SuRF, the Supply and Read-out/Fan-out board

Johannes Mülmenstädt

March 31, 2004
$Revision: 1.16 $

Abstract

The SuRF (Supply and Read-out/Fan-out) board is part of the ATLAS module burn-in system. It supplies four modules with power and HV and switches data and control between the four modules and a TPCC. In this document we provide a short user's guide to the board; then we describe what is on the board and how the board is computer-controllable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document describes the SuRF board. The SuRF board is one component of the pixel module burn-in system. The other components are the burn-in shell AMBuSh, documented at `http://pixdata.lbl.gov/~surf/`, and the burn-in data viewer, documented at (blah — have to ask Lukaš for documentation).

An important consideration in the design of the pixel-module burn-in system was to make the integration into existing pixel setups easy. The only new piece of hardware required for the burn-in system is the SuRF (**Su**pply and **R**eadout/**F**anout) board. The SuRF board fans out control from a TPCC to four modules, multiplexes data returned by the modules and provides power and high voltage to the modules. It also provides slow controls — setting analog and digital supply voltages for each module and reading back currents and temperatures — through a USB interface.

This document is structured as follows. Ch. 2 describes the SuRF board from a user's perspective. The chapter deals with elementary matters:

- what sits where on the board; and
- what connector should connect to what.

It also covers more involved topics:

- how to download firmware into the board, what jumpers to load on a new board, what kluges need to be applied to a new board to make it work; and
- how the board signals that it's unhappy, what diagnostic tools exist to figure out what is making the board unhappy, and how to use the diagnostic tools to make the board happy again.

For the benefit of the designers of future generations of burn-in systems, Ch. 3 describes the components of the SuRF board and the firmware that runs them. This section should help with isolating hardware and firmware bugs that might eventually pop up.

1.1 How to get this document

This document is intended to be used with Rev. 1 of the SuRF board. Since there may be slight differences between revisions, you should not use this document for any other revision of the SuRF board. To get the User's Guide for a different revision of the SuRF board, or to get the most up-to-date version of the User's Guide for Rev. 1, please follow the instructions below.

The User's Guide is distributed with the ATLAS pixel module burn-in software. Typing `make surf.ps` in the `jboard/` directory will generate the PS version of the Guide. The User's Guide is also available as a stand-alone PS file (and a sloppy HTML rendition of the PS file) on the ATLAS pixel module burn-in page (`http://www-cdf.lbl.gov/~jmuelmen/ambush`). A copy of the most recent version for each revision of the SuRF board is kept there.

Chapter 2

Using the hardware

This chapter gives a quick overview over the SuRF board. Then it describes the care and maintenance your board may need in times of illness. Finally we prepare you for the magical moment when your board will have little boards of its own — especially the corrective surgery required on newborn boards (brith), the tests for signs of life and the necessary calibrations.

2.1 Getting started quickly

This section gives an overview over using the SuRF board. We first describe how to connect the board to the rest of the burn-in setup. Then we walk the reader through some short diagnostic steps using host-side software that test whether the board works as expected.

2.1.1 Connecting the wires

Fig. 2.1 shows how the parts go together. But before you actually put the parts together, please read this warning.

> **Warning!** The state of the module supply regulators is undefined when the SuRF board is powered up. For this reason, you should never attach modules to the SuRF board until the board has completed its boot-up routine and all regulator enable LEDs are off. This serious misdesign in the SuRF board will be corrected in future revisions of the board.

- The SuRF board wants to be connected to a TPCC through one of the four module connectors on the TPCC.
- There is a high-voltage power supply connector for each module which simply gets routed to the module ELCO connector.
- The SuRF board also requires one bulk power supply to feed the four analog regulators and one bulk power supply to feed the four digital regulators; and one power supply that drives the control circuitry on the board. 5 V power supplies work well for all three needs. The module supplies need to supply up to 6 A, while the board supply needs to supply up to 1 A. Please be careful about the pinout of the power connectors: the two pins facing the bottom (SN label/USB connector side) of the board are ground; the two pins facing the top (TPCC connector side) of the board are power.

Jumper name	Purpose	Default setting
J6	USB signal routing	Connect 1–2, 5–6, 7–8, 11–12
J8	Debug UART signal routing	Not loaded
JP1	Clock delay	Delay cable/jumper connected
JP2	EEPROM write protect (WP)	Loaded (protected)
JP3	I^2C SCL line	Loaded
JP4	I^2C SDA line	Loaded
JP400	Tristate LVDS drivers	Not loaded

Table 2.1: SuRF board jumper settings

- Finally, the USB port of the board needs to be connected to a computer running the AMBuSh software. If more than one SuRF board is desired, the boards can be daisy-chained. Connect the upstream USB connector (USB1) of the first board to the host computer and the downstream USB connector (USB2) of the first board to the USB1 connector of the second board etc.

Fig. 2.2 shows the location of the connectors on the SuRF board. Also shown are the various jumpers; refer to Table 2.1 for the correct jumper settings.

2.1.2 Running some quick tests

If you are the proud owner of a SuRF board, you more likely than not also want to own AMBuSh (`http://pixdata.lbl.gov/~surf/`). At the AMBuSh prompt, you can perform some quick tests that your SuRF boards are working.

When you plug the USB end of the SuRF board into the host computer, the USB OK LED must light up. If that is not the case, check that the J6 jumper is set correctly.

When the USB OK LED indicates that the board has upstream connectivity, type `surf_print_boards();` at the AMBuSh prompt (remember to hit return twice). This should produce a listing like the following:

```
002/022 4242/BEEF 200402LBL004 (rev 00.05.02)
002/020 4242/BEEF 200402LBL006 (rev 00.05.02)
```

Don't be shy about daisy-chaining all your boards together. In fact, attach a USB mouse or digital camera to the last one. As long as the USB OK LED lights up on each, the boards will be happy.

Now you can address each module by its serial number. For example, if board `200403LBL001` is connected to your computer, you can enable the digital regulator for channel 2 on that board by saying `surf_brd_end("200403LBL001", 2, 1);` and disable it with `surf_brd_end("200403LBL001", 2, 0);` you can easily see if the board understood you by checking that the END2 LED turns on when you issue the former command and turns off when you issue the latter.

A comprehensive listing of functions that the SuRF board understands is given in Ch. 4. That chapter also explains how you can invoke SuRF board functions not by board name and channel number but instead by module serial number. Lastly, if you would like to see a not-so-quick list of tests, please consult Sec. 2.5; there we summarize the tests performed on newborn boards.

2.2 If you encounter problems . . .

2.3 If your board needs new firmware . . .

You want a part in here that talks about how to get the firmware into the board. Blah blah blah.

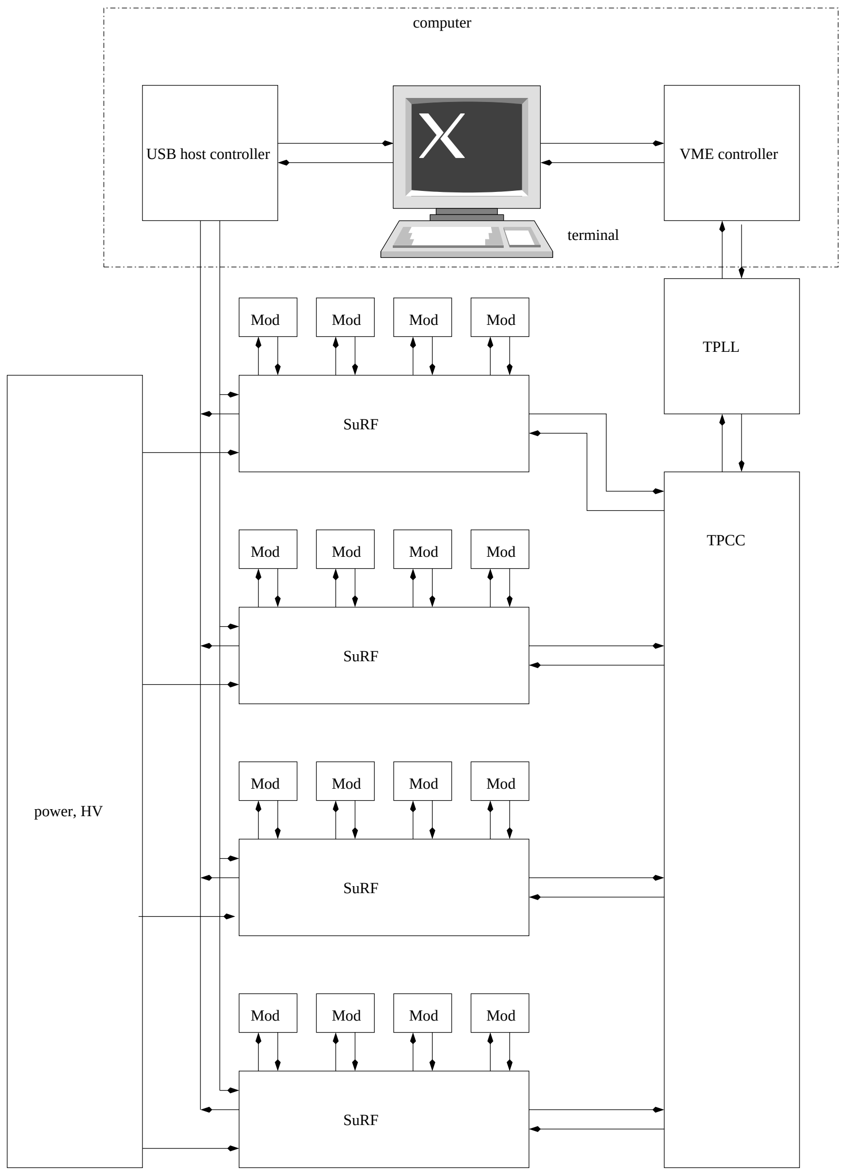

Figure 2.1: Schematic of AMBuSh hardware

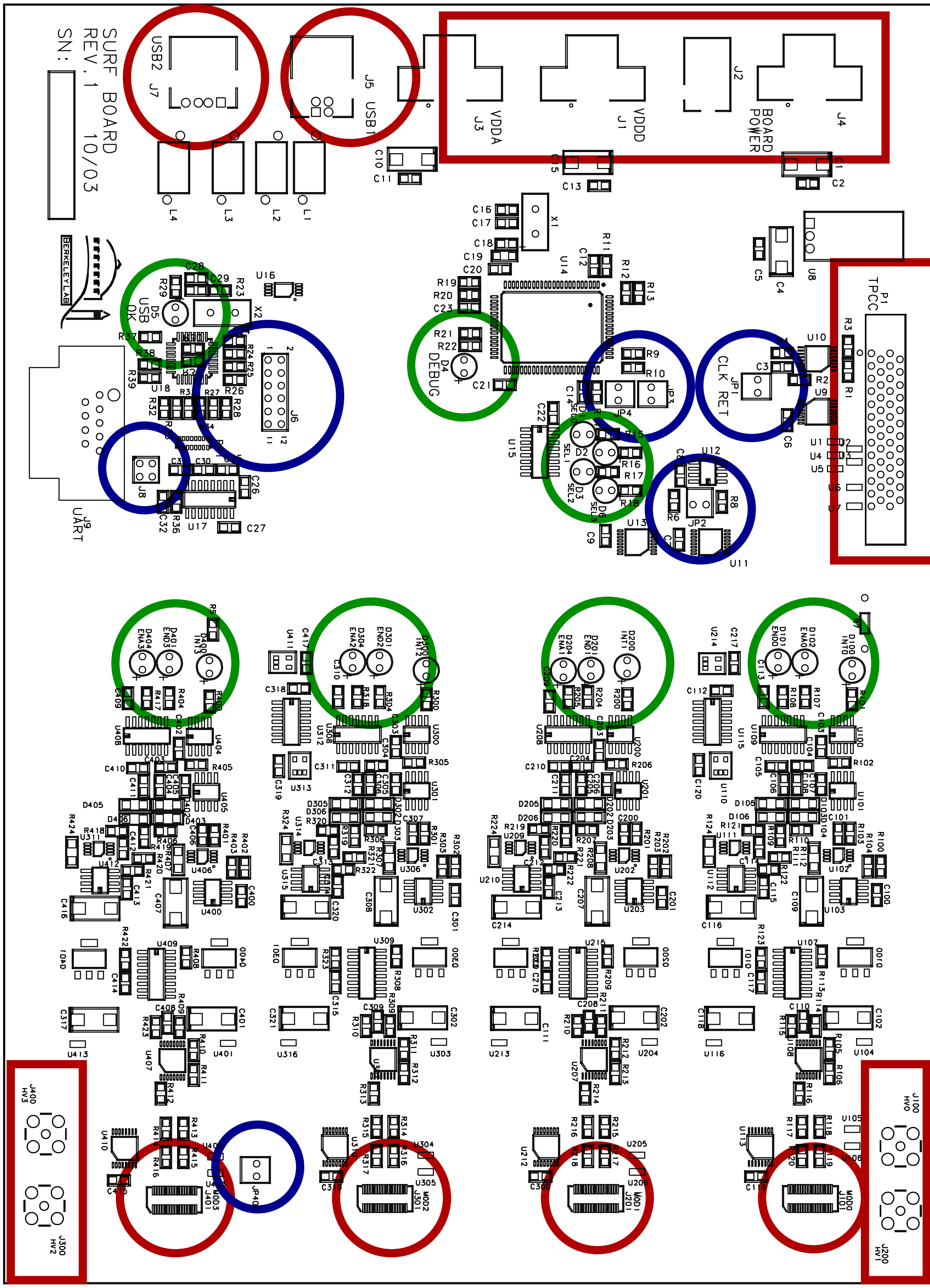

Figure 2.2: Physical layout of the SuRF board. Connectors are shown in red, diagnostic LEDs in green and jumpers in blue.

LED	Purpose
USB OK	Upstream USB connection established
DEBUG	General-purpose debugging LED; probably a bad sign in the production environment. See debugging section (Sec. **??**)
SELx	Module x selected for DTO read (and DCI write if ES = 1)
ENAx	Module x analog supply enabled
ENDx	Module x digital supply enabled
INTx	Module x thermal interlock

Table 2.2: SuRF board status indicator LEDs

2.4 Kluges required for rev. 1 boards

Rev. 1 of the SuRF board requires a number of kluges to work:

- Some series resistors in the CMOS module communications lines need to be jumpered. These resistors are R2, R4, R114, R115, R211, R210, R309, R310, R409 and R423.
- On the LVDS receiver chips the enable pin (pin 16) needs to be tied to high. Note that the orientation of the LVDS receiver bypass cap (good place to get Vcc) is reversed on channel 0!
- The AD5242 chips need the SHDN# pin tied to power, not ground. Lift pin 6 and connect to pin 5.
- Use the LT1713 instead of AD8561 comparator (U100, U200, U300 and U404); also lift pin 2 of the 1713 and bridge a 10k resistor between pin 2 and the end of R102/R206/R305/R405 that's adjacent to pin 3 of the 1713.
- Don't load R100, R111, R202, R221, R302, R321, R402 and R420.

Fig. 2.3 shows where on the board these changes need to be made.

2.5 Testing

Blah blah blah tests on newborn boards. Blah blah blah list

2.6 Calibration

Blah blah blah calibration of ADCs and DACs. Blah blah blah calibration programs, where do calibration curves get stored, blah blah blah

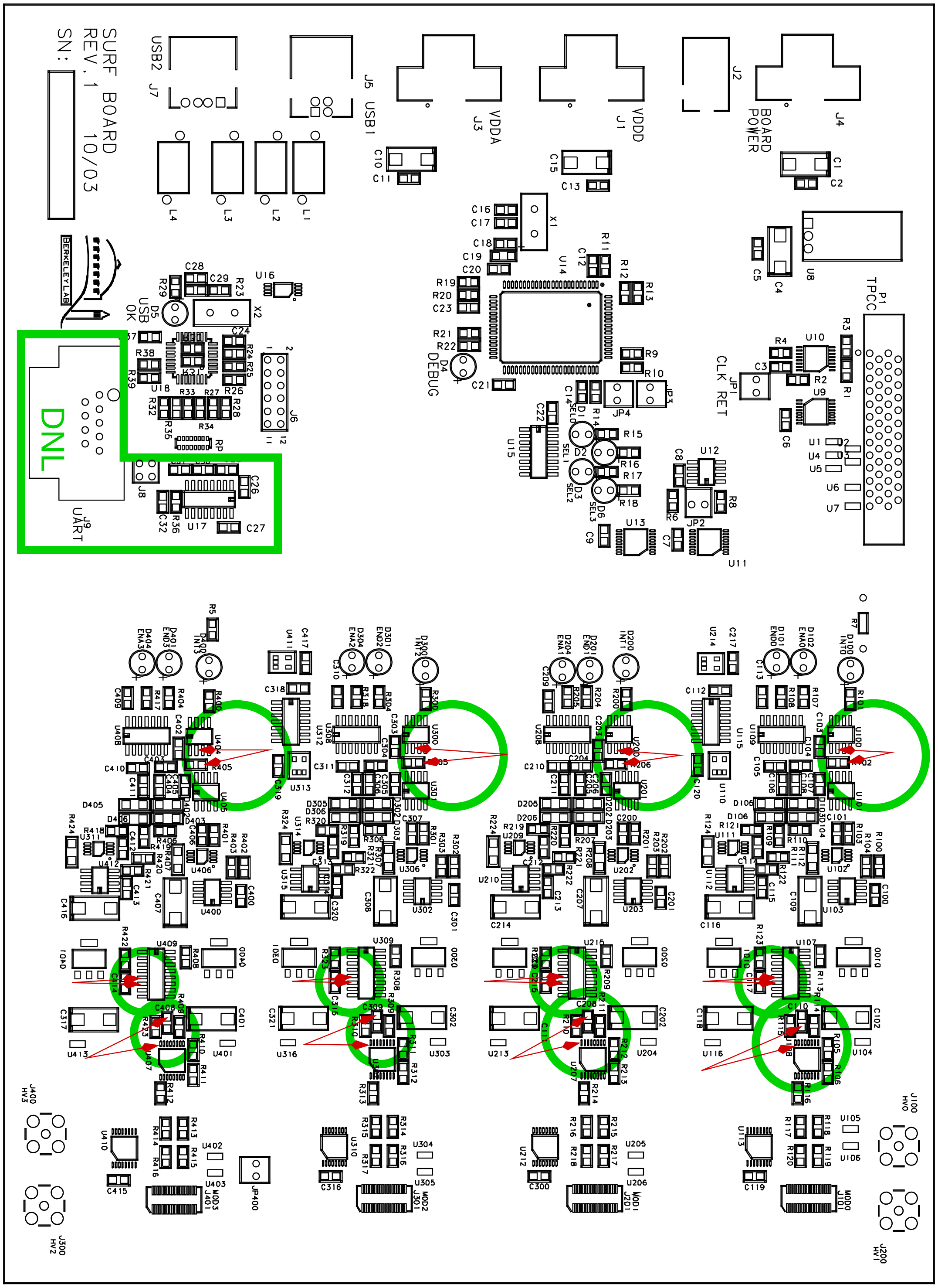

Figure 2.3: Locations of the most invasive kluges required on the SuRF board. Refer to Sec. 2.4 for explanation.

Chapter 3

Hacking the hardware

The SuRF board consists of four components.

3.1 Regulators

Linear regulators provide VDDD and VDDA for each module connected to the SuRF board. Fig. 3.1 shows the two regulators supplying each module. The resistor R?/R? current-limits the regulators to

$$I_{\mathrm{max}} = \frac{57\ \mathrm{mV}}{R_{\mathrm{s}}} = 1.5\ \mathrm{A}.$$

The capacitor-diode network blah blah blah makes the output voltage ramp when the regulator is enabled.

The EZ-USB processor on the SuRF board (Sec. 3.6) can set the voltage level of each regulator by talking to the digital potentiometers (U?); the processor can also disable each regulator entirely. Fig. 3.4 shows a plot of analog and digital supply voltage as a function of potentiometer resistance (R_{v}).

3.2 Temperature interlock

The SuRF board reads out the NTC on each module and shuts off that module's supplies if the temperature exceeds blah blah blah. The EZ-USB can reset the supplies by pulling on the NRESET line.

3.3 Clock and control fan-out

The TPCC sends control (DCI) and 40 MHz clock signals over LVDS lines. These signals are converted to CMOS on the SuRF board and fanned out to a separate set of LVDS drivers for every module. The clock is unconditionally distributed to all modules. If the EZ-USB pulls down the ES (enable select) line, all modules receive DCI. If the ES line is high, DCI is only transmitted to the module that is selected using the NSEL line.

3.4 Data multiplexing

The module returns two bits of data. These bits are converted from LVDS to CMOS on the SuRF board. Only one module at a time can be read out; the EZ-USB selects which module is read out by pulling down

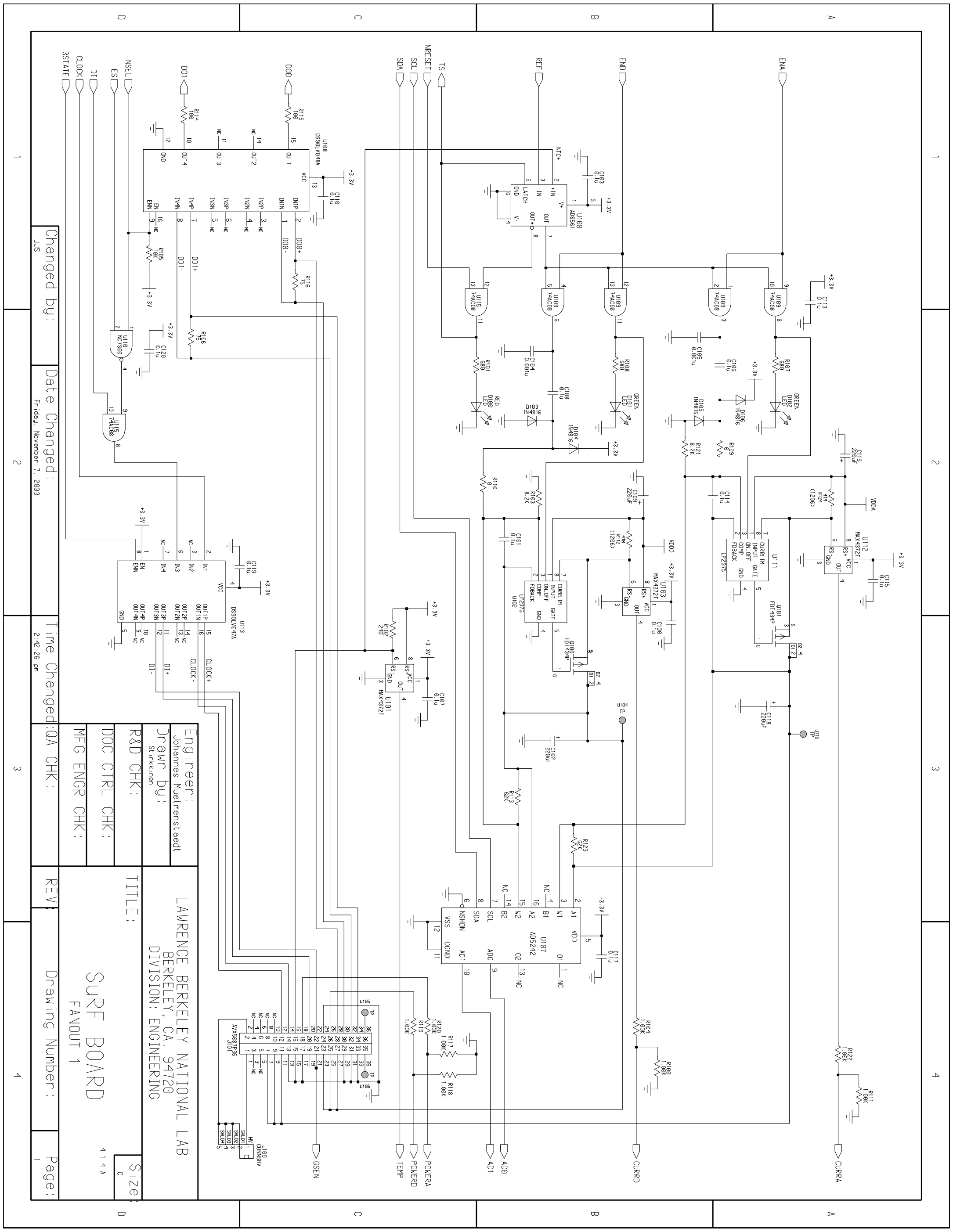

Figure 3.1: Regulator schematic

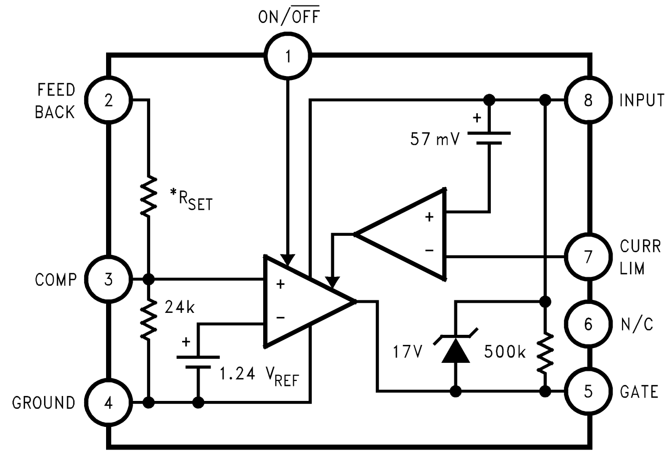

Figure 3.2: LP2975 schematic[?]. $R_{\text{set}} = 39.9\ \text{k}\Omega$.

Figure 3.3: LP2975 schematic[?]. $R_{\text{set}} = 39.9\ \text{k}\Omega$.

the NSEL line for that module. The data bits from the selected module are converted back to LVDS and transmitted to the TPCC.

3.5 Clock return

In the pixel module scheme, the board closest to the modules returns the 40 MHz clock. The jumper J? allows inserting a delay into the clock return to simulate the effect of long lines between the SuRF board and the modules.

3.6 Microprocessor/USB controller

All slow-control interaction between the user and the SuRF board happens through the USB controller. Our USB controller chip (a Cypress EZ-USB AN2131) is a combination of USB line driver, RAM, souped-up 8051 microprocessor and I^{2}C controller.

3.6.1 USB interface

The USB controller part of the EZ-USB provides an interface to the SuRF board for the machine running the AMBuSh software. It accepts requests to enable or disable the various features of the SuRF board; set the settable settings; and read out the status of the board. Tables 3.1 and 3.2 list the requests that the host computer is allowed to make of the SuRF board. (The additional requests required by the USB protocol specification are not listed in this here manual.)

Global requests are made through control transfers to the EZ-USB's endpoint EP0. The format of the request is shown in Table 3.3.

Module-specific requests are made through bulk transfers; for the nth module (zero-offset), the transfer is made to the $n+1$st endpoint. (This arrangement is necessary because the USB specification requires EP0 to be the control endpoint.) Byte 0 of a module-specific transfer specifies the request (choose from one of those

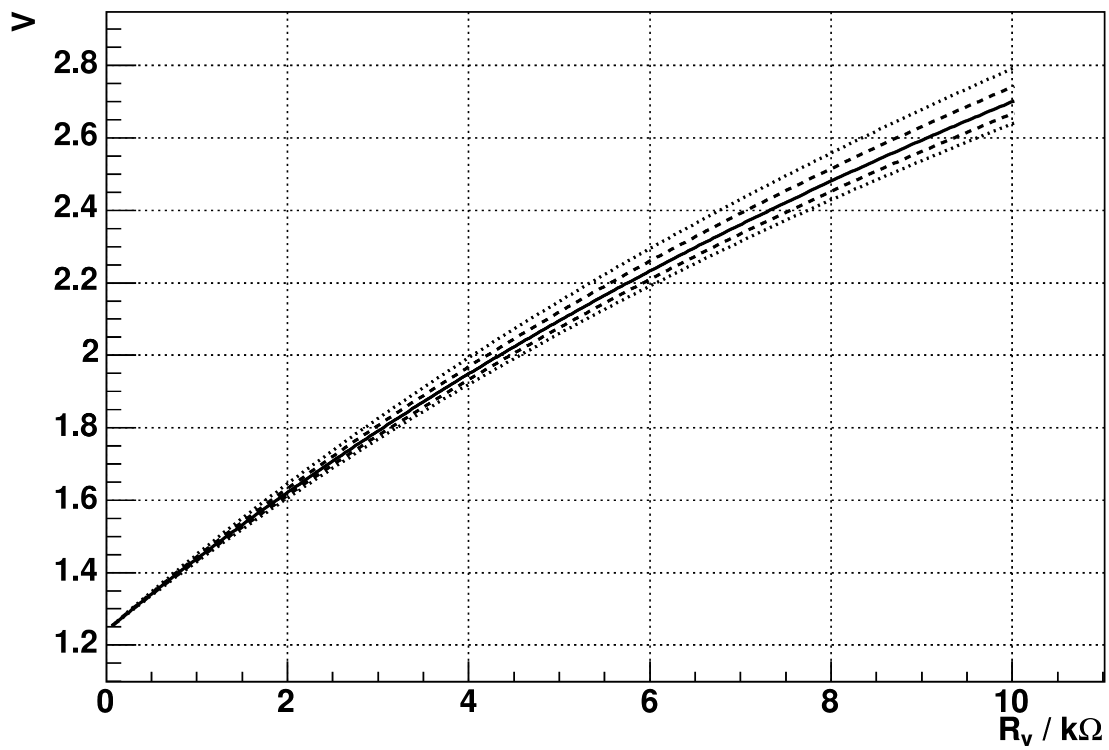

Figure 3.4: The dependence on the high-tolerance (20%) internal 24-kΩ resistor is shown. The solid line indicates 0% deviation from 24 kΩ, the dashed lines ±10% and the dotted lines ±20%.

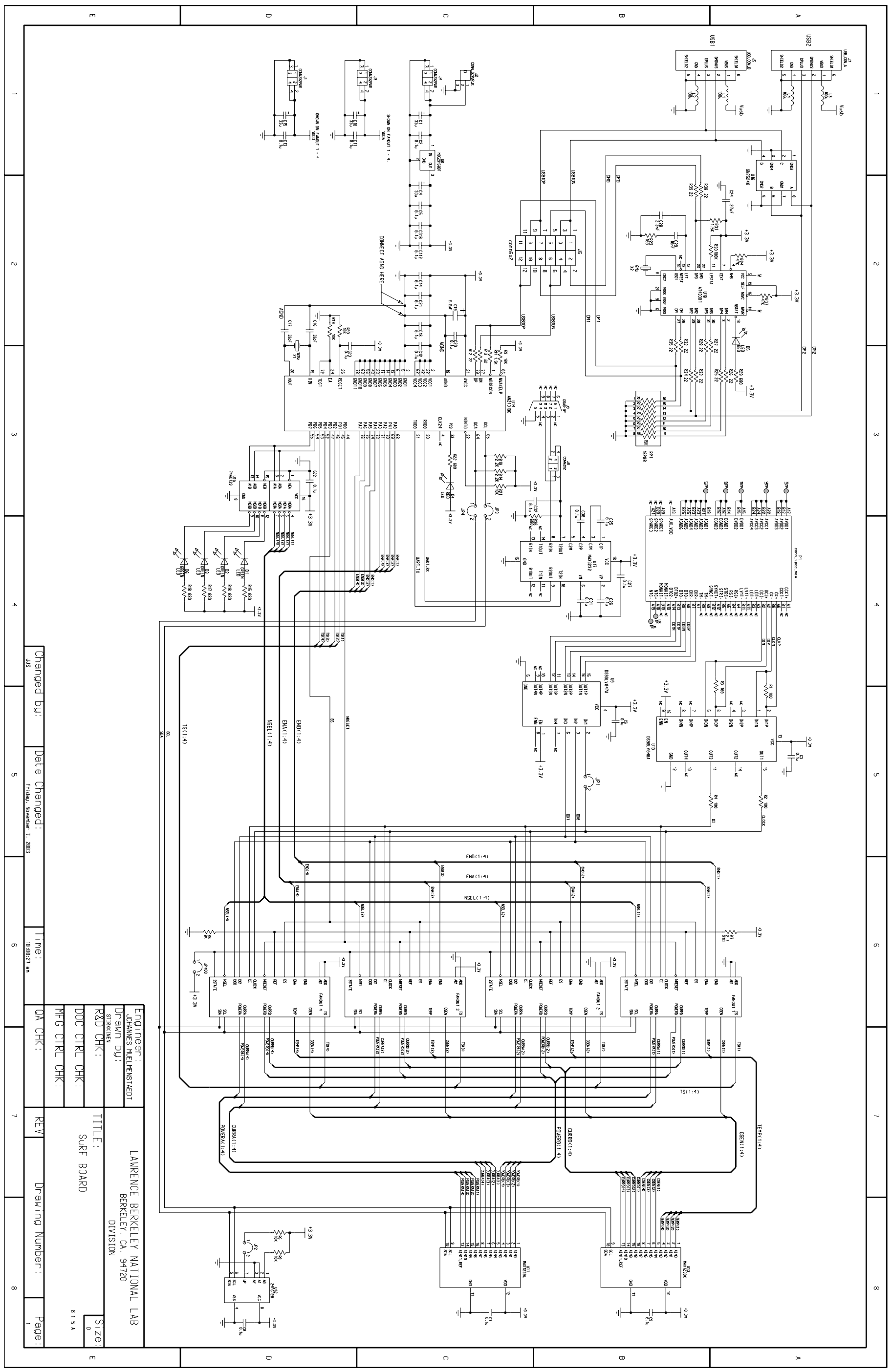

Figure 3.5: The SuRF board

Request name	**Purpose**	**Argument**
`SURF_GLOB_EN_SINGLE`	Mine eyes have seen the glory of the coming of the lord, he is trampling out the vintage where the grapes of wrath have snored	Module number (0...4)
`SURF_GLOB_EN_ANAL`	Enable analog supplies	Bitmap: $0000m_0m_1m_2m_3$
`SURF_GLOB_EN_DIG`	Enable digital supplies	Bitmap: $0000m_0m_1m_2m_3$
`SURF_GLOB_RESET`	Reset temperature interlocking mechanism	None

Table 3.1: Global requests supported by the SuRF board USB interface. $m_n = 1$ indicates that the nth module is enabled. These requests should be addressed to EP 0.

Request name	**Purpose**	**Arguments**
`SURF_MOD_SEL_RD`	Select this module for readback. Unselect whichever was selected before.	None
`SURF_MOD_SET_ANAL`	Set the analog DAC so that the analog supply voltage at the module corresponds to the DAC value specified in the argument.	Byte 1: LSB of 12-bit ADC target value Byte 2: MSB of 12-bit ADC target value Byte 3: DAC hint
`SURF_MOD_SET_DIG`	*Idem* for digital supply voltage.	
`SURF_MOD_STATUS`	Generate the module status report.	None

Table 3.2: Module-specific requests supported by the SuRF board USB interface. A request to module n (zero-offset) should be addressed to EP $n + 1$. See the last paragraph of Sec. 3.6.2 for an explanation of the "DAC hint" argument to the `SURF_MOD_SET_ANAL`/`SURF_MOD_SET_DIG` request.

listed in Table 3.2); the remaining 0 to 63 bytes are arguments to the command as specified in Table 3.2. Note that if you ask an endpoint for the status of its module by sending it the `SURF_MOD_STATUS` request, the next request you make to that endpoint must be a bulk in request so that the endpoint can rid itself of the status information it has generated. Table 3.4 shows the format of the status report.

3.6.2 Microprocessor I/O

The microprocessor part of the EZ-USB has two eight-bit digital I/O ports and an I^{2}C bus controller. The I/O bits are allocated as follows:

*Regulator enable (*END *and* ENA*):* Eight bits are used to turn on the eight regulators independently of each other.

Request name	**Purpose**	**Argument**
`SURF_GLOB_EN_SINGLE`	Mine eyes have seen the glory of the coming of the lord, he is trampling out the vintage where the grapes of wrath have snored	Module number (0...4)
`SURF_GLOB_EN_ANAL`	Enable analog supplies	Bitmap: $0000m_0m_1m_2m_3$
`SURF_GLOB_EN_DIG`	Enable digital supplies	Bitmap: $0000m_0m_1m_2m_3$
`SURF_GLOB_RESET`	Reset temperature interlocking mechanism	None

Table 3.3: Global request format.

Request name	Purpose	Argument
SURF_GLOB_EN_SINGLE	Mine eyes have seen the glory of the coming of the lord, he is trampling out the vintage where the grapes of wrath have snored	Module number (0…4)
SURF_GLOB_EN_ANAL	Enable analog supplies	Bitmap: $0000m_0m_1m_2m_3$
SURF_GLOB_EN_DIG	Enable digital supplies	Bitmap: $0000m_0m_1m_2m_3$
SURF_GLOB_RESET	Reset temperature interlocking mechanism	None

Table 3.4: Module status report format.

*Thermal interlock reset (*NRESET*):* One bit sends a reset signal to the thermal shutdown mechanism; any power supplies in thermal interlock are brought out of thermal interlock.

*Thermal shutdown readout (*TS*):* Each module has a bit that goes high when the thermal interlock engages.

*Readout select (*NSEL*):* Two bits select which module can transmit data back to the TPCC.

*Enable single-module control (*ES*):* The ES bit determines whether the TPCC writes to all modules or only the selected one.

SuRF uses the I^{2}C bus controller of the EZ-USB for the following purposes:

EEPROM *access:* The SuRF board contains a 128-Kb EEPROM. The EEPROM stores the 8051 firmware (see Sec. 3.6.4).

ADC *readout:* Two twelve-channel ADCs digitize the module quantities of interest. The twenty-four quantities that the EZ-USB can read are: module temperature × 4; remote-sensed module ground × 4; remote-sensed module VDD × 4; remote-sensed VDDA × 4; analog supply current × 4; and digital supply current × 4.

Regulator control: The EZ-USB sets the voltage levels of the supply regulators through the AD5242 digital potentiometers in Fig. 3.1.

3.6.3 "Firmware regulation"

In addition to processing the set/get requests from the host computer, the 8051 also optionally corrects for the voltage drop in the module power supply. The firmware continuously digitizes the supply voltages sensed at the modules; it trims the regulator output until the sensed voltage falls within tolerance of the set voltage.

3.6.4 Microprocessor firmware

The controller firmware lives in the `jboard/firm` directory. To build it you need SDCC, the Small Device C Compiler, from `http://sdcc.sourceforge.net`. Typing `make` in the firmware directory will build two files: `firm.ee` and `loader.ihx`. The former is the actual firmware, in a format suitable for bootloading from the on-board EEPROM. The latter is loader firmware for the EZ-USB that allows the host computer to write to the EEPROM through the EZ-USB.

To download the firmware into the EEPROM on the SuRF board, do the following:

- find a Linux machine on which you can be root
- build and load the EZ-USB firmware download module from `http://ezusb2131.sourceforge.net`
- remove the jumper on the I^{2}C SDA line (J?) on the SuRF board to prevent the EZ-USB from loading the current EEPROM contents

- powercycle the SuRF board
- replace the SDA jumper; place a jumper on the EEPROMWPline (JP?) to disable EEPROM write protection
- become root on the host computer
- in the `jboard/firm` directory type `cat loader.ihx > /proc/ezusb/dev0`
- in the same directory type `./eeprom.pl -w -i firm.ee`
- remove the WP jumper and powercycle the SuRF board.

Chapter 4

libsurf, the SuRF board application program interface

libsurf is a C function library written to facilitate programmer interaction with the SuRF board. The library provides two conveniences. The intricacies of dealing with the hardware on the board are abstracted, allowing the programmer to think in natural units (volts, ampères and degrees) and natural operations (setting supply voltages *etc*). The intricacies of the serial bus are also abstracted, allowing the programmer to address modules by their names rather than by USB device and channel number.

Most libsurf functions exist in two forms: board-based and module-based. The board-based functions take a board serial number and channel number as arguments, while the module-based functions take a module serial number as their argument. The bottom-most layer of libsurf keeps track of the location of each board on the bus; unplugging and replugging of boards is transparent. (This doesn't work on windows yet.) Because each SuRF board knows its unique serial number, the board-based functions do not need any configuration files; the module-based functions, however, require a map that says which module is plugged into which channel on which SuRF board. Please read the man page for surf_load_mod_tbl below (Sec. 4.1.10).

4.1 libsurf manual

We list each of the functions in libsurf here. Most functions have board-based and module-based variants. The listing is grouped by functionality.

4.1.1 Library initialization

4.1.1.1 Name

surf_init — initialize libsurf

4.1.1.2 Synopsis

```
int surf_init ();
```

4.1.1.3 Description

`surf_init` initializes the library. Among the initialized data structures are the mutexes for access to the module table and to the SuRF boards, so it is essential that `surf_init` be called before any threads are spawned.

4.1.1.4 Return value

0 on succes, −1 on error.

4.1.2 Data read/write functions — status setters

4.1.2.1 Name

`surf_mod_sel_rd`, `surf_mod_sel_wr`, `surf_mod_unsel_wr`, `surf_brd_sel_rd`, `surf_brd_sel_wr`, `surf_brd_unsel_wr`, `surf_brd_sel_wr_glob` — select a module for reading/writing from/to the TPCC

4.1.2.2 Synopsis

```
int surf_mod_sel_rd (const char *mod);
int surf_mod_sel_wr (const char *mod);
int surf_mod_unsel_wr (const char *mod);

int surf_brd_sel_rd (const char *brd, int chn);
int surf_brd_sel_wr (const char *brd, int chn);
int surf_brd_unsel_wr (const char *brd, int chn);
int surf_brd_sel_wr_glob (const char *brd);
```

4.1.2.3 Description

The `sel_rd` and `sel_wr` functions select a module, either named `mod` or plugged into channel `chn` $(0 \ldots 3)$ on the board with serial number `brd`, for reading/writing from/to the TPCC. If a module is selected for writing, the DCI stream (but not the clock) to the other modules is suppressed. When the module is unselected, or when the board receives a `surf_brd_sel_wr_glob` call, DCI stream transmission to the remaining modules resumes. When a module is selected for reading, its DTO/DTO2 stream is transmitted to the TPCC. `sel_rd` and `sel_wr` calls are not independent; unless global writing is set, the module selected for writing is the same as the module selected for reading, and the last `sel_rd` or `sel_wr` determines which module is selected.

4.1.2.4 Return value

On error `surf_errno` is set and −1 is returned. Otherwise the number of bytes transmitted in the last data transfer over the USB is returned.

4.1.3 Data read/write functions — status getters

4.1.3.1 Name

`surf_mod_get_sel_rd`, `surf_mod_get_sel_wr`, `surf_brd_get_sel_rd`, `surf_brd_get_sel_wr` — find out which module is reading/writing from/to the TPCC

4.1.3.2 Synopsis

```
int  surf_mod_get_sel_rd (const char *mod);
int  surf_mod_get_sel_wr (const char *mod);

int  surf_brd_get_sel_rd (const char *brd, int chn);
int  surf_brd_get_sel_wr (const char *brd, int chn);
```

4.1.3.3 Description

blah blah blah

4.1.3.4 Return value

On error `surf_errno` is set and -1 is returned. If the module indicated in the argument of `get_sel_rd` is selected for reading 1 is returned; if it is not selected for reading 0 is returned. If the module indicated in the argument of `get_sel_wr` is selected for writing 1 is returned; if it is not selected for writing 0 is returned; if global writing is selected on the board to which the module is connected `SURF_GLOB_WRITE` is returned.

4.1.4 Regulator enable functions — status setters

4.1.4.1 Name

`surf_mod_ena`, `surf_mod_end`, `surf_brd_ena`, `surf_brd_end` — enable analog/digital regulators

4.1.4.2 Synopsis

```
int  surf_mod_get_sel_rd (const char *mod);
int  surf_mod_get_sel_wr (const char *mod);

int  surf_brd_get_sel_rd (const char *brd, int chn);
int  surf_brd_get_sel_wr (const char *brd, int chn);
```

4.1.4.3 Description

blah blah blah

4.1.4.4 Return value

blah blah blah

4.1.5 Supply voltage set functions

4.1.5.1 Name

`surf_mod_set_anl_v`, `surf_mod_set_dig_v` — set analog/digital regulator voltages

4.1.5.2 Synopsis

foo

4.1.5.3 Description

These functions are going the way of the dodo.

4.1.6 Sense voltage read functions

4.1.6.1 Name

`surf_mod_get_anl_v`, `surf_mod_get_dig_v` — get analog/digital sense voltages

4.1.6.2 Synopsis

```
double surf_mod_get_anl_v (const char *mod);
double surf_mod_get_dig_v (const char *mod);
```

4.1.6.3 Description

These functions are here to stay. They use the ground sense to calculate their answer.

4.1.6.4 Return value

-1 on error.

4.1.7 Sense voltage ADC count read functions

4.1.7.1 Name

`surf_brd_get_anl_v_adc`, `surf_brd_get_dig_v_adc` — get analog/digital/ground sense voltage ADC counts

4.1.7.2 Synopsis

```
int surf_brd_get_anl_v_adc (const char *brd, int chn);
int surf_brd_get_dig_v_adc (const char *brd, int chn);
int surf_brd_get_gnd_v_adc (const char *brd, int chn);
```

4.1.7.3 Description

foo

4.1.7.4 Return value

-1 on error.

4.1.8 Supply current read functions

4.1.8.1 Name

`surf_mod_get_anl_i`, `surf_mod_get_dig_i` — get analog/digital supply currents

4.1.8.2 Synopsis

```
double surf_mod_get_anl_i (const char *mod);
double surf_mod_get_dig_i (const char *mod);
```

4.1.8.3 Description

These functions are way cool.

4.1.8.4 Return value

−1 on error.

4.1.9 Module temperature read functions

4.1.9.1 Name

`surf_mod_get_temp` - get module NTC temperature

4.1.9.2 Synopsis

```
double surf_mod_get_anl_i (const char *mod);
double surf_mod_get_dig_i (const char *mod);
```

4.1.9.3 Description

These functions are way cool.

4.1.9.4 Return value

−1 on error.

4.1.10 Module table

4.1.10.1 Name

`surf_load_mod_tbl`, `surf_print_mods`, `surf_mod_tbl_get_mod` — load/print the module table; get entries from the module table

4.1.10.2 Synopsis

```
int  surf_load_mod_tbl (const char *file_name);
void surf_print_mods ();
int  surf_mod_tbl_get_mod (char *name, int i);
```

4.1.10.3 Description

The *module table* is a map that says which module is plugged into which channel on which SuRF board. `libsurf` must be given such a map before you can use any module-based `libsurf` functions.

`libsurf` provides the function `surf_load_mod_tbl` to allow the user to add entries to the module table maintained in memory by `libsurf`. `surf_load_mod_tbl(const char *file_name)` loads the module entries from the file with name `file_name` into the module table, superseding existing entries if the file contains entries with the same module serial number. The file should contain lines of the form

board_serial_number channel module_serial_number `<newline>`

Comments are begun by `/*` and terminated by `*/`. The `#` character indicates that the text to the next newline is comment. An example file is given in `libsurf/mod.table`; it looks like this:

```
# $Id: surf.tex,v 1.16 2004/04/01 05:37:02 jmuelmen Exp $

# The format of this file is
# <board SN>  <chan>  <module SN>

# modules connected to 20040LBL006:
200402LBL006 0 M510177
200402LBL006 2 M510176
200402LBL006 3 M510175
200402LBL006 1 M510174

# modules connected to 20040LBL004:
200402LBL004 0 M500174
200402LBL004 2 M500175
200402LBL004 3 M510178
200402LBL004 1 M510170
```

If `libsurf` encounters syntax errors in the file, it will complain; entries read before errors are encountered are added to the module table.

When `libsurf` is compiled as part of AMBuSh, *i.e.* with the `LIBSURF_LOG_TO_AMBUSH` defined, `surf_load_mod_tbl` has an additional effect: AMBuSh attempts to open log files in blah blah blah.

`surf_print_mods` prints the module table currently in `libsurf` memory.

`surf_mod_tbl_get_mod` can be used to get a list of modules currently in

4.1.10.4 Thread safety

`surf_load_mod_tbl`, `surf_print_mods` and `surf_mod_tbl_get_mod` are all thread-safe, as are the functions `libsurf` uses internally when the `surf_mod_`... family of functions needs to resolve a module name.

4.1.10.5 Return value

foo bar

Appendix A

Calibration

This appendix contains the results of the calibrations performed on the various boards. Refer to Sec. 2.6 for a description of the calibration procedure.