Physics and Astronomy Dept.
UBC
SCUBA-2 Project

Stuart Hadfield
3/9/2007 1:21 PM
SC2-ELE-S585-504
Version 1.3

Power-Supply Controller Firmware Description

Revision History:

Rev. 1.1 SH Dec. 22, 2006 Initial realease

Rev. 1.2 BB Feb. 14, 2007 formatting

Rev. 1.3 MA Mar. 8, 2007 added headers and rev. history

Created on: 22 Dec. 2006

Last saved on: 09 March 2007

0. Important Points

- Software version variable at beginning of scuba2ps.c file (2 consecutive places)
- Key parameters to change for different settings located in IO.h
- **Any changes to firmware MUST be followed by recalculation of soft reset address. See soft_reset.* section below.

I. Building Hex File
Open Keil uVision project file. Should see SCUBA2PS.c as target file. Make necessary changes. Select 'project -> rebuild target files'. Building should complete with 0 errors. Hex file can be found in target directory. Refer to Kiel manual.

II. Programming the PSUC:
See document "Programming the PSUC"

III. Version Summary
TBD

1. Introduction

This document provides a (brief) overview of the SCUBA2 PSUC card firmware design. The intended audience is already familiar with general SCUBA2 MCE Subrack design, and PSU/PSUC hardware design.

For a general introduction to the PSUC firmware and hardware, see document "061024 - Development of PSUC Firmware.pdf". Contains a more detailed description of basic PSU/PSUC operation. Note that this document is based on an earlier version of firmware.

The CC-PSUC SPI communication protocol is specified in document "SPI Communications Interface between CC and PS."

Furthermore, consult PSU and PSUC circuit schematics for more detail.

PSUC firmware code (as of Dec. 2006) is included as an appendix. This code is heavily commented and should be viewed in conjunction with its description.

2. General Overview of Firmware

Connectivity between PSU, PSUC, and subrack backplane is summarized in Figure 1 below.

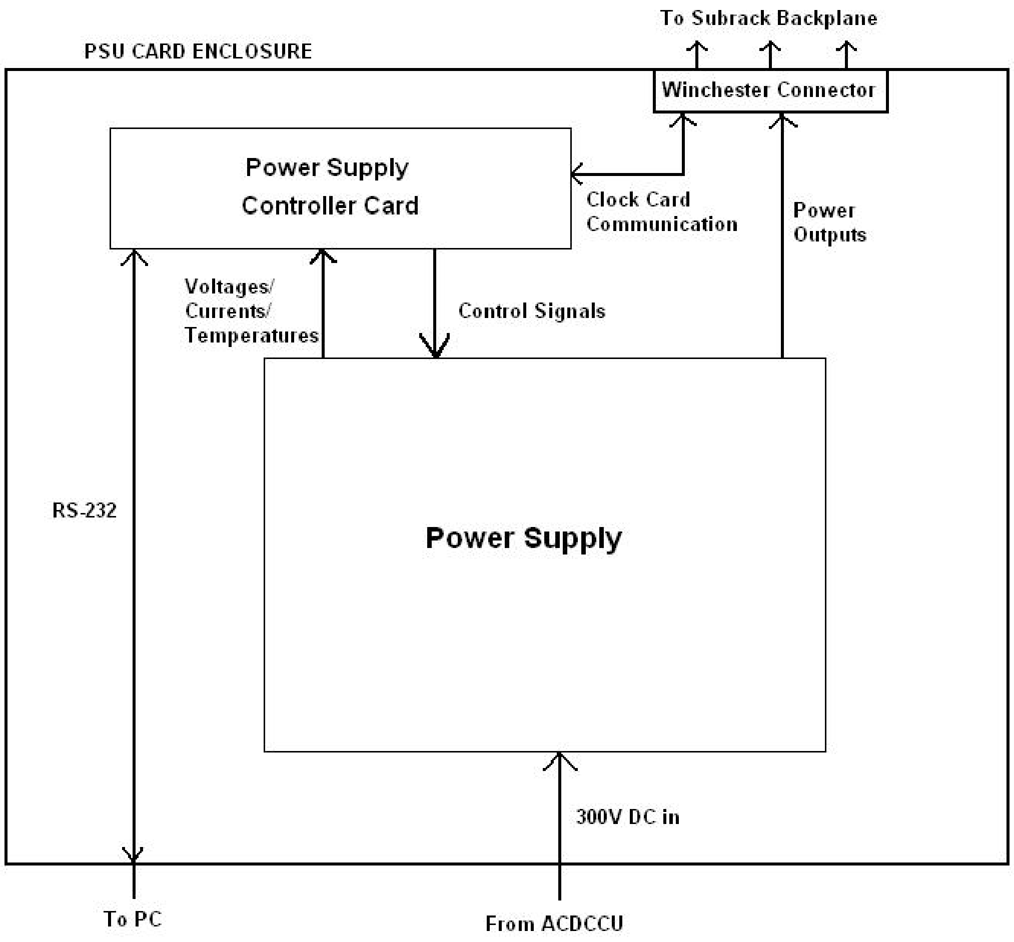


Figure 1: Power Supply Assembly Connectivity Diagram

The PSUC is essentially an AT89 microcontroller with a bunch of interface circuitry such that PSU can be controlled and voltages/temperatures/currents can be read. The microcontroller can directly communicate with the Clock Card over the backplane via the SPI interface. Refer to "SPI Comm…".

Power Supply Data Block

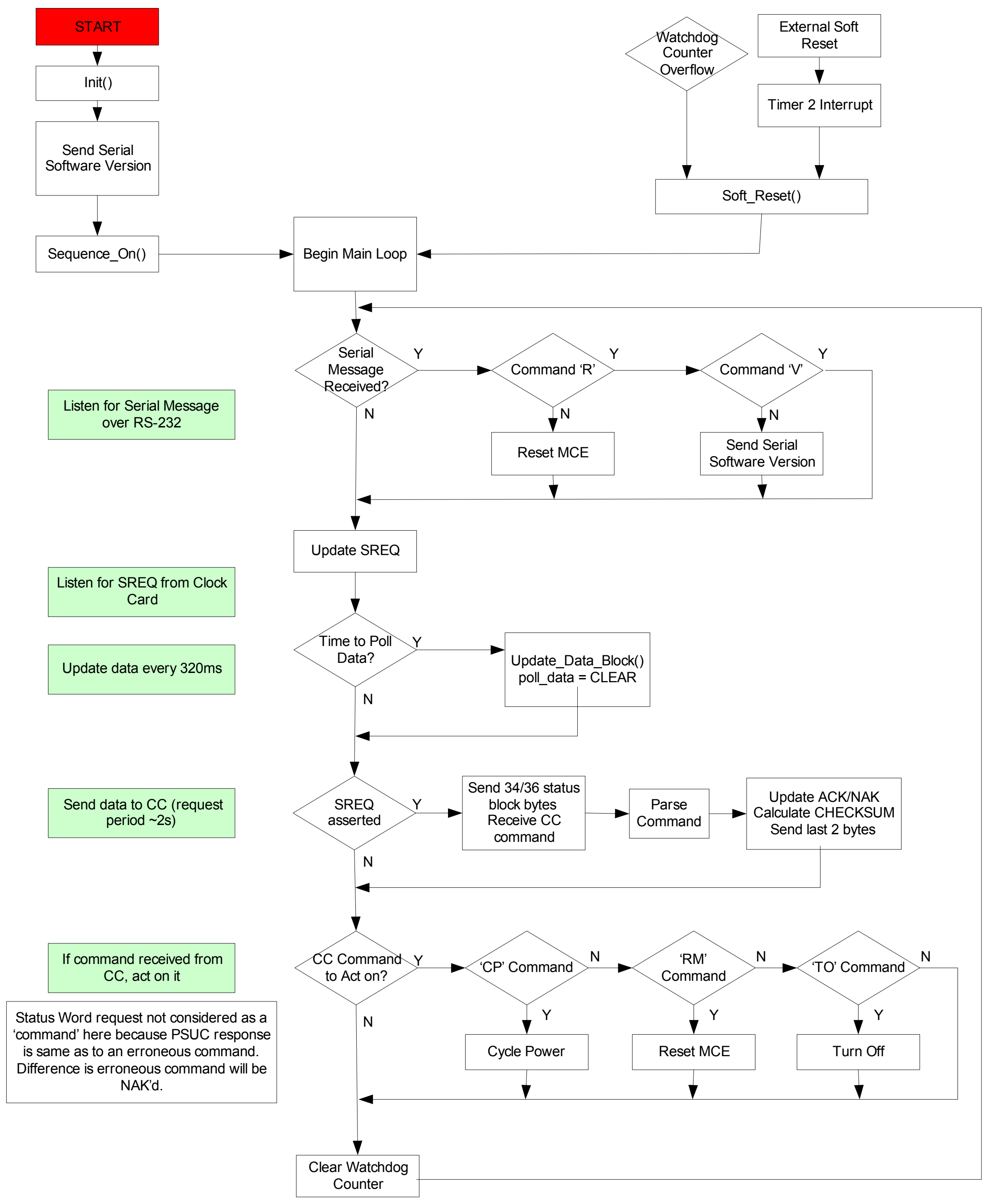
Byte #s	Item	Bytes	Description
0	Silicon ID	4	32 least sig bits of 48 bit ID
4	Software Version	1	Encoded as hex byte. 0xYZ = version Y.Z
5	Fan1 Tachometer	1	Currently not used
6	Fan2 Tachometer	1	Currently not used
7	PSU Temperature 1	1	8 bit two's compliment (1 deg. increments)

8	PSU Temperature 2	1	8 bit two's compliment (1 deg. increments)
9	PSU Box Temperature 3	1	8 bit two's compliment (1 deg. increments)
10	ADC Offset	2	Digitized Ground (bipolar, 2 bits/deg C)
12	Supply Voltage 1	2	+Vcore Supply – unipolar scaled to 73.2%
14	Supply Voltage 2	2	+Vlvd Supply – unipolar scaled to 73.2%
16	Supply Voltage 3	2	+Vah Supply – unipolar scaled to 73.2%
18	Supply Voltage 4	2	+Va Supply – unipolar scaled to 73.2%
20	Supply Voltage 5	2	-Va Supply – unipolar scaled to 73.2%
22	Supply Current 1	2	Current +Vcore– unipolar scaled to 61%
24	Supply Current 2	2	Current +Vlvd– unipolar scaled to 61%
26	Supply Current 3	2	Current +Vah– unipolar scaled to 61%
28	Supply Current 4	2	Current +Va– unipolar scaled to 61%
30	Supply Current 5	2	Current -Va– unipolar scaled to 61%
32	Status Word	2	For future expansion
34	ACK/NAK	1	ACK if command correct/NAK otherwise
35	Check Digit	1	2's compliment of sum of all other bytes
	Total	36	36 x 8 = 288 clocks on the SPI Interface

All firmware was written in C (plus one assembly file) and compiled/built (to hex file) using Kiel uVision3. The microcontroller itself was programmed using Atmel FLIP 2.46 utility. Project file, hex file, and all code stored in SCUBA2 CVS repository in PSUC card directory.

The high level operation of the PSUC firmware is given diagrammatically in Figure 2 below.

SCUBAPS.c - High Level Flow Chart - PSUC Firmware Version 2.3


START
Init()
Send Serial Software Version
Sequence_On()
Begin Main Loop
Watchdog Counter Overflow
External Soft Reset
Timer 2 Interrupt
Soft_Reset()
Serial Message Received?
Y
N
Command 'R'
Y
N
Reset MCE
Command 'V'
Y
N
Send Serial Software Version
Listen for Serial Message over RS-232
Update SREQ
Listen for SREQ from Clock Card
Time to Poll Data?
Y
N
Update_Data_Block() poll_data = CLEAR
Update data every 320ms
SREQ asserted
Y
N
Send 34/36 status block bytes Receive CC command
Parse Command
Update ACK/NAK Calculate CHECKSUM Send last 2 bytes
Send data to CC (request period ~2s)
CC Command to Act on?
Y
N
'CP' Command
N
Y
Cycle Power
'RM' Command
N
Y
Reset MCE
'TO' Command
N
Y
Turn Off
If command received from CC, act on it
Status Word request not considered as a 'command' here because PSUC response is same as to an erroneous command. Difference is erroneous command will be NAK'd.
Clear Watchdog Counter

<u>Summary of high level operation:</u>

On initial power-up / hard reset[1], the init() function is run first. This routine sets all internal registers, initializes software variables, checks which external devices are connected (DS18S20), and performs all '1-Time only' tasks (i.e. getting PSU silicon ID). Next, the firmware version is dumped out the serial port (whether or not anything is listening) and power supply outputs are sequenced on. Next begins the main loop.

The main loop operates as follows:

First it checks if a serial message (command) has been received (very rare event). If so, it acts on the two possible commands, Reset_MCE or a Sodtware Version request. If not, it proceeds to next step, updating SREQ. The status of this input indicates a pending CC status request.

Next, the status of the 'Time to Poll Data' indicator bit is checked[2] (see 'Timer Operation' below). If this bit is set, Update_Data_Block() is called. This function sequentially polls the PSUC ADCs (current/voltage) and PSU DS18S20s (temperature), and calculates a partial checksum from this data[3].

Next, the (updated) status block is sent to the CC via SPI <u>if</u> SREQ had been set earlier. The delay between checking for a status block request and actually sending it is to allow for data to be updated (in the rare case where timer for updating has expired at same time request is issued). This ensures the CC is always sent the most current data.
If the CC has requested the status block, the first 34 (of 36) bytes will be sent. While these bytes are being sent, the PSUC receives three commands from the CC[4]. These commands are then parsed. If a valid command has been received, the ACK/NAK byte is set to ACK; else to NAK. The final checksum is then calculated and the final two bytes (ACK/NAK and Checksum) are sent.

Next, the PSUC acts on the CC command (if valid command received). 'Reset MCE', 'Turn-Off,' and 'Cycle Power' commands are as named. No action is taken for default 'Status-Block Request' command (program continues through the loop…).
Similarly, no action is taken for an invalid command. However in this case the invalidity will have been indicated to the CC via the ACK/NAK byte and it is assumed the CC will re-send its command as necessary.

Next is minor loop maintenance. The watchdog counter is cleared, with the idea being a typical run through this loop takes much less time than it does for the watchdog counter

[1] Hard reset is when reset button on PSUC is pressed (or PSA is power cycled). Soft reset is when button on PSA front panel is pressed (reset program counter to beginning of loop only).
[2] 'Poll data' bit set periodically every 320ms
[3] This function cannot calculate the FINAL checksum value as it depends on the ACK/NAK byte…
[4] Refer to "SPI Communication.." document.

to overflow[5]. Thus this counter should never overflow unless the firmware is hung midloop somewhere (unlikely…). Loop then returns to beginning (infinite loop).

There are two cases where the program can break out of this loop. If the watchdog timer expires, an interrupt is triggered and soft_reset() is called immeadiately. Alternatively, if the reset button on the PSA front panel is pushed, a timer2 interrupt is triggered and soft_reset() is again called immeadiately. The soft_reset() function resets the program counter to the beginning of the main loop (***NOT to the beginning of the program, so JMP address is NOT 0x0000). The advantage of this over the built-in (hard) reset is that the outputs stay fixed in the former but not the latter, preventing inadvertent switching of key signals such as BRST and nPSU_ON.

Timer Operation: There are three timers on the AT89 microcontroller.

Timer 0: This timer is set to always run with interrupt occurring every 32ms. On every interrupt, it checks if the watchdog count has overflowed, and if so calls soft_reset(). In internal variable is counted such that every 320ms the watchdog count is incremented and the ‘Time to Poll Data’ flag is set.

Timer 1: This timer is set to overflow every 5ms when running. This timer is used solely for the wait_time() function was allows a variable (determined by function argument) time period to pass when needed.

Timer 2: Due to last minute design inclusion of external soft reset button, this timer is used exclusively as an external interrupt. (no actual external interrupt pin was available….). Timer is set to 1 value less than overflow, set to always run, and set to increment only when soft reset front panel button pressed. On interrupt, soft_reset() is called. Correct buffer values are automatically reloaded.

3. Functional Description

File(s)	Description
SCUBA2PS.c, SCUBA2PS.h	Main Program
IO.h	Input/Output Settings and Global Variables
DS18S20.c, DS18S20.h	DS18S20 ID/Temperature Sensor Interface
MAX1271.c	MAX1271 ADC Interface
SOFT_RESET.a, SOFT_RESET.h	Soft Reset Assembly Code

Table 1. Source Code File Summary

[5] Watchdog set to overflow after 5.12 seconds (without a clear). Typical p

The following is a list of functions implemented in PSUC firmware. The code is heavily commented and is the main reference. See Appendix.

Return Type	Function Name	Arguments	Description / Notes
void	init	void	Initializes hardware and software variables
void	sequence_on	void	Powers On MCE
void	sequence_off	void	Powers Off MCE
void	reset_MCE	void	Resets MCE
void	cycle_power	void	Cycle MCE Power
void	send_psu_data_block	void	Send PSU Datablock to CC via SPI interface
void	wait_time	unsigned char	Waits arg*5ms (millisecond wait timer)
void	wait_time_x2us_plus3	unsigned char	Waits arg*2us + 3us (microsecond wait timer)
void	snd_msg	char*	Sends message pointed to by arg over RS-232
void	update_data_block	void	Updates voltage/current/temperature readings
void	check_digit	void	Calculates partial checksum (before ACK/NAK added)
void	parse_command	void	Reads CC command rcv'd from first 6 bytes received from SPI transaction
bit	commands_match	char*, char*, char*	Returns true if the three commands rcv'd match
bit	command_valid	char*	Returns true if command rcv'd is a valid command

Table 2. SCUBA2PS.*

SCUBA2ps.c contains the main loop and the most general functions. Main loop operation is detailed in flowchart above. Refer to code.

Return Type	Function Name	Arguments	Description / Notes
"Public" Functions			These three functions only should be called explicitly from the main program
bit	ds_initialize	char	Initializes DS18S20, checks if present
void	ds_get_4byte_id	char, char*	Reads Silicon ID, sets target value
	ds_get_temperature	char, char*	Reads temperature from DS memory, sets target value
"Private" Functions			These functions inplicitly implement the 1-Wire Bus protocol The following functions declared 'static' as they should only be
void	ds_convert_T	void	Start temperature conversion
bit	ds_reset	void	Generates reset pulse, returns 1 if DS detected
void	ds_write_byte	unsigned char	Write target byte to DS
unsigned char	ds_read_byte	void	Read byte from DS
void	ds_write_bit	bit	Write given bit to DS (low level)
bit	ds_read_bit	void	Read and return bit from DS (low level)
bit	read_bus	void	Read bus state (Physical level)

Table 3. DS18S20.*

This file contains functions for interfacing with the DS18S20 temperature sensors via the '1-Wire Bus' Protocol. Refer to "1-Wire Communication Through Software" document[6] and code. Firmware implements protocol's strict timing requirements as described in aforementioned document.

The only three functions which should ever be called externally (from the main file) are initialize, get ID, and get Temperature. Temperature is stored directly in degrees Celsius with a one bit per degree correspondence.

The DS is read by issuing a READ SCRATCHPAD command which triggers the sending of 8 databytes. Only the first two bytes correspond to temperature, and the implemented code terminates the transaction after these two bytes have been received. This is done to speed up the DS transaction as it is the slowest part of the update_data_block() function. One of the ignored bytes is a CRC check byte. This could later be used to check data integrity, but would further slow down this process.

**Important: This code was separated from the main program file to allow for reusability. Unfortunately, input lines are accessed on our microcontroller using custom types 'sbit' and 'sfr' which the Kiel uVision compiler does NOT allow to be passed into functions / pointed to. Thus, in order to reuse this code with different DS18S20s connected to different input pins, a 'mask' byte is passed into the functions which indicates which output to use[7].

**Important: Current code implementation not designed for parasitic power mode. It is likely future ECO will add DS18S20 temperature sensors into PSU box, powered parasitically. This will require *minor* code modifications[8].

Return Type	**Function Name**	**Arguments**	**Description / Notes**
void	read_adc	char, char, bit, char*	Reads ADC and assigns value to char*. First two chars indicate read mode and channel, bit indicates voltage or current ADC.

Table 4. MAX12171.c

MAX1271.c contains a single function for reading a specified ADC channel. The first two char arguments specify the channel to read and the mode[9] to read with. The bit argument selects either the Current or Voltage ADC, and the place to store the read value

[6] from Dallas Semiconductor
[7] Fortunately this trick is possible because all DS18S20s are on the same input (8-bit) port
[8] Mainly, a couple places where software waits for response from DS will have to be replaced with wait_time()…
[9] Mode used is uni-polar, full-scale for all channels except the ground channel which uses bipolar, half-scale.

is specified by the char pointer. The MAX1271 communication protocol is specified in detail in the datasheet.

The total number of clock cycles required for a complete transaction with the MAX is 25. As this number is not divisible by eight, it prevents efficient use of the AT89 microcontroller's built in SPI functionality, which operates on the byte level. Thus the MAX transaction is manually clocked at the bit level. This is done in code by disabling the SPI interface, manually writing to and reading from the MAX, and then re-enabling SPI. Refer to code and datasheet.

Note: A large part of the 25 clock cycle ADC transaction is waiting while the acquisition and conversion processes takes place[10]. The ADC sends out a pulse on the SSTRB line when acquisition is complete and conversion begins. As the 6th (of 12) data bits is being clocked out from the MAX, the next control word can begin to be clocked in. Thus the transactions can be effectively pipelined (within the same ADC), reducing time to 18 clock cycles per transaction.

PSUC Rev. F did not contain AT89 connections to either ADC's SSTRB line. Adding these connections was recommended by the author and implemented in Rev. G. Due to time/priority constraints, firmware could not be updated in time to make use of these signal lines. Future firmware updates could use these signals to improve code efficiency (replace counted clock cycles with 'wait for SSTRB' where applicable) and implement pipelined reading. However, current (non-pipelined) version takes about 80 microseconds per ADC channel and about 1 millisecond to read all 11 channels. This is much faster than the time is takes to read from the DS18S20 (which takes about 6.5 ms due to slow temperature conversion), so improving the efficiency of the MAX code will only marginally improve the speed of the update_data_block() function.

All voltage channels (except ground[11]) are routed through a low pass filter and non-inverting amplifier, scaled (ideally) to 73.2% of ADC full range. The MAX1271 maximum readable input voltage is 4.096V, so this scaling corresponds to a 3.0V input. This implies that if all voltages are at their nominal level, the ADC should report out at 73.2% of its maximal value (i.e. should report = 0.732 * 0xFFF = 0xBB6)[12] on each channel. A grounded ADC input is also read to measure any ground offset. This channel is the only channel read in bi-polar mode (i.e. read in negative voltage range also).

The PSU output currents are measured as follows: Each PSU output voltage signal runs across a very small 'shunt' resistance[13]. The PSUC then receives a voltage signal from both sides of this resistance, which is fed into a differential amplifier followed by a non-inverting scaling amplifier. Current across the shunts is thus inferred from the voltage

[10] 'Track and Hold'
[11] Ground reported in twos compliment format, 2 bits per deg. C
[12] MAX1271 has 12-bit output resolution so maximum output is 0x0FFF.
[13] Refer to page 4 of PSU schematic (Appendix 5.2)

difference and known resistances values. Current measurements are scaled to 2.5V, or equivalently 61% of full range (i.e. at nominal current levels, ADCs will output 0.61 * 0xFFF = 0x9C2 on each channel).

Return Type	Function Name	Arguments	Description / Notes
void	soft_reset	void	Resets program counter to beginning of main loop (does not affect state of outputs)

Table 5. SOFT_RESET.*

As the soft_reset function is so small is included here:

```
;----------------------------
?PR?SOFT_RESET  SEGMENT CODE
RSEG ?PR?SOFT_RESET
USING 0
                                        ; C prototype:  void soft_reset (void);
PUBLIC soft_reset
     soft_reset: POP  ACC               ; pop return address
          POP  ACC
          CLR  A                        ; push 0 as new
          ADD A, #0xD4                  ; lower order adress byte
          PUSH ACC                      ; return address to stack
          CLR A
          ADD A, #0x02                  ; higher order adress byte
          PUSH ACC
          RETI                          ; execute return of interrupt
          END
```

The point of this code is to reset the program counter to the beginning of the main loop. The reset jumps here, and not to the beginning of the program (address 0x0000) in order to maintain the state of the outputs, specifically not to cause an inadvertent reset or power glitch. The built in reset functions (in the AT89 library) are intrinsically hard resets; they cause a jump to the beginning of the program and cause outputs to momentary return to default values.

**This code is implemented in assembly solely because it must be able to function correctly when being called after an interrupt. The watchdog timer and soft reset button are both pointless if they are not acted on immediately. The original implementation of this function was done in C by type casting the reset vector address to a function pointer, this but failed to work correctly because it did not clear the AT89 interrupt system. Thus the above assembly code was used.

The code works as follows: After an interrupt has been triggered the AT89 will store the current program counter address in the stack and jump to the interrupt address. After this jump, the above code is executed, which pops the return address of the stack, pushes the desired address back on, and then executes return from the interrupt. The program thus returns ('resets') to the desired point in the program.

This code was implemented in a separate assembly file. A header file simply declares the C style header of this function so that it can be called from the main C program. Inline assembly inclusion was originally attempted but abandoned due to compiler issues.

******<u>Important</u>**: In the above code, the return address is 0x02D4. This address is **<u>not</u>** fixed and must be recalculated **every** time **any** part of the PSUC firmware has been modified. This is done be compiling the modified firmware in Kiel, and entering the debugging mode. Set a break point at the beginning of the main loop (desired jump point) and select 'Run to Break Point'. Open disassembly window. Cursor should be at break point, which gives the desired address. This address must then be changed in the soft_reset.a file.

**The program is very sensitive to this reset address value and an error here could cause unpredictable consequences.

4. Future Versions / Features

- EEPROM Support (needs specs + new commands..)
- More SPI/Serial commands

5. Other

Fans omitted from inside PSA. Thus firmware design, specifically timer allocation, was optimized for not measuring fan speed.

5. Appendix

PSUC Firmware C Code: See \scuba2_repository\cards\psuc_card\PSUC Firmware

Code to be included here if necessary.