted to

Circuit Cellar

for

The NXP mbed Design Challenge

Introduction

This document describes the “(VI)sualizer” -- a lab instrument for profiling solar, chemical, and grid-powered energy delivery devices.

The (VI)sualizer gets its name from the fact that it allows the user to examine – or visualize -- the voltage (V) and current (I) delivery ability of an energy source. It can calculate the load resistance into which a solar cell delivers maximum power, or measure and capture the Amp-hour capacity of a battery, or, using pulsatile loads, test the voltage regulation accuracy of a conventional power supply.

By including the signal generation, testing algorithms, and data capture/storage functions in a single instrument, the (VI)sualizer offers an advantage over lash-ups using several pieces of commercially available gear to achieve the same results.

Hardware overview

Figure 1 illustrates the main components of the (VI)sualizer. An mbed module accepts input data from the local and/or remote interfaces, commands the desired load current, and monitors operating conditions to protect against over stress.

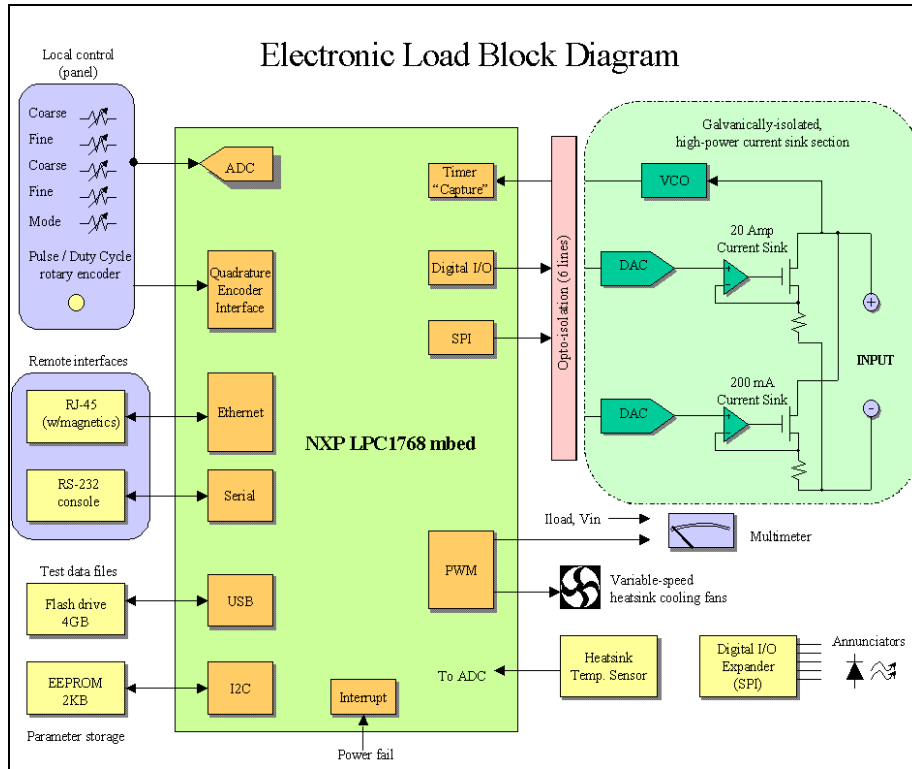


Figure 1 - Block Diagram

The load current is controlled by a bank of power MOSFETs mounted on fan-cooled heat sinks, controlled by a DAC/op-amp drive circuit that is used to set the target current. The DACs are driven through an optical isolation interface.

A voltage-controlled oscillator is used to sense the applied voltage. Its output is also optically isolated.

Two storage devices augment the mbed’s native storage: one, a 4GB flash memory (USB flash drive) for test data files and the other a small EEPROM for parameter storage.

A meter shows current, voltage, wattage, and pulse values. Some signals are generated by PWM outputs on the mbed; the others come directly from hardware.

Another PWM signal drives the variable-speed cooling fans, according to the temperature reading reported by a sensor attached to the heat sink.

Software Overview

The main modules of the firmware are a main task loop that implements a cooperative multi-tasking executive; a collection of state-machine-like task modules that handle the test algorithms; and interrupt handlers that control data capture and pulse generation.

The remote user interface comprises five html/Javascript web pages. They rely on RPC calls to invoke functions on the mbed. Those functions return data to the browser by means of a JSON-compatible string.

Also, two new drivers were written in C++ especially for this project: one for the QEI hardware, and one for Timer 2 “Capture” inputs. A snippet of the Capture driver is shown below:

```

17 /*****
18  * @brief      Create a Capture object and configure it.
19  * @param pin  Selects p30 or p29 as the Capture input
20  * @return     None
21  *****/
22 Capture::Capture(PinName pin)
23 {
24     #define DEFAULT_PCLK_DIV    PCLK_DIV_4        // Default to PCLK/4 (24MHz)
25     #define DEFAULT_EDGE       CAP_EDGE_RISE     // Default to the rising edge
26     #define DEFAULT_INT_MODE   CAP_INT_ON       // Default to interrupts on
27
28     /* Set up clock and power, using default PCLK divider choice */
29     LPC_SC->PCOMP |= PCOMP_PCTIM2; // Turn on the power to the timer
30     LPC_SC->PCLKSEL1 = (LPC_SC->PCLKSEL1 & ~PCLKSEL1_PCLK_TIMER2_MASK) | (DEFAULT_PCLK_DIV << 12);
31
32
33     /* Assign the capture pin. Use a pull-up. Also select capture edge and int mode */
34     switch (pin) {
35     case p30:
36         channel = 0;
37         // Enable pin 30 for CAP 2.0
38         LPC_PINCON->PINSEL0 = (LPC_PINCON->PINSEL0 & PINSEL0_CAP20_MASK) | PINSEL0_CAP20;
39         LPC_PINCON->PINMODE0 = (LPC_PINCON->PINMODE0 & PINMODE0_CAP20_MASK) | PINMODE0_CAP20;
40         // Select the capture edge and int mode
41         LPC_TIM2->CCR = (LPC_TIM2->CCR & ~(uint32_t)0x7) | DEFAULT_EDGE | (DEFAULT_INT_MODE << 2);
42         break;
43     case p29:
44         channel = 1;
45         // Enable pin 29 for CAP 2.1
46         LPC_PINCON->PINSEL0 = (LPC_PINCON->PINSEL0 & PINSEL0_CAP21_MASK) | PINSEL0_CAP21;
47         LPC_PINCON->PINMODE0 = (LPC_PINCON->PINMODE0 & PINMODE0_CAP21_MASK) | PINMODE0_CAP21;
48         // Select the capture edge and int mode
49         LPC_TIM2->CCR = (LPC_TIM2->CCR & ~(uint32_t)0x38) | (DEFAULT_EDGE << 3) | (DEFAULT_INT_MODE << 5);
50         break;

```

Altogether, the software and libraries take up about 135 kB of Flash, and about 8 kB of RAM.

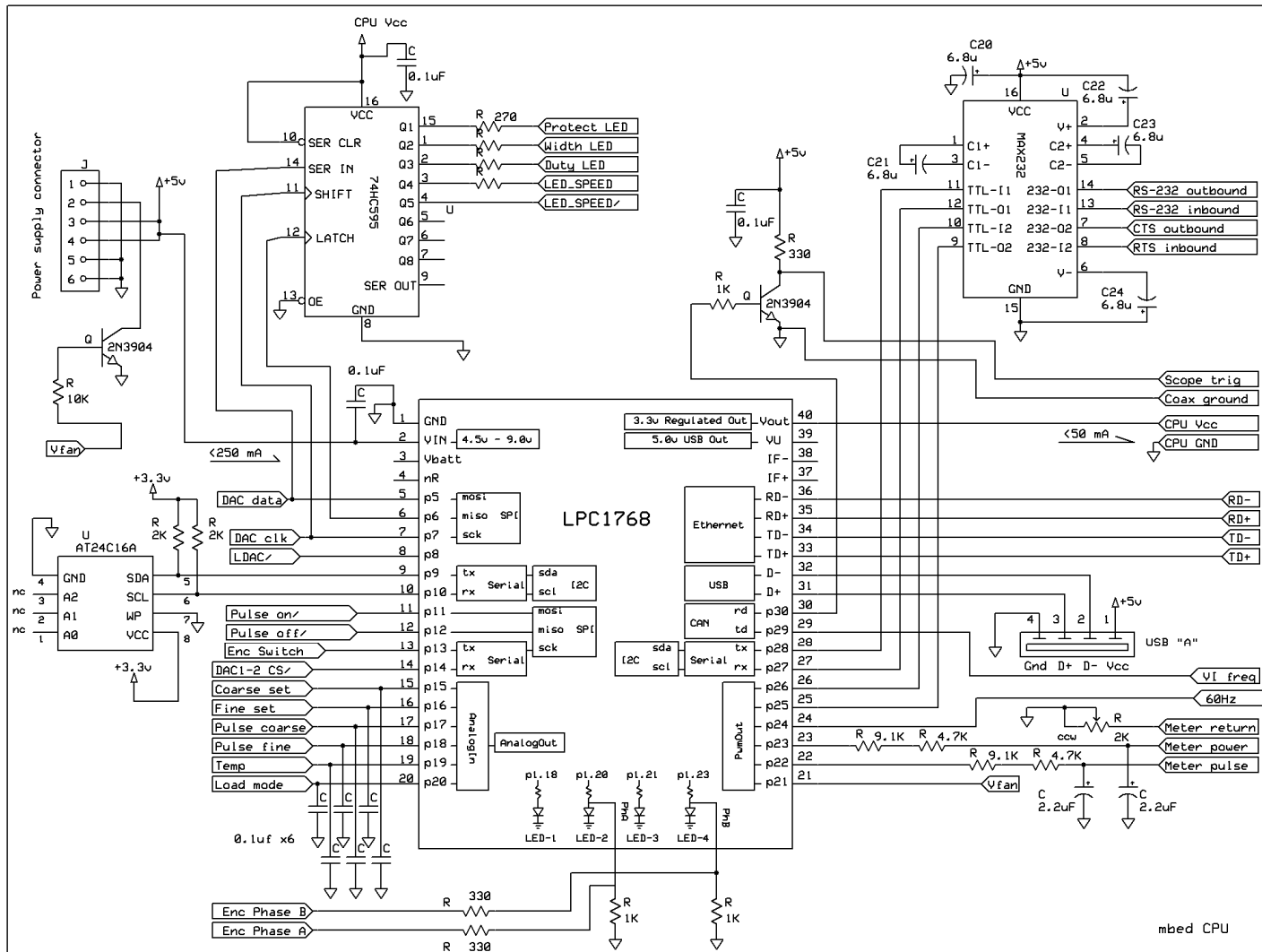
Summary

Every major module of the mbed was used (save for the CAN module, which was not required). All I/O pins were used – in fact, an SPI port expander was used to generate five extra digital outputs. Some 13 mbed Handbook functions were used, along with 4 Cookbook functions and 2 of my own.

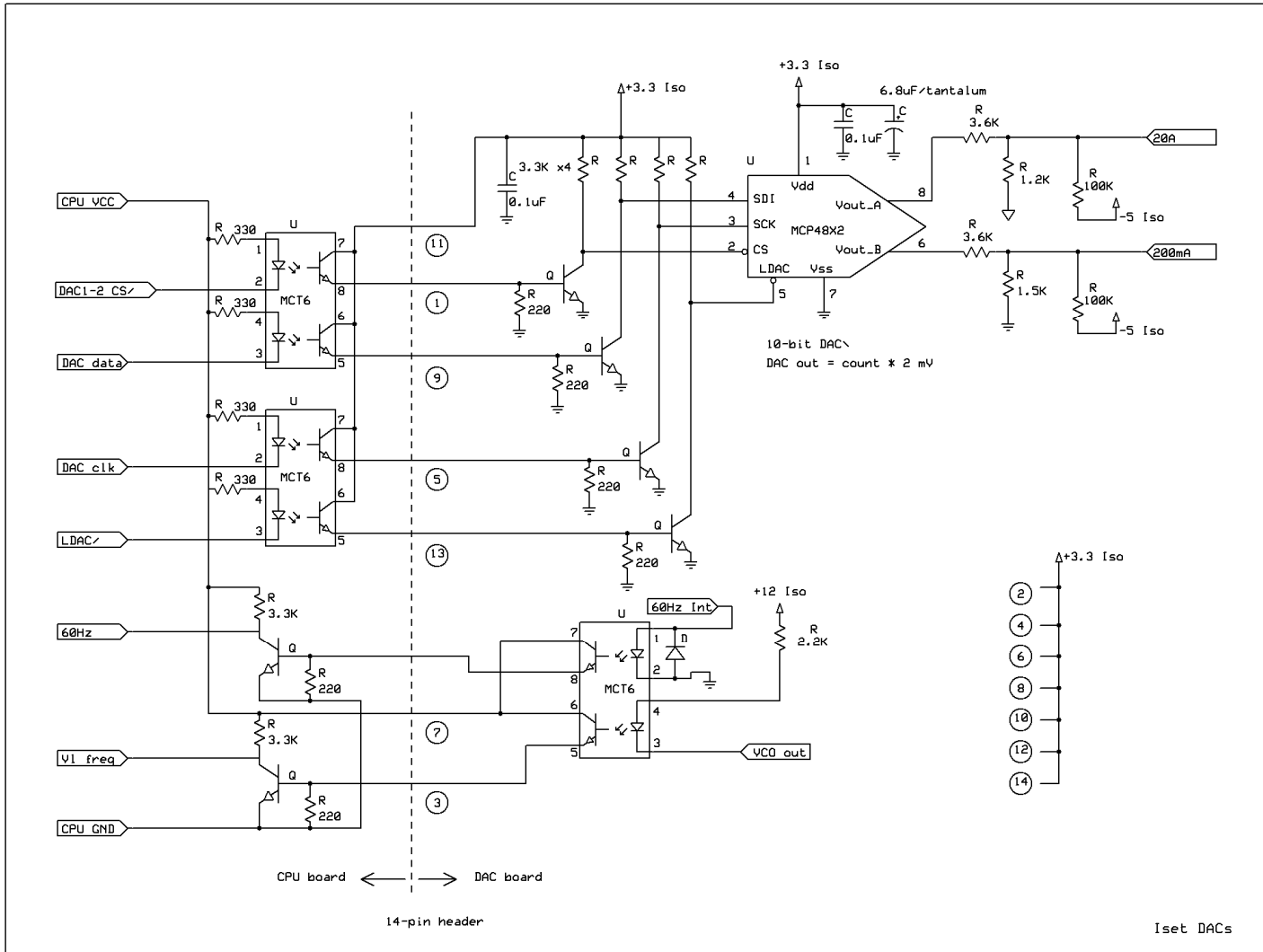
All together, the mbed hardware and software provided an excellent environment for the development of this project.

I can truly say that bringing this design to the same level of complexity would not have been possible in the same time frame had I used a bare-metal processor without the full range of software provided to mbed users.

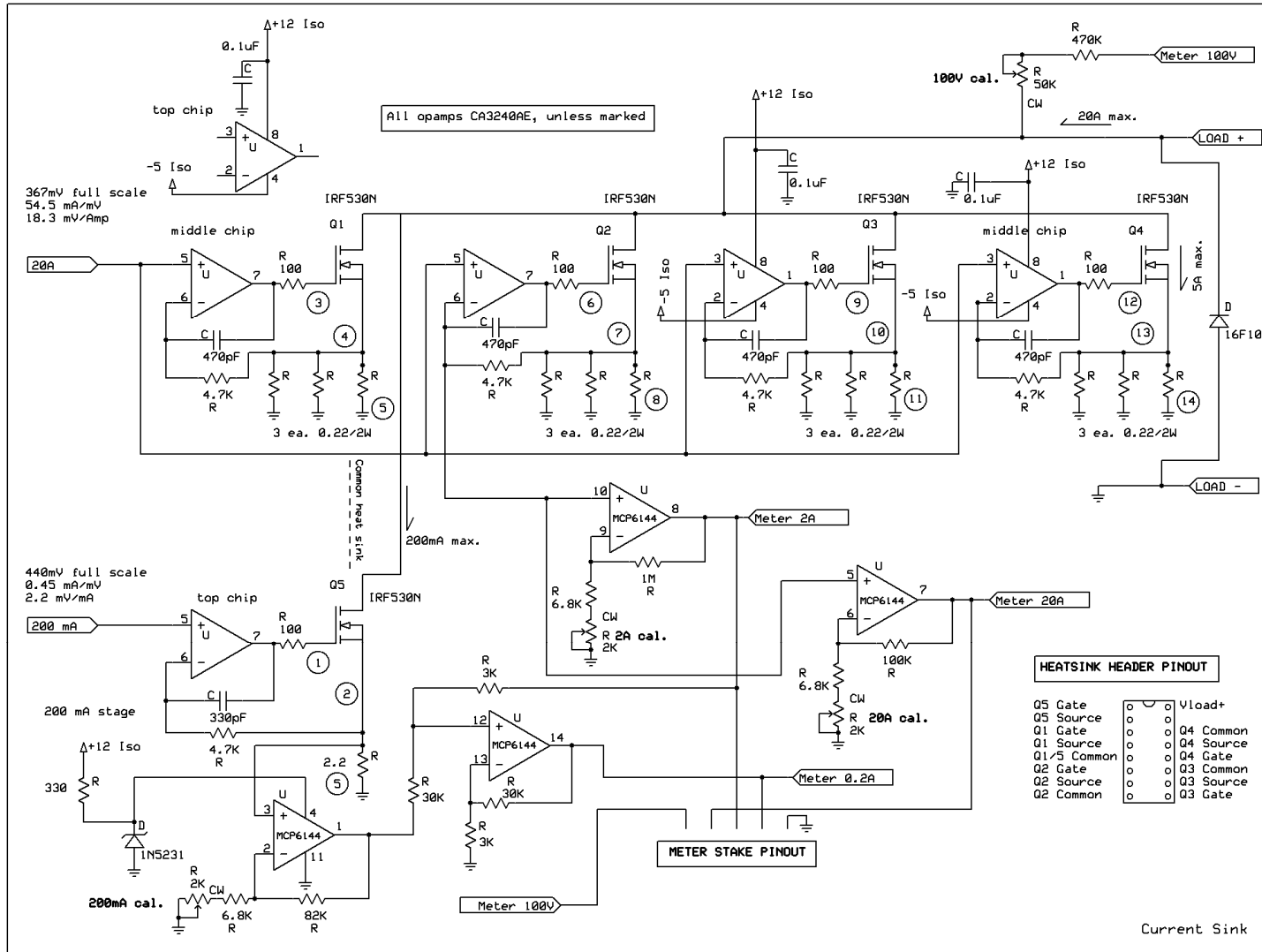
Mbed CPU



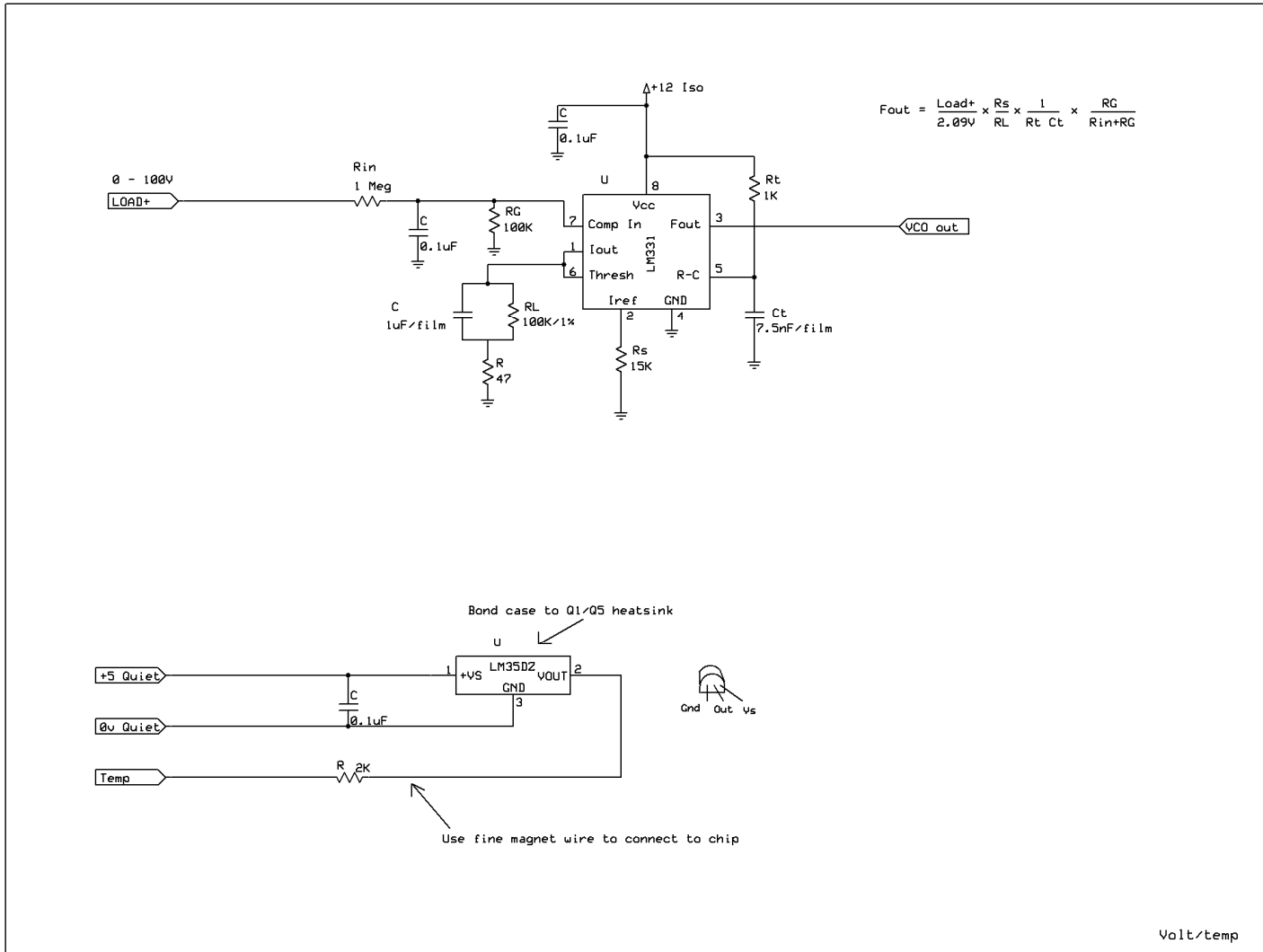
Iset DACs



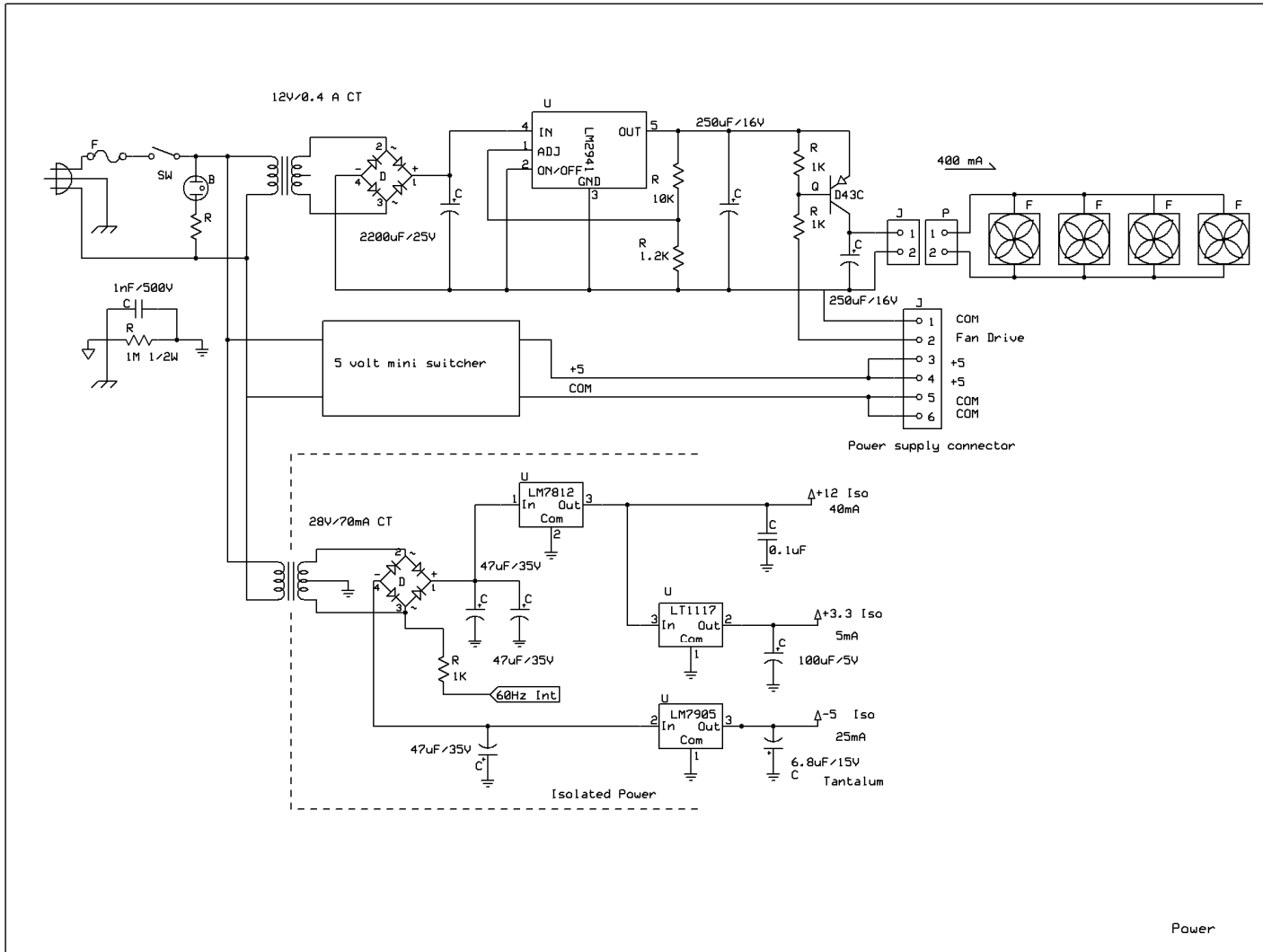
Current Sink



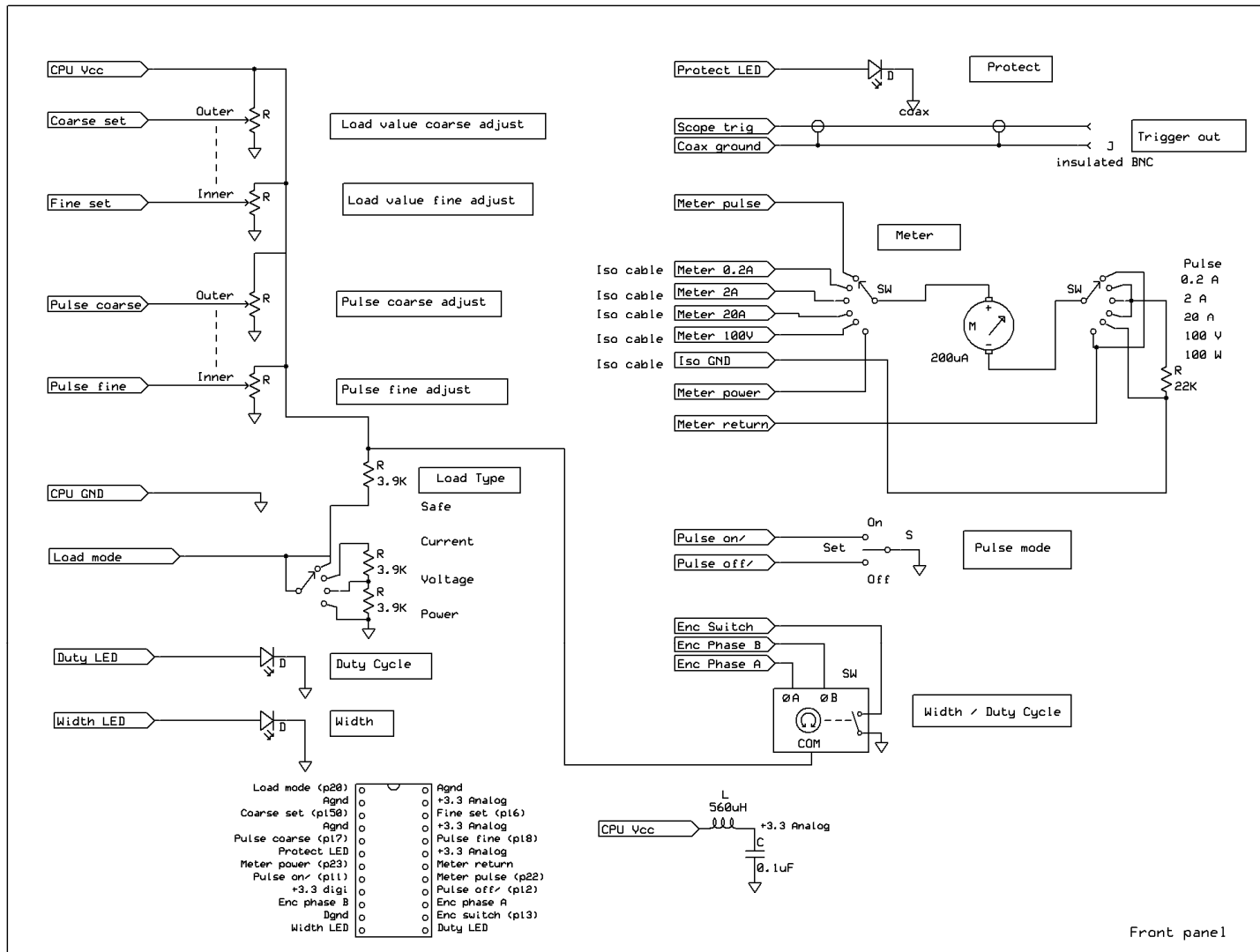
Volt/temp Monitor



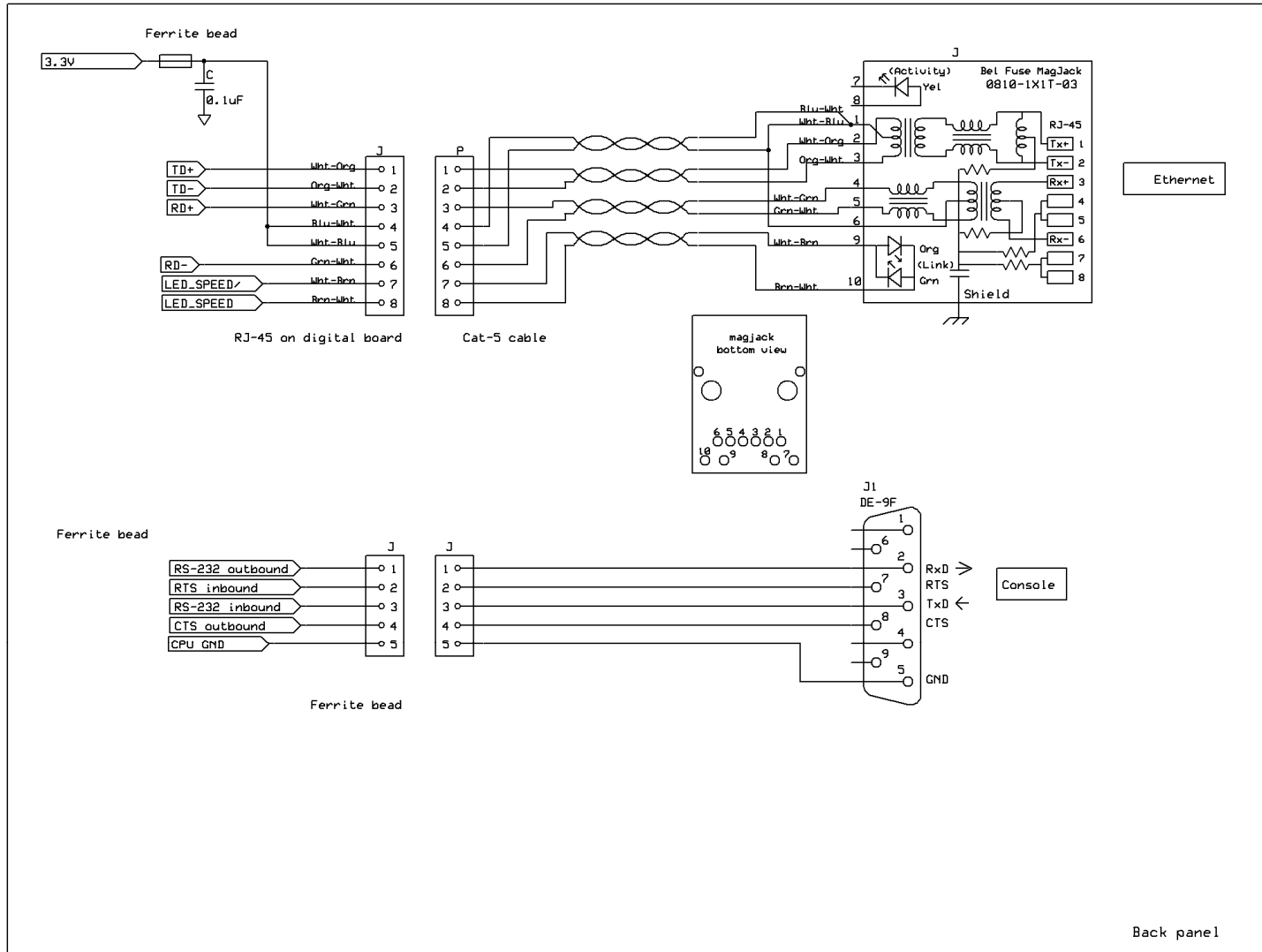
Power



Front Panel



Back Panel



Back panel

