Serial Command Interface
SCI_v1.6f

1 Overview

The Serial Command Interface is a serial communications protocol, which efficiently handles the control of the regulator. The regulator unit is acting like a slave or stand-alone unit. Each command from the master unit results in a response from the regulator.
At startup of the regulator, the values in the internal EEPROM memory will be copied into the runtime registers. The master can then change the EEPROM memory by first writing to the runtime registers and then issue the RW command to write runtime registers to EEPROM. There is no need to issue a RW command after each register change. Change all registers first, and then write to EEPROM if needed.
All communication is by ASCII chars, which will be possible to stream to file or terminal for debugging, if needed.

Features

- RS232 serial interface, 115200 baud, 1 start, 8 bit, 1 stop bit, no parity, no handshake (no flow control).
- Slave and/or stand-alone unit
- All communication with ASCII char
- EEPROM register holding startup regulator values
- Advanced PID regulator system, where all parameters can be adjusted
- Possibility to get the actual temperature values
- Possibility to get runtime register values
- Possibility to upgrade the firmware by using boot loader mode
- IEEE754 transfer mode implemented, to improve data accuracy
- Saving LOGG data every 20min. Can be used to check voltage/current/temp values.

2 Command

The command set is built up by (with an example):

start char	*command*	*data*	*stop char*
$	R41=	23.5	<CR>

When the regulator is ready to receive next command, we get following ASCII chars: 'CR' 'LF' '>' 'SP' (HEX 0D 0A 3E 20).
Each character sent to the regulator is echoed back from the regulator, and when the end of the command (CR) has been sent, the regulator responds with ASCII chars: 'CR' 'LF' followed by response as the table below. Note that after each command and response, following string is sent in ASCII chars: 'CR' 'LF' '>' 'SP' (HEX 0D 0A 3E 20).
If a sent command is unknown, the regulator responds with a question mark followed by the sent command. Note that the commands are case sensitive.

Command	**Description**	**Regulator Response**
$Q	Clear RUN flag	<Stop>
$W	Set Run flag	<Run>
$V	Show current software version	<version string>
$v	Show current software version and interface version	<version string><interface version>
$S	Get status flag	<flag1><alarm1><alarm2>
$SC	Clear status flag	<flag1><alarm1><alarm2>
$RW	Write regulator register values to EEPROM	
$RR	Display regulator register	<parameter list>
$Rxx=data	Write data (float or int) to register xx	If int <Downloaded data> If float <no response>

$Rxx?	Read data (float or int) from register xx	ex. +2.000e+01
$RNxx=data	Write float data in IEEE754 type to register xx	
$RNxx?	Read float data in IEEE754 type from register xx	ex. A1A8FCBA
$Ax	Continuous log of values, where X is 1..7	continues data
$B	Enter BOOT LOADER mode (described in a boot loader document)	Se bootloader document
$BC	Reboot the board	boot text
$LI	Get board info and ID name	ID string of data
$LD	Display logg data (max/min/avr)	<num> VIN, CURR, T1, T4
$LL	Load logg data from eeprom to ram	<num> VIN, CURR, T1, T4
$LC	Clear logg data in eeprom, use with care	<num> VIN, CURR, T1, T4
<CR>	Perform last command	As previous command
	Unknown command	?<command>

IEEE754 is handled as a 32 bit hex value, displayed as 8 char with the MSB char first. This helps us to move values more accurate data to/from controller unit to/from a PC.

3 Register

The following registers are defined for the moment, but will change prior to the software release.

Register	Default value	R/W	Type	Description
0	20.0	RW	float/IEEE	Set Point - (fgTref) Main temperature reference
1	20.0	RW	float/IEEE	PID – (Kp) P constant
2	2.0	RW	float/IEEE	PID – (Ki) I constant
3	5.0	RW	float/IEEE	PID – (Kd) D constant
4	2.0	RW	float/IEEE	PID – (KLP_A) Low pass filter A (+value)
5	3.0	RW	float/IEEE	PID – (KLP_B) Low pass filter B (+value)
6	100.0	RW	float/IEEE	MAIN – (TcLimit) Limit the Tc signal (0..100)
7	3.0	RW	float/IEEE	MAIN – (TcDeadBand) Dead band of Tc signal (0..100)
8	100.0	RW	float/IEEE	PID – (iLim) Limit I value (0..100)
9	0.05	R	float/IEEE	MAIN – (Ts) Sample rate 1/Hz
10	1.0	RW	float/IEEE	MAIN – (Cool Gain) Gain of cool part of Tc signal (+value)
11	1.0	RW	float/IEEE	MAIN – (Heat Gain) Gain of heat part of Tc signal (+value)
12	0.1	RW	float/IEEE	PID – (Decay) Decay of I and low pass filter part (+value)
13	128	RW	int	REGULATOR MODE – Control register
14	5.0	RW	float/IEEE	ON/OFF Dead band (+value)
15	5.0	RW	float/IEEE	ON/OFF Hysteresis (+value)
16	0	RW	int	FAN 1 Mode select
17	20.0	RW	float/IEEE	FAN 1 Set temperature
18	8.0	RW	float/IEEE	FAN 1 Dead band (+value)
19	4.0	RW	float/IEEE	FAN 1 Low speed hysteresis (+value)
20	2.0	RW	float/IEEE	FAN 1 High speed hysteresis (+value)
21	30.0	RW	float/IEEE	FAN 1 Low Speed voltage (0..30)
22	30.0	RW	float/IEEE	FAN 1 High Speed voltage (0..30)
23	0	RW	int	FAN 2 Mode select
24	20.0	RW	float/IEEE	FAN 2 Set temperature
25	8.0	RW	float/IEEE	FAN 2 Dead band (+value)
26	4.0	RW	float/IEEE	FAN 2 Low speed hysteresis (+value)
27	2.0	RW	float/IEEE	FAN 2 High speed hysteresis (+value)

28	30.0	RW	float/IEEE	FAN 2 Low Speed voltage (0..30)
29	30.0	RW	float/IEEE	FAN 2 High Speed voltage (0..30)
30	0	RW	float/IEEE	POT input - AD offset
31	0	RW	float/IEEE	POT input - offset
32	1	RW	float/IEEE	POT input - gain
33	0	RW	float/IEEE	Expansion port AD out – offset
34	1	RW	float/IEEE	Expansion port AD out - gain
35	1.0	RW	float/IEEE	Temp 1 gain
36	0.0	RW	float/IEEE	Temp 1 offset
37	1.0	RW	float/IEEE	Temp 2 gain
38	0.0	RW	float/IEEE	Temp 2 offset
39	1.0	RW	float/IEEE	Temp 3 gain
40	0.0	RW	float/IEEE	Temp 3 offset
41	1.0	RW	float/IEEE	Temp FET gain
42	0.0	RW	float/IEEE	Temp FET offset
43	-	RW	int	Temp 1 digital pot offset (0..255)
44	-	RW	int	Temp 1 digital pot gain (0..255)
45	30.0	RW	float/IEEE	ALARM level voltage high
46	10.0	RW	float/IEEE	ALARM level voltage low
47	15.0	RW	float/IEEE	ALARM level main current high
48	0.1	RW	float/IEEE	ALARM level main current low
49	2.0	RW	float/IEEE	ALARM level FAN 1 current high
50	0.1	RW	float/IEEE	ALARM level FAN 1 current low
51	2.0	RW	float/IEEE	ALARM level FAN 2 current high
52	0.1	RW	float/IEEE	ALARM level FAN 2 current low
53	13.0	RW	float/IEEE	ALARM level internal 12v high
54	7.0	RW	float/IEEE	ALARM level internal 12v low
55	12	RW	int	Temp1 mode
56	4	RW	int	Temp2 mode
57	4	RW	int	Temp3 mode
58	4	RW	int	Temp4 mode
59	1.396917e-03	RW	float/IEEE	Temp1 Steinhart coeff A
60	2.378257e-04	RW	float/IEEE	Temp1 Steinhart coeff B
61	9.372652e-08	RW	float/IEEE	Temp1 Steinhart coeff C
62	1.396917e-03	RW	float/IEEE	Temp2 Steinhart coeff A
63	2.378257e-05	RW	float/IEEE	Temp2 Steinhart coeff B
64	9.372652e-07	RW	float/IEEE	Temp2 Steinhart coeff C
65	1.396917e-03	RW	float/IEEE	Temp3 Steinhart coeff A
66	2.378257e-05	RW	float/IEEE	Temp3 Steinhart coeff B
67	9.372652e-07	RW	float/IEEE	Temp3 Steinhart coeff C
68	6.843508e-03	RW	float/IEEE	Temp4 Steinhart coeff A
69	2.895852e-04	RW	float/IEEE	Temp4 Steinhart coeff B
70	-8.177021e-08	RW	float/IEEE	Temp4 Steinhart coeff C
71	80.0	RW	float/IEEE	ALARM Temp 1 high
72	-40.0	RW	float/IEEE	ALARM Temp 1 low
73	50.0	RW	float/IEEE	ALARM Temp 2 high
74	-10.0	RW	float/IEEE	ALARM Temp 2 low
75	50.0	RW	float/IEEE	ALARM Temp 3 high
76	-10.0	RW	float/IEEE	ALARM Temp 3 low

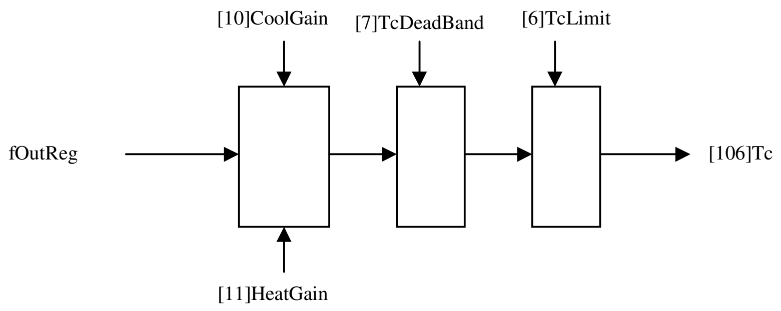
77	60.0	RW	float/IEEE	ALARM Temp 4 high
78	-10.0	RW	float/IEEE	ALARM Temp 4 low
79	759.4	RW	float/IEEE	Temp1 Steinhart resistance value H
80	3057.7	RW	float/IEEE	Temp1 Steinhart resistance value M
81	29875.8	RW	float/IEEE	Temp1 Steinhart resistance value L
82	759.4	RW	float/IEEE	Temp2 Steinhart resistance value H
83	3057.7	RW	float/IEEE	Temp2 Steinhart resistance value M
84	29875.8	RW	float/IEEE	Temp2 Steinhart resistance value L
85	759.4	RW	float/IEEE	Temp3 Steinhart resistance value H
86	3057.7	RW	float/IEEE	Temp3 Steinhart resistance value M
87	29875.8	RW	float/IEEE	Temp3 Steinhart resistance value L
88	2965.14	RW	float/IEEE	Temp4 Steinhart resistance value H
89	28836.8	RW	float/IEEE	Temp4 Steinhart resistance value M
90	78219	RW	float/IEEE	Temp4 Steinhart resistance value L
91	351	RW	uint	Alarm enable bits low
92	255	RW	uint	Alarm enable bits high
93	8.0	RW	float/IEEE	Setpoint 2, when in test loop mode
94	300	RW	unit	TimeHigh, when in loop mode (cycles)
95	200	RW	unit	TimeLow, when in loop mode (cycles)
96	65532	RW	uint	Sensor Alarm Mask, select warning or alarm of sensors
99	-	R	uint	Regulator event count. Increments one time each regulator event.
100	-	R	float/IEEE	Temp 1 value (AN5)
101	-	R	float/IEEE	Temp 2 value (AN6)
102	-	R	float/IEEE	Temp 3 value (AN7)
103	-	R	float/IEEE	Temp FET value (AN8)
104	-	R	float/IEEE	Temp POT reference (AN0)
105	-	R	float/IEEE	TRef
106	-	R	float/IEEE	MAIN (Tc) output value (-100..+100)
107	-	R	float/IEEE	FAN 1 output value (0..100)
108	-	R	float/IEEE	FAN 2 output value (0..100)
110	-	R	float/IEEE	PID – Ta
111	-	R	float/IEEE	PID – Te
112	-	R	float/IEEE	PID – Tp
113	-	R	float/IEEE	PID – Ti
114	-	R	float/IEEE	PID – Td
117	-	R	float/IEEE	PID – TLP_A
118	-	R	float/IEEE	PID – TLP_B
122	-	R	int	ON/OFF – runtime state
123	-	R	float/IEEE	ON/OFF – runtime max
124	-	R	float/IEEE	ON/OFF – runtime min
125	-	R	int	FAN1 – runtime state
126	-	R	float/IEEE	FAN1 – runtime max
127	-	R	float/IEEE	FAN1 – runtime min
128	-	R	int	FAN2 – runtime state
129	-	R	float/IEEE	FAN2 - runtime max
130	-	R	float/IEEE	FAN2 – runtime min
150	-	R	float/IEEE	(AN1) Input voltage
151	-	R	float/IEEE	(AN10) Internal 12v
152	-	R	float/IEEE	(AN9) Main current

153	-	R	float/IEEE	(AN11) FAN1 current
154	-	R	float/IEEE	(AN4) FAN2 current
155	-	RW	float/IEEE	FAN1 and FAN2 internal gain value (do not use)

NOTE:
- The master is responsible for checking that the value is within the range of the parameter. If a value is outside recommended value, unpredictable behavior may occur. If the value is not decodable, zero is inserted.
- Float values will only display six decimals. Example 4.123456, -3.878667, +4.887667, or width E value, +1.23456e-04 etc.
- IEEE mode. Float value sent by 8 char of hex byte.
- Only Steinhart coeff A, B, C is used in the regulator, but the Steinhart resistance parameter is used by the PC program to be able to calculate right set points.
- All parameters are possible to change on the fly (when regulator is active) and will take effect directly. It is also possible to change all parameters and then save to internal EEPROM memory. This gives the control unit the possibility to work online with a master unit or as a stand-alone unit.

4 Main regulator

The main regulator is controlling the main output, which is a H-bridge output. The output is regulating the power and current direction through the peltier modules connected. You can set the output control in different modes, PID, POWER and ON/OFF control mode.



Issue command “$W” to set RUN flag, and “$Q” to clear the regulator flag. To store the RUN flag into EEPROM , issue command “$RW” after issuing “$W” and the current settings on all registers and flags are stored in the EEPROM. From now on and after each power up, these are the values that are in use.

We have a delay of 3 seconds, which is in effect after each power up, and after an error is cleared.

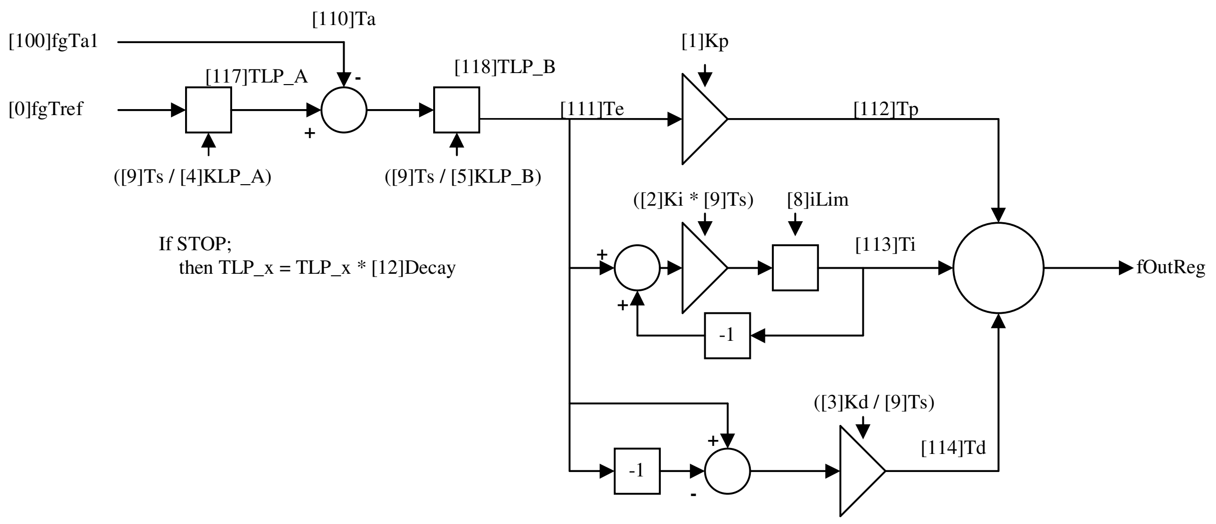
We have a Watchdog activated to catch any fault in the system.

Parameters:
[0] Set point – to set the temperature to regulate towards
[6] Limit Tc – is to limit the output signal to a max value. This is always in effect. Good to use if we only need a max effect on the output.
[7] Dead band of Tc signal – is limiting the signal around zero value. Good to adjust if we do not like fast switching from one voltage/current direction to the other direction. This helps to save the life of the peltier modules.
[9] Sample rate – 1/0.05 = 20Hz. This is a fix sample rate. Contact Supercool if there is a need for a different sample rate.
[10] Cool gain – to adjust the cool gain part. Normally this is 1.0
[11] Heat gain – to adjust the heat gain part. Normally this is 1.0
[13] Regulator Mode – use to switch between the different regulator modes.

0 = no regulator mode
1 = POWER mode
2 = ON/OFF mode
3 = P regulator mode
4 = PI regulator mode
5 = PD regulator mode
6 = PID regulator mode

4.1 PID regulator mode

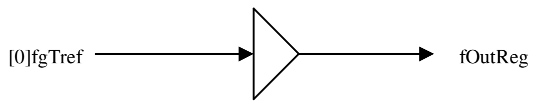
The PID regulator is controlling the main output when selected. This mode is possible to use in P, PI, PD, or PID mode. Of course it is possible to use PID mode and only set P, I or D constants to zero, but the possibility to change mode without changing the constants helps to trim and adjust the values for optimal use.



Parameters:
[1] P constant
[2] I constant
[3] D constant
[4] Low pass filter A
[5] Low pass filter B
[8] Limit I signal
[12] Decay of I signal, and low pass filter parts when the regulator is switched off.

4.2 Power regulator mode

This mode is normally only used when to check that the load is working, and when to adjust the COOL [10] and HEAT [11] gain parameters. The internal set point [0] is used to set the output power needed. This means that we enter a control value from –100 to +100 and the main output will set this on the output. Note that the dead band [7] should be zero, and limit [6] should be 100, if the full range is to be used.



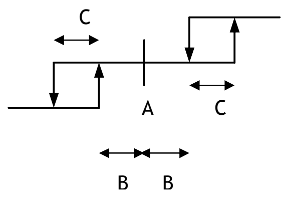
4.3 ON/OFF regulator mode

ON/OFF Regulator is a regulator mode used to simulate an old regulator with a set point, dead band and hysteresis around it.

Temperature control

(ON/OFF)

[0] A	Set point	*-50 … 100°C*
[14] B	Dead band	0 … 50°C
[15] C	Hysteresis	0 … 10°C



5 FAN regulator

The FAN is possible to set in different modes. The RUN flag “$W” must be activated for the FAN regulator to run, and that there are no errors or Alarms activated.

[16] FAN Mode selection register
- 0 = always off
- 1 = always on
- 2 = COOL mode
- 3 = HEAT mode
- 4 = COOL/HEAT mode
- 5 = FAN regulator mode

5.1 Always off

The FAN is always off.

5.2 Always on

The FAN is always on if the RUN flag “$W” is activated and there is no error.

5.3 COOL mode

In this mode the FAN is on when runtime register [106]main output is negative. The FAN is operating at the speed set in the register [22] FAN High Speed Value.

5.4 HEAT mode

In this mode the FAN is on when runtime register [106] main output is positive. The FAN is operating at the speed set in the register [22] FAN High Speed Value.

5.5 COOL / HEAT mode

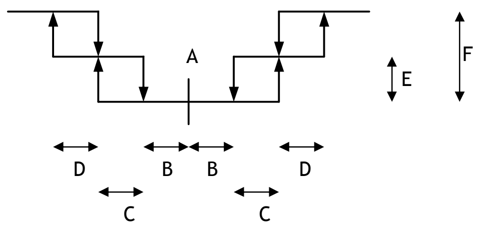
In this mode the FAN is on when runtime register [106] main output not is zero. The FAN is operating at the speed set in the register [22] FAN High Speed Value.

5.6 FAN regulator mode

In this mode we get the FAN to switch on and off with the input temperature, adjusted to the FAN set point register. The FAN has two speeds, which are adjustable.

Fan 1 control

[17] A	Set point	-50 … 100°C
[18] B	Dead band	0 … 50°C
[19] C	Low speed hysteresis	0 … 10°C



[20]	D	High speed hysteresis	0 … 10°C
[21]	E	Low speed voltage	0 … 30V
[22]	F	High speed voltage	0 … 30V

Fan 2 control

[23]	A	Set point	-50 … 100°C
[24]	B	Dead band	0 … 50°C
[25]	C	Low speed hysteresis	0 … 10°C
[26]	D	High speed hysteresis	0 … 10°C
[27]	E	Low speed voltage	0 … 30V
[28]	F	High speed voltage	0 … 30V

6 Temp sensor

The Temperature settings are possible to adjust for good quality of the signal.
The basic use of gain and offset is in the following equation:

temp = (1024- ad_value + offset) * gain;

The 1024 value is adjusting AD value to go from 1..1024 for rising temperature, if NTC resistance is in use.

NOTE: standard mode is to use Steinhart coefficients to calculate right temperature.

6.1 Temp 1

Temp 1 is the main temp sensor for the system. This is the only sensor with the possibility to adjust the electrical parts on board by changing digital potentiometers. The connector's pin 1 is feeding voltage to the sensor with at serial resistor of 20k. Pin 3 is the ground. Pin 2 is the input signal with a pull-up resistor of 510kohm. The purpose of this resistor is to put the input signal high if no sensor is connected. The digital pot offset control is adjusting the reference voltage from 0 to 5 Volt, where pot value of 0 is 0 volt, and pot value of 255 is 5Volt. The pot gain value of 0 is max gain, and the value of 255 is the gain of 1.
The purpose of the digital pot values is mainly to adjust to the sensor connected, but it is also possible to zoom in on the signal and track it. Then the temperature resolution can be very high.

In the future we will integrate a possibility to put in the K factors of the sensor, and automatically get high-resolution sensor signal. But for now, we need to adjust this by hand. Contact Supercool to get help with the best values for your sensor.

Sensor mode bit:
0 – not implemented
1 – not implemented
2 – to activate Steinhart calculation
3 – to activate Zoom mode (internal control of digital pot). Have this bit set to achieve maximal resolution.
4 – to activate PT mode
5..7 – not implemented

NOTE: standard mode is to use Steinhart coefficients to calculate right temperature.

[35] Temp 1 gain
[36] Temp 1 offset
[43] Temp 1 digital pot offset
[44] Temp 1 digital pot gain

[55] Temperature 1 sensor mode
[59] Temp1 Steinhart coeff A
[60] Temp1 Steinhart coeff B
[61] Temp1 Steinhart coeff C
[79] Temp1 Steinhart resistance value high
[80] Temp1 Steinhart resistance value mid
[81] Temp1 Steinhart resistance value low

6.2 Temp 2 and Temp 3

Temp 2 and 3 are temp sensor inputs of the same kind. The electrical gain is 1.05, but can be changed by changing the resistor values. There is a pull-up resistor of 5k1 ohm. If no sensor is connected to the input, the AD value will be 1023 (the highest value of a 10bits AD). With this in mind it is possible to calculate the right values for the gain and offset values. The values can be pos or negative.

Sensor mode bit:
0 – not implemented
1 – not implemented
2 – to activate Steinhart calculation
3..7 – not implemented

NOTE: standard mode is to use Steinhart coefficients to calculate right temperature.

[37] Temp 2 gain
[38] Temp 2 offset
[56] Temperature 2 sensor mode
[62] Temp2 Steinhart coeff A
[63] Temp2 Steinhart coeff B
[64] Temp2 Steinhart coeff C
[82] Temp2 Steinhart resistance value high
[83] Temp2 Steinhart resistance value mid
[84] Temp2 Steinhart resistance value low
[39] Temp 3 gain
[40] Temp 3 offset
[57] Temperature 3 sensor mode
[65] Temp3 Steinhart coeff A
[66] Temp3 Steinhart coeff B
[67] Temp3 Steinhart coeff C
[85] Temp3 Steinhart resistance value high
[86] Temp3 Steinhart resistance value mid
[87] Temp3 Steinhart resistance value low

6.3 Temp FET

Temp FET is a temp sensor on the board, close to the power transistors (FET), which gives us the possibility to check for overheating before it becomes a problem. The electrical gain is 1.05, but can be changed by changing the resistor values. There is a pull-up resistor of 5k1 ohm. If no sensor is connected to the input, the AD value will be 1023 (the highest value of a 10bits AD). With this in mind it is possible to calculate the right values for the gain and offset values. The values can be positive or negative.
The NTC resistance used is 1kohm at +25 degree C. B25/50 value is 3200 K.

NOTE: standard mode is to use Steinhart coefficients to calculate right temperature.

[41] Temp 4 gain
[42] Temp 4 offset

[58] Temperature 4 sensor mode
[68] Temp4 Steinhart coeff A
[69] Temp4 Steinhart coeff B
[70] Temp4 Steinhart coeff C
[88] Temp4 Steinhart resistance value high
[89] Temp4 Steinhart resistance value mid
[90] Temp4 Steinhart resistance value low

7 "$Ax" - Continues log

We have a possibility to get log values at the regulator speed (20Hz), as a continuous log. Let the application software receive the log values for real time presentation of values in different modes, or use for debug purpose, log to a text file, and analyze the data in an Excel spreadsheet. When read into Excel use filter with <SPACE> as tab separator.
After issuing the command, the first text line is the header data. After that we get a text line for every regulator sample, at the speed of 20Hz. All values are <space> separated. The first char in the row is the log mode value. So A1, will have number 1 as the first char.

"$A" – stop the log
"$A1" – log all A/D values
"$A2" – log some of the global values
"$A3" – log PID regulator values
"$A4" – log all temperature related values
"$A5" – log temperature regulator input values
"$A6" – runtime data
"$A7" – runtime data in IEEE754 mode
"$A8" – runtime data of LOGG data

7.1 "$A1"

Data value:
1 - Mode value 1
2 – AD0, not in use, value can be anything
3 – AD1, input voltage to the regulator
4 – AD2, FAN2 current
5 – AD3, temperature input 1
6 – AD4, temperature input 2
7 – AD5, temperature input 3
8 – AD6, temperature input FET
9 – AD7, main out current
10 – AD8, internal voltage value
11 – AD9, FAN 1 current
12 – AD10, not in use, value can be anything
13 – AD11, not in use, value can be anything

7.2 "$A2"

Data value:
1 – Mode value 2
2 – error flags, value in hex
3 – regulator mode flag, value in hex
4 – AD value of temperature 1
5 – Main (Tc) output value

6 – FAN 1 output value
7 – FAN 2 output value

7.3 “$A3”

Data value:
1 – Mode value 3
2 – error flags, value in hex
3 – regulator mode flag, value in hex
4 – Tc value (+/- 100)
5 – Ta1 temperature 1 value in degree C
6 – Ta2 temperature 2 value in degree C
7 – Tr set point value in degree C
8 – Ta value in degree C
9 – Tp value
10 – Ti value
11 – Td value
12 – TLP_A value
13 – TLP_B value

7.4 “$A4”

Data value:
1 – Mode value 4
2 – error flags, value in hex
3 – regulator mode flag, value in hex
4 – Tc value (+/- 100)
5 – Tr set point value in degree C
6 – AD value of current load

7.5 “$A5”

Data value:
1 – Mode value 5
2 – error flags, value in hex
3 – regulator mode flag, value in hex
4 – TrExt, temperature value, external ref
5 – Tref, temperature value
6 – Tr, temperature value

7.6 “$A6”

Data value:
1 – Mode value 6
<runtime data> contact Supercool to get more information

7.7 “$A7”

Data value:
1 – Mode value 7
<runtime data IEEE754 mode> contact Supercool to get more information

7.8 “$A8”

Data value:

1 – Mode value 8
2 – logg count value, will save when the value is 24000, and will cleared to zero. (20Hz * 60s * 20min = 24000)

8 ALARM settings

We can check parts of the system by monitoring the value and alarm if we get to the trig value selected. Clear enable bit to zero if no alarm checking should be performed. The value zero in ALARM register is now a valid number, and will not stop the alarm check, as it used to.

[45] ALARM level voltage high
[46] ALARM level voltage low
[47] ALARM level main load current high
[48] ALARM level main load current low
[49] ALARM level FAN 1 current high
[50] ALARM level FAN 1 current low
[51] ALARM level FAN 2 current high
[52] ALARM level FAN 2 current low
[53] ALARM level internal 12 voltage high
[54] ALARM level internal 12 voltage low
[91] ALARM low enable bits
[92] ALARM high enable bits

The alarm values entered are real values.

Use “$A1” to get the A/D values.

9 Status and error flags

We have Status and Error flags.

“$S” – to read the status and error flags.
“$SC” – to clear error flags.

Result string from unit:
XXXX YYYY ZZZZ

Where
XXXX = Temperature alarm flags in HEX
YYYY = current error flags in HEX
ZZZZ = old error flags in HEX, show the errors that occurred since power up or last error clear.

Status bits:
0 = Temperature sensor 1 - to high
1 = Temperature sensor 1 - to low
2 = Temperature sensor 1 – shortcut
3 = Temperature sensor 1 - missing
4 = Temperature sensor 2 - to high
5 = Temperature sensor 2 - to low
6 = Temperature sensor 2 – shortcut
7 = Temperature sensor 2 - missing
8 = Temperature sensor 3 - to high
9 = Temperature sensor 3 - to low
10 = Temperature sensor 3 – shortcut

11 = Temperature sensor 3 - missing
12 = Temperature sensor 4 - to high
13 = Temperature sensor 4 - to low
14 = Temperature sensor 4 – shortcut
15 = Temperature sensor 4 - missing

ERROR flag bits:
0 = STARTUP_DELAY, internal telling that we are in the startup delay mode
1 = DOWNLOAD_ERROR, we got error while downloading registers
2 = C_ERROR, critical error flag
3 = R_ERROR, regulator overload error
4 = HIGH_VOLT, we detected high voltage, alarm value set in register [45]
5 = LOW_VOLT, we detected low voltage, alarm value set in register [46]
6 = HIGH_12V, internal 12V is too high, alarm value set in register [53]
7 = LOW_12V, internal 12V is to low, alarm value set in register [54]
8 = CURRENT_HIGH, we detected high load current, alarm value set in register [47]
9 = CURRENT_LOW, we detected low load current, alarm value set in register [48]
10 = FAN1_HIGH, we detected high FAN current, alarm value set in register [49]
11 = FAN1_LOW, we detected low FAN current, alarm value set in register [50]
12 = FAN2_HIGH, we detected high FAN current, alarm value set in register [51]
13 = FAN2_LOW, we detected low FAN current, alarm value set in register [52]
14 = Temp sensor alarm has stoped the regulator
15= Temp sensor alarm is indication only

10 Runtime registers

The master unit can read the runtime register values and watch over the regulator function and performance, if needed. Read the same way as normal registers.

11 The parameter.h file

Use this file in the PC application to always access the right register.

```
// -----------------------------
// PARAMETER.H
// -----------------------------
//
// Supercool Parameter file
//
// 2005-03-02, ver 1.6d
// * Add FAN gain variable
//
// 2005-02-28, ver 1.6c
// * Change _stain_ to _coff_ values
// * Support of the PT sensor mode
//
// 2004-12-20, ver 1.6b
// * add filter type selections
//
// 2004-11-08, ver 1.6d
// * adding alarm mask variabel
//
// 2006-11-29, ver 1.6e
// * add current gain calibration value
//

#ifndef _PARAMETER_H
#define _PARAMETER_H

#define VER_INTERFACE "SSCI_v1.6d"

#ifdef _WIN32
typedef unsigned int uns16;
```

```
#endif

// ---------------------------------------------------------
typedef union {
   uns16 ALL;
   struct {
       unsigned STARTUP_DELAY:1;    // mark startup delay
       unsigned DOWNLOAD_ERROR:1;   // we got a timeout while downloading parameters
       unsigned C_ERROR:1;          // critical error, outside voltage, outside current
       unsigned REG_OVERLOAD_ERROR:1;   // regulator overload error

       unsigned HIGH_VOLT:1;        // we have high voltage error
       unsigned LOW_VOLT:1;     // we have low voltage error
       unsigned HIGH_12V:1;     // we have high 12 voltage in the system
       unsigned LOW_12V:1;          // we have low 12 voltage in the system

       unsigned CURRENT_HIGH:1;     // over or under current check
       unsigned CURRENT_LOW:1;      // over or under current check
       unsigned FAN1_HIGH:1;
       unsigned FAN1_LOW:1;

       unsigned FAN2_HIGH:1;
       unsigned FAN2_LOW:1;
       unsigned TEMP_SENSOR_ALARM_STOP:1;   // sensor alarm to stop regulator
       unsigned TEMP_SENSOR_ALARM_IND:1;// sensor alarm indication
       // ### OBS! max 16 bits
   }BIT;
} ERROR_BITS;  // g_error, g_error_old

// This bits will indicate an alarm,
// and when used with mask, will cause error
typedef union {
   uns16 ALL;
   struct {
       unsigned TEMP1_HIGH:1;
       unsigned TEMP1_LOW:1;
       unsigned TEMP1_SHORT:1;
       unsigned TEMP1_MISSING:1;

       unsigned TEMP2_HIGH:1;
       unsigned TEMP2_LOW:1;
       unsigned TEMP2_SHORT:1;
       unsigned TEMP2_MISSING:1;

       unsigned TEMP3_HIGH:1;
       unsigned TEMP3_LOW:1;
       unsigned TEMP3_SHORT:1;
       unsigned TEMP3_MISSING:1;

       unsigned TEMP4_HIGH:1;
       unsigned TEMP4_LOW:1;
       unsigned TEMP4_SHORT:1;
       unsigned TEMP4_MISSING:1;
       // ### OBS! max 16 bits
   }BIT;
} TEMP_ALARM_BITS;  // g_error_tempsensor

// ---------------------------------------------------------
// Alarm enable bits
// ---------------------------------------------------------
typedef union {
   uns16 ALL;
   struct {
       unsigned OVER_VIN:1;
       unsigned UNDER_VIN:1;
       unsigned OVER_12V:1;
       unsigned UNDER_12V:1;

       unsigned OVER_CURR:1;
       unsigned UNDER_CURR:1;
       unsigned OVER_FAN1:1;
       unsigned UNDER_FAN1:1;

       unsigned OVER_FAN2:1;
       unsigned UNDER_FAN2:1;
       // ### obs! max 16 bits
   } BIT;
} E_ALARM_L;   // used in G_ui_alarm_enable_L

typedef union {
   uns16 ALL;
   struct {
       unsigned HIGH_T1:1;
       unsigned LOW_T1:1;
       unsigned HIGH_T2:1;
       unsigned LOW_T2:1;

       unsigned HIGH_T3:1;
       unsigned LOW_T3:1;
       unsigned HIGH_T4:1;
       unsigned LOW_T4:1;

       // ### obs! max 16 bits
   } BIT;
} E_ALARM_H;   // used in G_ui_alarm_enable_H


// ------------------------------------------------------------
// G_ui_R_Mode
// ------------------------------------------------------------
#defineR_MODE_OFF       0   // no regulator
#defineR_MODE_POWER 1   // POWER mode
#defineR_MODE_ALGO      2   // ON/OFF mode
#defineR_MODE_P    3   // P regulator
#defineR_MODE_PI        4   // PI regulator
#defineR_MODE_PD        5   // PD regulator
#defineR_MODE_PID       6   // PID regulator
#define R_MODE_MASK             0x000f  // bit 0..3
#define R_MODE_TrExtSelect      0x0010  // bit 4
#define R_MODE_TcPowerInt       0x0020  // bit 5
#define R_MODE_DownloadParameters 0x0040// bit 6
#define R_MODE_AutoStart        0x0080  // bit 7
#define R_MODE_LoopMode         0x0100  // bit 8
#define R_MODE_InvertOutput     0x0200  // bit 9
//...
#define R_MODE_FilterAMask      0x3000
#define R_MODE_FilterA_OFF      0       // off
#define R_MODE_FilterA_MUL      0x1000  // multiplication type
#define R_MODE_FilterA_LIN      0x2000  // linjear type, not implemented yet
#define R_MODE_FilterA_LEAD_LAG 0x3000  // lead / lag type, not implemented yet

#define R_MODE_FilterBMask      0xc000
#define R_MODE_FilterB_OFF      0       // off
#define R_MODE_FilterB_MUL      0x4000  // multiplication type
#define R_MODE_FilterB_LIN      0x8000  // linjear type, not implemented yet
#define R_MODE_FilterB_LEAD_LAG 0xc000  // lead / lag type, not implemented yet
//## max 16 bits

// ------------------------------------------------------------
// G_ui_Fx_Mode
// ------------------------------------------------------------
#define RFx_MODE_MASK       0x0f// bit 0..3
#defineRFx_MODE_OFF     0   // FAN off
#defineRFx_MODE_ALWAYS_ON   1
#defineRFx_MODE_COOL        2
#defineRFx_MODE_HEAT        3
#defineRFx_MODE_COOL_HEAT   4
#defineRFx_MODE_ALGO        5
//## max 8 bits

// ------------------------------------------------------------
// G_ui_temp1_mode
// ------------------------------------------------------------
//#define TEMPx_MODE_bVrefPlus  0x0001  // ## removed, not in use
//#define TEMPx_MODE_bVrefMinus 0x0002  // ## removed, not in use
#define TEMPx_MODE_bNTC         0x0004  // NTC sensor with stainhart algorithm
#define TEMPx_MODE_bZoom        0x0008  // ZOOM mode
#define TEMPx_MODE_bPT          0x0010  // Platina sensor, type PT100, PT1000 etc
//## max 16 bits


// ------------------------------------------------------------
// g_param[] index list
// ------------------------------------------------------------
```

```
#define REG_VERSION 0x04        // ## change this if you edit in the list

//
// REGISTER LIST, used to save data to FLASH
// Will be saved in FLASH if issued the command
//
// 32bits values
//
enum {
   G_TrInt = 0,     // (0)
   G_Kp,                // (1)
   G_Ki,                // (2)
   G_Kd,                // (3)
   G_KLP_A,         // (4)
   G_KLP_B,         // (5)
   G_RegLim,            // (6)
   G_Deadband,          // (7)
   G_ILim,              // (8)
   G_Ts,                // (9)
   G_CoolGain,          // (10)
   G_HeatGain,          // (11)
   G_Decay,         // (12)
   G_ui_R_Mode,     // (13) *16*
   G_ON_TDb,            // (14)
   G_ON_THyst,          // (15)

   G_ui_F1_Mode,        // (16) *8* FAN 1
   G_F1_Tr,         // (17)
   G_F1_Db,         // (18)
   G_F1_Hyst,           // (19)
   G_F1_Fh,         // (20)
   G_F1_LSV,            // (21)
   G_F1_HSV,            // (22)

   G_ui_F2_Mode,        // (23) *8* FAN 2
   G_F2_Tr,         // (24)
   G_F2_Db,         // (25)
   G_F2_Hyst,           // (26)
   G_F2_Fh,         // (27)
   G_F2_LSV,            // (28)
   G_F2_HSV,            // (29)

   G_ScaleAin_AD_offset,    // (30)
   G_ScaleAin_offset,       // (31)
   G_ScaleAin_gain,     // (32)
   G_ScaleAout_offset,      // (33)
   G_ScaleAout_gain,        // (34)

   G_Temp1_gain,        // (35)
   G_Temp1_offset,      // (36)
   G_Temp2_gain,        // (37)
   G_Temp2_offset,      // (38)
   G_Temp3_gain,        // (39)
   G_Temp3_offset,      // (40)
   G_Temp4_gain,        // (41)
   G_Temp4_offset,      // (42)

   G_ui_pot_offset, // (43) *8*
   G_ui_pot_gain,       // (44) *8*

   G_v_high,            // (45) error level check
   G_v_low,         // (46)
   G_curr_high,     // (47)
   G_curr_low,          // (48)
   G_fan1_high,     // (49)
   G_fan1_low,          // (50)
   G_fan2_high,     // (51)
   G_fan2_low,          // (52)
   G_v12_high,          // (53)
   G_v12_low,           // (54)

   G_ui_temp1_mode, // (55) *8*
   G_ui_temp2_mode, // (56) *8*
   G_ui_temp3_mode, // (57) *8*
   G_ui_temp4_mode, // (58) *8*
   // in NTC mode:
   // coff A, B, C is the stainhart coff's
   // and res_H, M, L is the resistor values for the points
   //
   // in PT mode
   // coff A, B, and res_H (Ro) is used in the algorithm
   //
   G_t1_coff_A,     // (59)
   G_t1_coff_B,     // (60)
   G_t1_coff_C,     // (61)
   G_t2_coff_A,     // (62)
   G_t2_coff_B,     // (63)
   G_t2_coff_C,     // (64)
   G_t3_coff_A,     // (65)
   G_t3_coff_B,     // (66)
   G_t3_coff_C,     // (67)
   G_t4_coff_A,     // (68)
   G_t4_coff_B,     // (69)
   G_t4_coff_C,     // (70)

   G_alarm_temp1_high,      // (71)
   G_alarm_temp1_low,       // (72)
   G_alarm_temp2_high,      // (73)
   G_alarm_temp2_low,       // (74)
   G_alarm_temp3_high,      // (75)
   G_alarm_temp3_low,       // (76)
   G_alarm_temp4_high,      // (77)
   G_alarm_temp4_low,       // (78)

   G_t1_res_H,      // (79) stain resistans (high/mid/low) point for coff A, B, C
   G_t1_res_M,      // (80)
   G_t1_res_L,      // (81)
   G_t2_res_H,      // (82)
   G_t2_res_M,      // (83)
   G_t2_res_L,      // (84)
   G_t3_res_H,      // (85)
   G_t3_res_M,      // (86)
   G_t3_res_L,      // (87)
   G_t4_res_H,      // (88)
   G_t4_res_M,      // (89)
   G_t4_res_L,      // (90)

   G_ui_alarm_enable_L, // (91) *16* [E_ALARM_L] enable bit for alarm setting
   G_ui_alarm_enable_H, // (92) *16* [E_ALARM_H]

   G_TrInt2,            // (93) Setpoint 2, when in loop mode low value
   G_uiTimeHigh,        // (94) *16* time, 0xffff=65535 loops / 20 = 3276 sekunder
   G_uiTimeLow,     // (95) *16* time, 0xffff=65535 loops / 20 = 3276 sekunder

   G_ui_sensor_alarm_mask,  // (96) *16* enable which alarm that should stop regulator

   G_fMainCurrGain, // (97) float32 main gain constant

   G_SIZE               // must be last in list, max value 255
} PARAM_NUM;


// RUNTIME DATA (READ)
// and (READ_WRITE) temporary adjustments value
#define GR_event_count  99
#define GR_temp1        100
#define GR_temp2        101
#define GR_temp3        102
#define GR_temp4        103
#define GR_pot_input    104
#define GR_tref         105
#define GR_tc_main      106
#define GR_fan1_main    107
#define GR_fan2_main    108

#define GR_pid_Ta       110
#define GR_pid_Te       111
#define GR_pid_Tp       112
#define GR_pid_Ti       113
#define GR_pid_Td       114

#define GR_pid_TLPa     117
```

```
#define GR_pid_TLPb     118

#define GR_onoff_runtime_state  122     // int8
#define GR_onoff_runtime_max    123     // float
#define GR_onoff_runtime_min    124     // float
#define GR_fan1_runtime_state   125     // int8
#define GR_fan1_runtime_max     126     // float
#define GR_fan1_runtime_min     127     // float
#define GR_fan2_runtime_state   128     // int8
#define GR_fan2_runtime_max     129     // float
#define GR_fan2_runtime_min     130     // float

#define GR_inputvolt    150     // float (Volt)
#define GR_internalvolt 151     // float (Volt)
#define GR_main_current 152     // float (Amp)
#define GR_fan1_current 153     // float (Amp)
#define GR_fan2_current 154     // float (Amp)

#define GRW_fan_gain    155     // float (gain)


#endif // _PARAMETER_H
```

Revision list:

Rev 1.1
- Created this document
- Description of new serial interface

Rev 1.2 040831 / Thomas
- Implemented full description of the commands.

Rev 1.3 040908 / Thomas
- Implementing PR59 register use

Rev 1.4 041021 / Thomas
- Removed register 119, 120, 121
- Added registers 78 to 92
- New command RNxx using IEEE754 type
- Include "parameter.h" file
- We now uses Steinhart coefficients to calculate temperature value
- Corrected some typo error
- Sensor information update

Rev 1.6b 050113 / Thomas
- Many changes in the document
- New command handling ID information
- New command handling auto logging of some data
- More registers defined and in use

Rev 1.6c 050211 / Thomas
- Correct some typo error

Rev 1.6d 050428 / Thomas
- New parameter list

Rev 1.6e 110411 / Stefan Winberg
- Corrected some errors in Chapter 2 regarding response on commands.
- Updated register list in Chapter 3.
- Updated information in chapter 5.
- Updated temp modes in chapter 6.1
- Updated chapter 8 with that all uploaded values are real, not AD values as before.

Rev 1.6f 120830 / Stefan Winberg
- Corrected definition of parameter B in the pictures in chapter 4.3 and 5.6.