J² TECHNOLOGY CONSULTING

AFE2401

24 Bit Sensor Analog Front End

Technical Manual

Revision 3, Updated 14 jan 21

J² TECHNOLOGY CONSULTING AB ▪ LINKÖPING, SWEDEN
EUROPE PHONE +46(0)70 2780460 ▪ EMAIL: info@j-2.se ▪ INTERNET: http://www.j-2.se

Revision history

Last changes made by administratör

Version	Date	Comment
0.1	5/11/20	a, initial draft
0.2	6/11/20	PA, changed name to AFE2401
0.3	14/1/21	PA, added pull-up information to i2c section.

DECLARATION OF CONFORMITY

Conformity to EMC and LV regulations has not been tested for this product.

NOTICE

Disclaimer
This limited warranty is the sole and exclusive warranty provided by J2 Holding AB in connection with your AFE2401 analog front end and is, where permitted by law, in lieu of all other warranties, conditions, guarantees, representations, obligations and liabilities, express or implied, statutory or otherwise in connection with the product, however arising (whether by contract, tort, negligence, principles of manufacturer´s liability, operation of law, conduct, statement or otherwise), including without restraint any implied warranty or condition of quality, merchantability or fitness for a particular purpose. Any implied warranty of merchantability or fitness for a particular purpose to the extent required under applicable law to apply to the product shall be limited in duration to the period stipulated under this limited warranty. In no event will J2 Holding AB be liable for any special, direct, indirect, incidental or consequential damages, losses, costs or expenses however arising whether in contract or tort including without restriction any economic losses of any kind, any loss or damage to property, any personal injury, any damage or injury arising from or as a result of misuse or abuse, or the incorrect installation, integration or operation of the product.

General information
Without limiting the generality of the above, unless specifically agreed to by it in writing, J2 Holding AB

A. Makes no warranty as to the accuracy, sufficiency or suitability of any technical or other information provided in manuals or other documentation provided by it in connection with the product.

B. Assumes no responsibility or liability for losses, damages, costs or expenses, whether special, direct, indirect, consequential or incidental, which might arise out of the use of such information. The use of such information will be entirely at the user's risk.

Product usage statement

The AFE2401 analog front end module is designed for analyzing small low frequency currents when excited with a low voltage source. The module also contain a heater driver output with a pt100 sensor input for automatic temperature control. The high power output could potentially be a fire hazard, and it is the end users responsibility to provide adequate current limiting by a fuse or other means.
The J2 Holding AB company shall not be liable for any loss or damage resulting from careless usage.

Product manual
The information contained in this manual is subject to changes by J2 Holding AB without prior notice. J2 Holding makes no warranty of any kind whatsoever, either expressed or implied, with respect to the information contained herein. J2 Holding AB shall not be liable in damages, of whatever kind, as a result of the reliance on or use of the information contained herein.

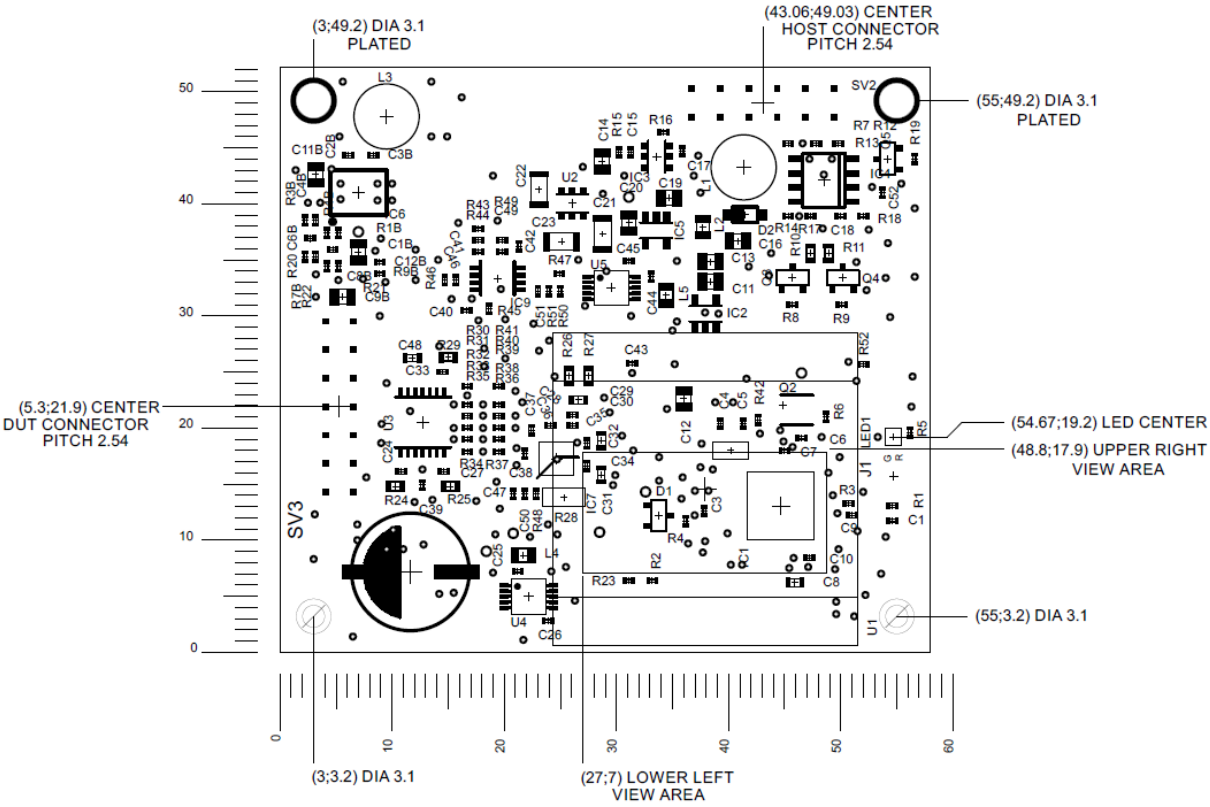
CONTENTS

1 SAFETY INFORMATION

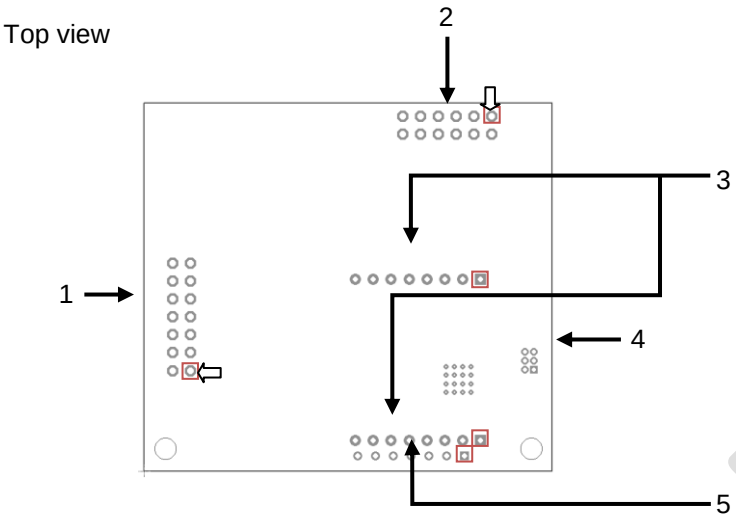
Fuse and current consumption
This module is built to provide up to 12W to a load connected on the heater output. Potentially this could present a fire hazard if used inappropriately. The heater chamber and temperature sensor to use in the closed loop regulation is to be provided by the end user.

> ⚠ Make sure that the device and *all associated equipment* is properly connected before switching on the power.

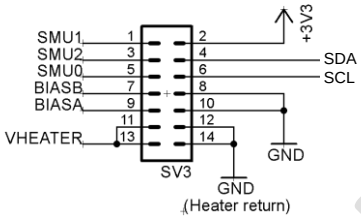
2 MECHANICAL



2.1 Connections

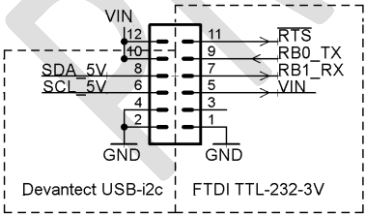


1. DUT connector. 14-pin header (2x7, 0.1" pitch). This connects to the sample and carries two SMU channels, one pt100 sensor channel, sensor i2c, heater drive (0-12V) and two bias channels (-5V...+5V).



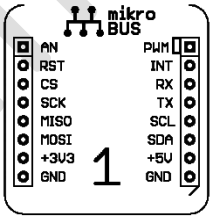
The i2c pins can be routed directly to the host connector by populating two resistors R52 and R53, but will always be connected to the board controller. If R52 and R53 are populated the i2c bus must not be pulled higher than 3.3V.

2. Host connector. 12-pin header (2x6, 0.1" pitch). This connects to the host computer via serial or i2c and carries input power (5-12V), level shifted i2c, 3.3V serial UART.



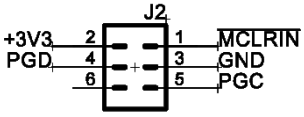
The pinout of odd pins (1,3,...,11) is arranged so a FTDI TTL-232-3V cable connector can directly be attached to these pins. Polarization must be observed. If VIN is supplied from this cable the USB port will limit the current and power available to the heater.

3. Mikrobus connector. This connector is normally not populated and is facing down from the bottom side. It is a MikroBus™ standard connector with SPI, i2c, pwm, analog and digital io signals. Expansion modules can be connected to this bus.

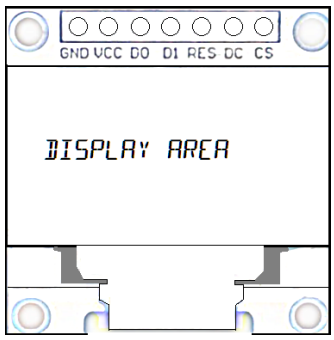


(Bottom view)

4. Microchip ICD connector. This is a miniature 6-Pin header (2x3, 0.05" pitch). It is an ICD connector used for firmware upgrade.



5. OLED connector. A 7-pin header (1x7 , 0.1" pitch). An OLED graphical display can be connected to this header.



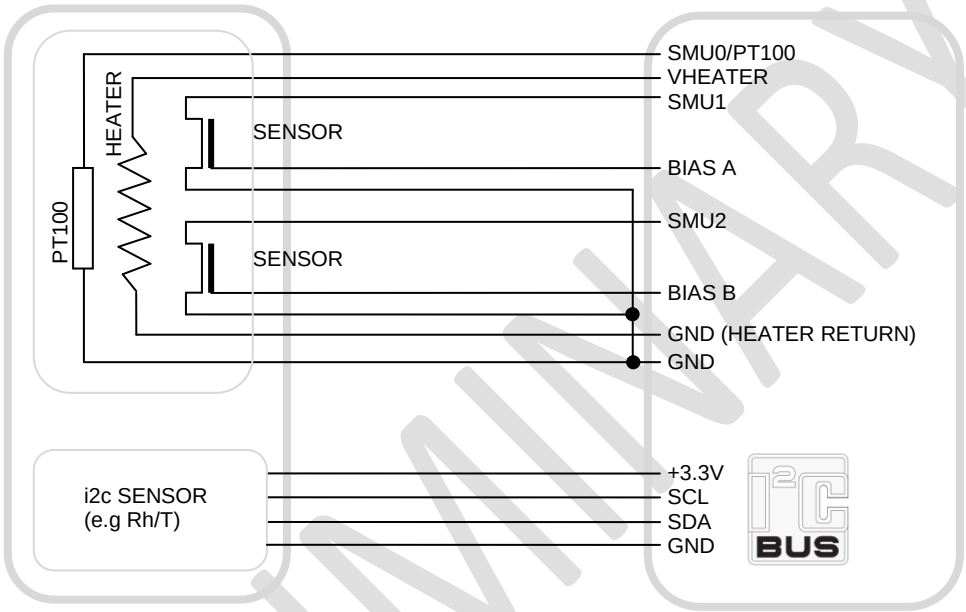
1 GND
2 VCC (+3.3V)
3 D0/SCK
4 D1/MOSI
5 RES (!RESET)
6 DC (D/!C)
7 CS (!CS)

IS 12864 OLED 0.96" SPI 128x64 SSD1306

2.2 Connecting the sample / DUT

The DUT connector provides the following features:

- **SMU0-2 (Pin 1,3,5):** Source/Measure Units, programmable sources (0-5V) with current meters. SMU0 is used for an external PT100 probe. Return: Pin 10.
- **BIAS A/B (Pin 9,7):** Bias voltages, programmable -5V ... +5V. Return: Pin 10.
- **Heater power (Pin 11,13):** Voltage powering the external heater, programmable source (0-12V) with current meter. PI regulator controlled. Return: Pin 12,14.
- **i2c digital interface (Pin 2,4,6):** i2c bus and +3.3V connected to the board controller, for DUT sensor platform i2c sensor (e.g Rh/T sensor). Return: Pin 8.



2.3 Connecting the Host

The host interface provide

- **Module power VIN (Pin 5, 12,10):** Input power for the module, +5V to +12V.
- **Host i2c Interface (Pin 8,6):** Open collector SCL/SDA interface.
  The SCL and SDA are level shifted internally and should be pulled up at the host to a voltage not exceeding VIN.
- **Serial interface (Pin 11,9,7):** 3.3V level UART, can be connected to e.g an FTDI TTL-232R-3V3 USB-Serial converter cable.

The serial port is mainly intended for setting up the device, calibration and test/debugging. In an end application the i2c slave interface is assumed to be used. The i2c interface can be connected to e.g an Arduino Uno, Beaglebone or Raspberry Pi.

3 Using the Serial port

3.1 Cabling

The module serial port connector can be used together with FTDI TTL-232R-3V3 USB-Serial converter cable available at Farnell/Newark as art no 1329311, Mouser 895-TTL-232R-3V3 etc. The connector fits directly on the serial port row (odd pins) of the host connector, with GND (pin 1) connected to the black wire.

3.2 Installing the drivers

VCP (Virtual Com Port) driver for the FTDI TTL-232R-3V3 cable can be obtained directly from the manufacturer FTDI at www.ftdichip.com.

3.3 Software and protocol

Protocol is

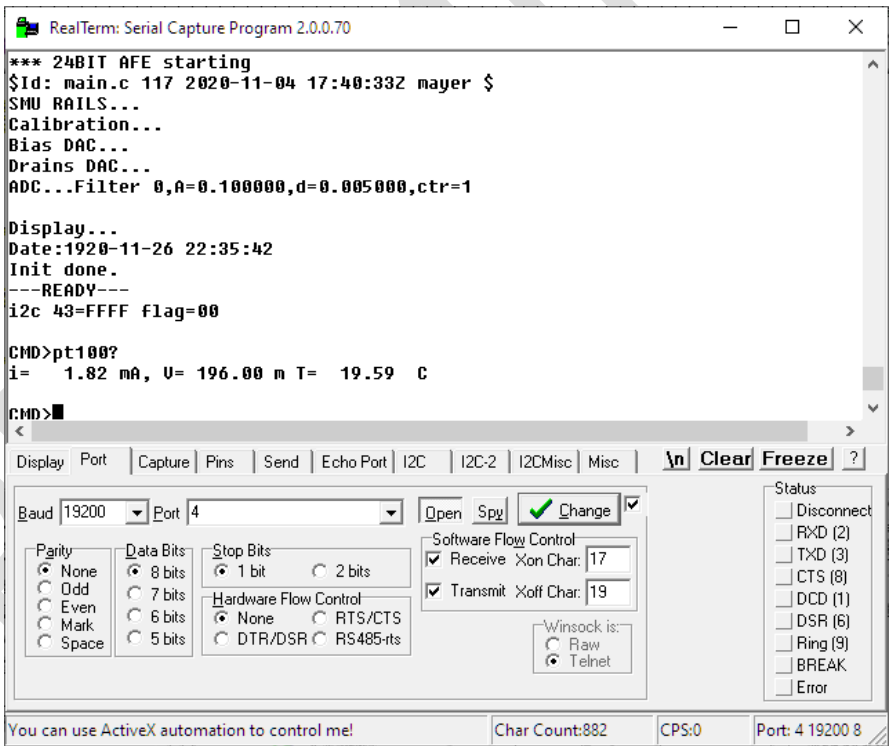
19200, 8N1, XON/ XOFF

(From host: Receive XON char:17, Transmit XOFF Char: 19).

The terminal program should be set up to perform a new line on reception of LF character (Unix standard).

Suggested software:

RealTerm	https://sourceforge.net/projects/realterm/
Putty	https://www.putty.org/
SerialPlot	https://hackaday.io/project/5334-serialplot-realtime-plotting-software



3.4 Remote control from the serial port

The serial port opens a command prompt where a command and optionally two arguments can be given. Of the different commands the most important are

sdump	Dumps real-time ADC data in CSV format for use with e.g SerialPlot
stop	Stops PI regulation and serial dump
echo	Sets verbosity level. "echo 0" turns off prompts and replies.

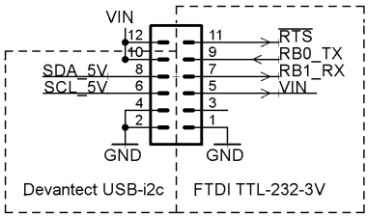
The following commands are available:

Command	Description	Command	Description
help	Show available commands	*idn?	Show identification string
ver	Show firmware version	echo	Select verbosity level, 0-2
pt100?	Read pt100 temperature	date	Show or set RTC date
rd?	Make one ADC reading, present as CSV: DRAIN1,DRAIN2,PT100,VHEATER,IHEATER, PI_TEMP,PI_SP,PI_ERR,PI_OUT,PI_STATUS, VIN	sleep	Go to sleep mode
pwr	Show heater power	tune	Auto tune the PI regulator and set new P/I parameters
adc	Show ADC information	raw <n>	Use raw ADC values (n=0)
adc <n>	Set ADC OSR index to <n>		
pt100 <u>	Set pt100 output (SMU0) to <u> mV	i2c	Show i2c registers
bias <ch> <u>	Set bias <ch=0,1> to <u> mV	cal?	Show calibration data
drain <ch> <u>	Set SMU <ch=0,1,2> to <u> mV. <ch=0,1> sets SMU 1,2. <ch=2> sets pt100 voltage and turn off PI regulation. <ch=3> sets heater voltage (See note 2)	cload	Load calibration data from EEPROM
dac <c> <n>	Directly set the 4-channel dac to value <n> (See note 1,2)	csave	Save calibration data to EEPROM
plot	See "show"	caoff	Set ADC offset
stop	Stop PI controller and "sdump"	casl	Set ADC slope
heater <u>	Set heater voltage to <u> mV (See note 2)	cal	Do offset calibration (See note 1)
show <n>	Set OLED display information. 0: Info screen 8-15 : Plot ADC channel <x>-8 16-17: reserved 18: PI out 19: PI error "e" 20: pt100 temperature (C)	reset	Restart the module
osr <n> <p>	Set oversampling rate index <n> and post processing <p>. <p>=1 for floating point filtering.	boost	Turn boost on or off (See note 1)
sdump	Start ADC dump to serial port (CSV)	i2caddr	Set i2c address
tset	Set temperature in deci-C (e.g tset 20000 is 200.00C) and start regulation	pip	Set PI controller "P" parameter (n/10)
rail	Turn on/off rails (See note 1)	pii	Set PI controller "I" parameter (n/10)
hvmax <u>	Limit the PI regulator output voltage to <u> mV	cdoff	Set DAC offset
Filt <u>	Set filter constant A to u/10.	cdsl	Set DAC slope

Note 1: Avoid using these commands as they can damage the module or destroy calibration.
Note 2: Setting the heater output voltage manually bypass firmware current and power limit. The heater output circuit will be destroyed on too high load. There is no internal fuse or current limit. Destroying the heater this way is not covered by warranty.

4 Using the i2c interface

4.1 Connection



Connect SV2 pin 8 (SDA), 6 (SCL) and any of 1, 2 or 4 (GND) to the host. Remember to use pull-up resistors to the host io-voltage.
If interfacing to a 3.3V microcontroller a typical value would be 3.3kOhm to +3.3V. If interfacing to a 5V microcontroller (e.g Ardiono Uno) a typical value would be 4.7kOhm to +3.3V.

Typically, pin 8 and 6 are not pulled up to VIN, but if using VIN=5V and a 5V host, R10 and R11 can be populated on the back side of AFE2401. This is not recommended; in case the board is later moved to a 3.3V system using e.g VIN=9V **the pull-ups might damage the host**.

4.2 Address and speed

The i2c interface responds to address 0x34. The interface has been tested at 400kHz.

4.3 Protocol

The i2c protocol give access to three types of registers: 8, 16 or 32-bit. A complete list of these registers can be found in the file “i2c_registers.h”, where the registers are also documented.

The protocol description below uses the ELV i2c-USB interface syntax. S=START, P=STOP. After START the address is given, 0x34=write, 0x35=Read. NB: START here is not the same as the START command in the CMD register. After the reading address the number of bytes to read is given, e.g S 31 **02** P will read 2 bytes.

Read version register:

S30 40 S31 02 P // Select register 0x40 (VERSION) and read 2 bytes

Reading version register with XOR checksum:

S30 40 S31 03 P // Select register 0x40 (VERSION) and read 3 bytes

The three bytes returned will be e.g (hex): 01 03 **13**

Note: **13** is the XOR of the array including (read) address and selected register: 13=31 xor 40 xor 01 xor 03.
To check if the reading is correct: simply XOR the last byte with the others.
The result should be 0.

Write, then read version register:

S30 40 AA 55 **8F** S31 02 P // Select register 0x40, write AA 55 8F and read 2 bytes.

Note: **8F** is the XOR of the array to send, 8F=30 xor 40 xor AA xor 55.
If the XOR value is incorrect the I2CERRORS register will be increased. This register can be checked to see if a command/parameter was accepted.

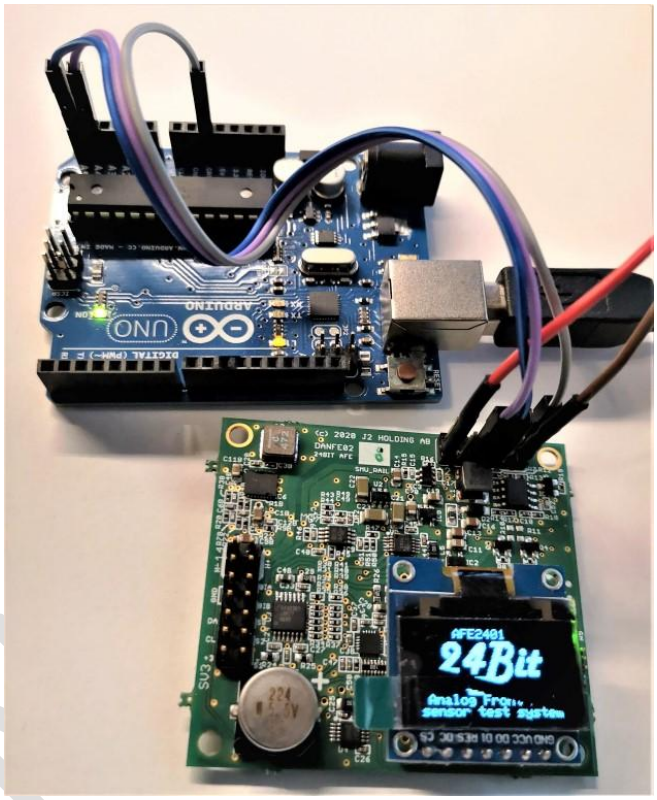
4.4 Arduino code

Sample code for Arduino Uno is available on request. The example code use a set of i2c communication functions implementing checksum etc, and is used on the form

```
#define DANFE_I2CADDRESS  0x34

r32=mktime(&rtc_currentTime)+UNIX_OFFSET;
i2cmcp_Write32bit(DANFE_I2CADDRESS,DANFE_REG_EPOCH,r32);
```

Example to show the logo:




```
#include <Wire.h>
#include "i2cmcp.h"
#include "i2c_registers.h"

#define DANFE_FW 0x1002 // Expected slave firmware
version. The one this program supports.

// ------------------------------------------------
// The function below is run once, at start up.
// ------------------------------------------------
void setup() {
  uint16_t r16;

  Serial.begin(115200);
  while (!Serial) delay(10);     // will pause Zero,
Leonardo, etc until serial console opens
  Serial.println("*** Logo test starting");

  Wire.begin();           // join i2c bus (address
optional for master)
  Wire.setClock(400000L); // Set clock to 400kHz

  // check if the I2C lines are LOW
  if (digitalRead(SDA) == LOW || digitalRead(SCL) ==
LOW)
  {
    Serial.println("i2c Bus error. The slave is hanging
the bus. Reset the system.");
    while(1);
  }
  Serial.print("Firmware: ");

  r16=i2cmcp_Read16bit(DANFE_I2CADDRESS,DANFE_REG_VERSION);

  Serial.print(r16>>8);
  Serial.print(".");
  Serial.println(r16&0xFF);
  Serial.println("");
  if (r16<DANFE_FW)
  {
     Serial.println("*** WARNING: This source code is written for another firmware version.");
     Serial.print("Source: ");
     Serial.print(DANFE_FW,HEX);
     Serial.println();
  }
  // Show the logo:

  i2cmcp_Write8bit(DANFE_I2CADDRESS,DANFE_REG_DISPLAY,2); // Show logo
}

void loop()
{

} // Loop
```

```
/*
 * File:   i2c_registers.h
 * Author: Admin
 *
 * Created on den 2 december 2019, 00:58
 * $Id: i2c_registers.h 126 2020-11-10 12:52:31Z mayer $
 *
 * FOR 24bit AFE
 *
 * Functions marked with (*) is yet to be implemented.
 * Dunctions marked with (!!NOT SUPPORTED!!) is not supported by hardware.
 *
 */

#ifndef I2C_REGISTERS_H
#define         I2C_REGISTERS_H

#ifdef          __cplusplus
extern "C" {
#endif

#define DANFE_I2CADDRESS  0x34

// Number of registers
#define n8BITREGS      10
#define n16BITREGS     15
#define n32BITREGS     10

// 8-bit
#define DANFE REG I2CERRORS   0   // (R)   Incremented if send operation caused a crc error @ the slave
#define DANFE REG I2CADDRESS  1   // (R/W) i2c address (can be changed but must be saved from serial port)
#define DANFE_REG_CL_STATUS   2   //(!!NOT SUPPORTED!!) (R)   Current loop (dac161S997) status register
#define DANFE_REG_COMMAND     3   // (R/W) Command register (see below)
#define DANFE_REG_ARG8        4   // (W) 8-bit argument to DANFE_REG_COMMAND, when applicable

#define DANFE_REG_PI_STATUS   5   // (R)   PI temp regulation status:

  #define PI STAT IDLE           0   /* Not in operation */
  #define PI_STAT_RUN            1   /* Regulating */
  #define PI_ERR_SBRK            2   /* Pt100 sensor off */
  #define PI_ERR_SSHRT           3   /* Pt100 sensor shorted */
  #define PI ERR HBRK            4   /* heater current too low */
  #define PI ERR HSHRT           5   /* heater current too high */
  #define PI ERR DRIVER OVERHEAT 6   /* (!!NOT SUPPORTED!!) Output stage overheating */
  #define PI_ERR_STUCK_AT_RAIL   7   /* Output not affecting the heater */
  #define PI_STAT_TUNING         8   /* Tuning */
  #define PI_STAT_POWERPULSE     9   /* Sending power pulse to reach init T */

#define DANFE_REG_OSR         6   // (R/W) Over Sampling Rate
//   DANFE REG OSR     Oversampling
//   15                98304 (not currently supported)
//   14                81920 (not currently supported)
//   13                49152
//   12                40960
//   11                24576
//   10                20480
//   9                 16384
//   8                 8192
//   7                 4096
//   6                 2048
//   5                 1024
//   4                 512
//   3                 256
//   2                 128
//   1                 64
//   0                 32

#define DANFE REG DISPLAY     7   // (R/W) OLED Display:
                                  //           0: Info screen
                                  //       8-13 : Plot ADC channel <x>-8
                                  //       18   : PI out
                                  //       19   : PI error e0
                                  //       20   : pt100 temp
                                  // info (else): logo

#define DANFE_REG_POST      8   // (R/W) Post processing, 1=Floting point filtered, 0=unfiltered
#define DANFE_REG_PGA       9   // (R/W) ADC gain, 0-7.
//   Value  PGA
//   7      64
//   6      32
//   5      16
//   4      8
//   3      4
//   2      2
//   1      1      DEFAULT
//   0      1/3
```

```
// 16-bit
#define DANFE_REG_VERSION    0x40 // (R) Slave firmware version
#define DANFE_REG_VDRAIN1    0x41 // (W) Drain 1 voltage, 0-5000 mV (12 bit DAC)
#define DANFE_REG_VDRAIN2    0x42 // (W) Drain 2 voltage, 0-5000 mV (12 bit DAC)
#define DANFE_REG_VPT100     0x43 // (W) PT100   voltage, 0-5000 mV (12 bit DAC).
                                  //     NB: PID regulation/Auto sampling of the pt100 temperature is enabled only
if DANFE_REG_VPT100==65535 (default)
#define DANFE_REG_VHTR       0x44 // (W) Heater  voltage, 0-12000 mV (12 bit DAC)
#define DANFE_REG_VBIAS_A    0x45 // (W) Bias voltage A, signed integer (+/-5000 mV)  (16 bit DAC)
#define DANFE_REG_VBIAS_B    0x46 // (W) Bias voltage B, signed integer (+/-5000 mV)  (16 bit DAC)

#define DANFE REG BT         0x47 //(!!NOT SUPPORTED!!) (R) (int16 t) Board temperature * 100
#define DANFE_REG_HT         0x48 //(!!NOT SUPPORTED!!) (R) (int16_t) Heater output driver temperature * 100
#define DANFE_REG_CL_DAC     0x49 //(!!NOT SUPPORTED!!) (W) Current loop (dac161S997) dac output, 4000-24000 uA

#define DANFE_REG_T_SET      0x4A // (W) Target set temperature, unsigned, <655.35 deg C (send as 65534), if
regulation is turned on (see DANFE_REG_COMMAND)
#define DANFE_REG_T_READ     0x4B // (R) Current temperature, unsigned, <655.35 deg C (send as 65534). NB: INVALID
if DANFE REG VPT100 != 65535
#define DANFE_REG_T_P        0x4C // (W) Temperature PI regulation P value
#define DANFE_REG_T_I        0x4D // (W) Temperature PI regulation I value

#define DANFE_REG_ARG16      0x4E //  (W) 16-bit argument to DANFE_REG_COMMAND, when applicable

// 32-bit
#define DANFE_REG_IDRAIN1    0x80 // (R) Current supplied to DRAIN1 (nA)
#define DANFE_REG_IDRAIN2    0x81 // (R) Current supplied to DRAIN2 (nA)
#define DANFE_REG_IPT100     0x82 // (R) Current supplied to PT100  (uA)
#define DANFE_REG_IHEATER    0x83 // (R) Current supplied to heater (uA)
#define DANFE_REG_VHTR_R     0x84 // (R) Voltage monitor of heater  (uV)
#define DANFE REG VBIAS A R  0x85 //(!!NOT SUPPORTED!!) (R) Voltage monitor of BIAS A  (uV)
#define DANFE_REG_VBIAS_B_R  0x86 //(!!NOT SUPPORTED!!) (R) Voltage monitor of BIAS B  (uV)
#define DANFE_REG_VPT100R    0x87 //(!!NOT SUPPORTED!!) (R) Voltage on pt100          (uV)

#define DANFE_REG_ARG32      0x88 // (W) 32-bit argument to DANFE_REG_COMMAND, when applicable

#define DANFE_REG_EPOCH      0x89 // (R/W) Unix epoch time

// Commands (REG_DANFE_CMD):
//       00: No operation.
//       01: STOP, turn off heater voltage
//       02: Start regulation to setpoint (DANFE_REG_T_SET). Sets DANFE_REG_VPT100=65535.
//       03: (*) Start auto tuning of PI parameters (DANFE_REG_T_P,DANFE_REG_T_I)
//       04: Start offset calibration
//       05: Toggle red LED
//       06: Set adc offset (DANFE_REG_ARG32 in uV or uA) for channel (DANFE_REG_ARG8 == 0-7 ,8-9).
//       07: Set adc slope  (DANFE_REG_ARG16 +/-3.2767)   for channel (DANFE_REG_ARG8 == 0-7).
//       08: Set dac offset (DANFE_REG_ARG32 in uV or uA) for channel (DANFE_REG_ARG8 == 0-5).
//       09: Set dac slope  (DANFE REG ARG16 +/-3.2767)   for channel (DANFE_REG_ARG8 == 0-5).
//       0A: Store calibration in EEPROM

#define DANFE_CMD_NOP              0
#define DANFE_CMD_STOP             1
#define DANFE_CMD_START_REGULATION 2
#define DANFE_CMD_START_AUTO_TUNE  3
#define DANFE_CMD_START_OFFSET_CAL 4
#define DANFE_CMD_RED_TOGGLE       5
#define DANFE_CMD_CAL_ADC_OFFSET   6
#define DANFE_CMD_CAL_ADC_SLOPE    7
#define DANFE_CMD_CAL_DAC_OFFSET   8
#define DANFE_CMD_CAL_DAC_SLOPE    9
#define DANFE CMD CAL SAVE         0x0A
#define DANFE CMD SLEEP            0x0b
#define DANFE_CMD_RESET            0x0c

// ADC channels (use with DANFE_CMD_CAL_ADC_* above) :
#define DANFE_ADC_I_DRAIN1 0
#define DANFE_ADC_I_DRAIN2 1
#define DANFE_ADC_I_PT100  2
#define DANFE_ADC_I_HEATER 3
#define DANFE_ADC_V_HEATER 4

// DAC channels (use with DANFE_CMD_CAL_DAC_* above)
#define DANFE_DAC_V_DRAIN1 0
#define DANFE DAC V DRAIN2 1
#define DANFE DAC V PT100  2
#define DANFE_DAC_V_HEATER 3
// DAC Channels (dac8562t) 16 bit
#define DANFE_DAC_V_BIASA  4
#define DANFE_DAC_V_BIASB  5

#ifdef          __cplusplus
}
#endif

#endif          /* I2C_REGISTERS_H */
```

5 Heater

5.1 Setting the heater temperature

The heater temperature can be automatically controlled by the PI regulator when a target temperature is specified and the PI regulation started.
The regulation might be aborted on

PI_ERR_SBRK	The pt100 sensor is broken (open circuit)
PI_ERR_SSHRT	The pt100 sensor is shorted
PI_ERR_HBRK	The heater element is broken (open circuit)
PI_ERR_HSHRT	The heater element is shorted
PI_ERR_STUCK_AT_RAIL	The heater element does not react

The regulator limits the output power and current from software to stay inside safe limits.

Command prompt

Command `tset <t*100>`

This sets the target temperature and start regulation. Temperature is given as an integer with two decimals, so "tset 21540" will set target temperature 215.40°C.

Example: `tset 15250`
Sets target temperature 152.5°C and starts regulation.

i2c control registers
Setting the i2c registers

DANFE_REG_T_SET	Temperature multiplied by 100 (as above)
DANFE_REG_COMMAND	DANFE_CMD_START_REGULATION starts regulation.

5.2 Setting the heater output voltage manually

The heater output voltage can be set manually. Setting the heater output voltage manually bypass firmware current and power limit.
It is advised that the load is disconnected before setting the output voltage.

> ⚠ The heater output circuit *will be destroyed* on too high load. There is no internal fuse or current limit. Destroying the heater this way is not covered by warranty.

Command prompt

Command `drain 3 <u>`
or `heater <u>`

Turns off PI regulation (if running) and sets the output voltage to <u> mV.

i2c control registers
Setting the i2c register

DANFE_REG_VHTR	<u>

Turns off PI regulation (if running) and sets the output voltage to <u> mV.

5.3 Auto tuning

The heater regulation use a combination of voltage pulse + PI regulation.
For large steps in setpoint changes, a constant voltage is applied for a short time on the load, to heat it quickly. When the temperature is close to the setpoint the normal PI regulation takes over.

To estimate the pulse time an approximation of the load C_v is measured, and the time for the heat pulse is then calculated as

$$t = C_V \cdot \frac{(T - T_0)}{P}$$

T-T_0 = Temperature step
P = Output power (assumed constant at constant voltage)

The auto tuning turns on the heater at the maximum allowed output voltage, then measure the time it takes to reach a certain maximum value and the average power used. It then estimate C_v from the formula above.

Temperature is then regulated with the PI algorithm to a constant value, using aggressive K_P and K_i constants. The voltage used to maintain the temperature is saved and used together with the temperature rise time to estimate a new set of K_i and K_i constants.

The PI algorithm is the classic

$$y(t) = K_P \cdot e(t) + K_i \sum_{n=0}^{t} e[n]$$

where $y(t)$ is the output signal (Volts), K_P is the proportional constant, K_i the integration constant, and $e(t)$ is the current error. Worth to notice is that when $e(t)$=0 we are at the setpoint, and the error is therefore 0. $y(t)$ then consist enirely of the integration sum.

The constants K_i and K_P can be adjusted manually from the command prompt or i2c.

Command prompt
Command `pip <u>`
or `pii <u>`

Sets the (P/10=K_P) and I (I/10=K_i) parameters.

i2c control registers
Setting the i2c register
DANFE_REG_T_P <u>
DANFE_REG_T_I <u>

Sets the (P/10=K_P) and I (I/10=K_i) parameters.

6 OLED Display

The optional OLED display can be used during development to quickly review circuit operation.
The display can show one "page" from a set of pre-defined pages.

Selecting the page is done from i2c or command prompt.

Command prompt

Command		`show <n>`
	or	`plot <n>`

i2c control registers

Setting the i2c register

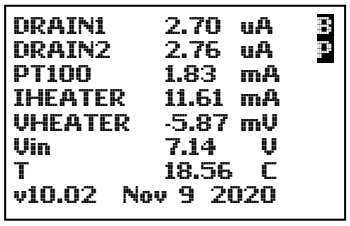
DANFE_REG_DISPLAY	<n>

Changes the OLED display to the selected page <n>:

0	Info screen
8-13	Plot ADC channel <x>-8
14-17	reserved
18	PI out
19	PI error e0
20	pt100 temperature

other numbers shows the logo.

6.1 Page 0: Info screen



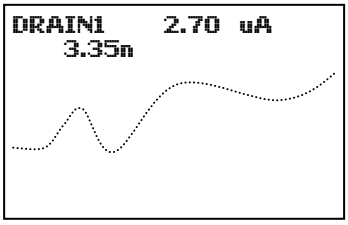
This page shows the following

DRAIN1 current	displayed in pA/nA/uA/mA
DRAIN2 current	displayed in pA/nA/uA/mA
PT100 current	displayed in pA/nA/uA/mA
HEATER current	displayed in pA/nA/uA/mA/A
HEATER voltage	Voltage on the heater element. Displayed in pV/nV/uV/mV/V.
Input voltage	Module input voltage Vin (usually 9V).
Heater element temperature	Temperature in °C
Firmware version	E.g "v10.02 Nov 5 2020".
Booster activity (on/off)	A filled "B" in the upper corner. On when Vin<7.9V.
PI regulator activity (on/off)	A filled "P" in the upper corner. On when the PI regulator is on.

6.2 Page 8-20: ADC channel plot

The following will be plotted (with auto scaling), depending on the selected channel:

8	ADC_I_DRAIN1
9	ADC_I_DRAIN2
10	ADC_I_PT100
11	ADC_I_HEATER
12	ADC_V_HEATER
13	ADC_V_IN
14-17	reserved
18	PI_OUT
19	PI_ERR
20	PI_PT100



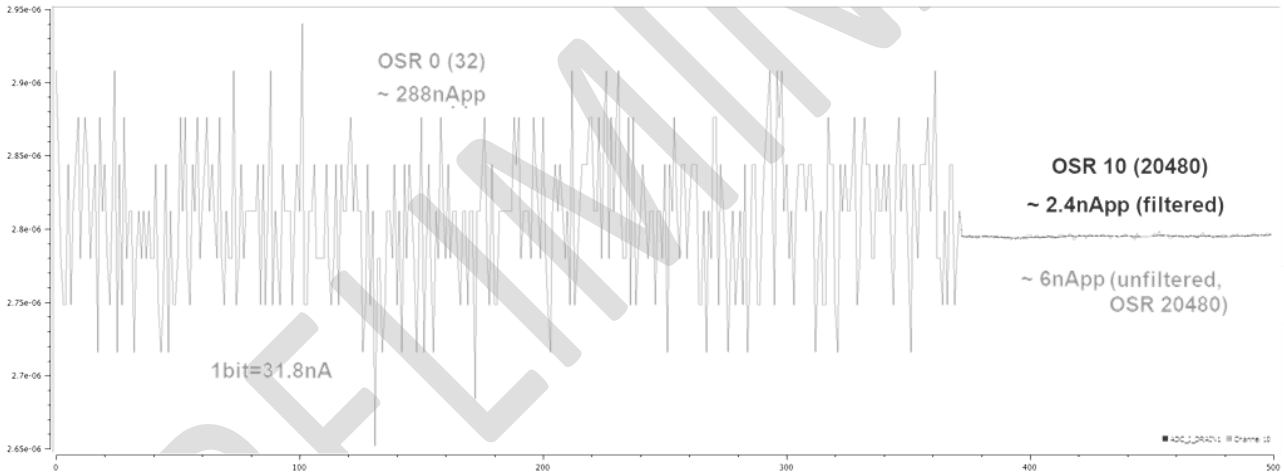
7 ADC

7.1 OSR - Oversampling Rate

OSR index sets the oversampling rate for the ADC. OSR index is written to the DANFE_REG_OSR or given as an argument to the "adc" command from the command prompt:

DANFE_REG_OSR	Oversampling
15	98304 (not currently supported)
14	81920 (not currently supported)
13	49152
12	40960
11	24576
10	20480 (default)
9	16384
8	8192
7	4096
6	2048
5	1024
4	512
3	256
2	128
1	64
0	32

High OSR give less noise but longer sampling time.
An example of the impact oversampling has on the input data is shown below:



7.2 PGA - Pgrogrammable Gain Amplifier

For very small signals the gain can be changed directly in the AD converter. The PGA setting i2c register sets the gain of the ADC:

DANFE_REG_PGA	Gain
7	64
6	32
5	16
4	8
3	4
2	2
1	1 (default)
0	1/3

7.3 SMU gain

Each SMU/Drain channel has a gain selected for optimal use of the ADC range within a current range.

The ADC uses differential inputs to measure a voltage which is proportional to the output current.

For SMU1:

ADC CH2= Positive input
ADC CH3= Negative input

ADC range = (CH2 - CH3) = Vref = 2.5V.
This represents the raw ADC value 2^23-1=0x7FFFFF.

The voltage at CH2 and CH3 is divided from the measured voltage ANP1 and ANN1 at a ratio 1/2.8:

CH2 = ANP1/2.8
CH3 = ANN1/2.8

(If, for example, ANP1=7V we get CH2=2.5V) and with

ANN1 is the programmed output voltage, the setpoint, (0-5V)
$ANP1=ANN1+i*R_G$
$R_G=$ 4700

The maximum output from the ADC of 0x7FFFFF is given when CH2=2.5V and CH3=0V.
The io stage is powered from a 7V source, so ANP1 can be maximum 7V. At maximum load, ANP1=7V and ANN1=0V gives the ADC max value of 0x7FFFFF.
This represents a maximum current of

$i=(ANP1-ANN1)/R_G$ = (7-0)/4700 = 1.489mA

However, the output voltage is not the same as the programmed voltage for the full range.
The output voltage is

$VSMU1=ANP1-i*R_G$

ANP1 can be maximum 7V to maintain the output voltage VSMU1. The maximum programmable output voltage is 5V, so the margin is 7V-5V=2V.
At 2V drop across R_G, we get a current of 2/4700=425.5µA. If the load is higher than this, we will get

$VSMU1=7-i*R_G$ for $i>$425.5µA

So the output will be

$VSMU1=7-i*R_G$ if $i>$425.5µA
or
$VSMU1=ANN1$ if $i<$425.5µA or $ANN1<7-i*R_G$

(see section "SMU output voltage vs current").

8 Sleep mode and RTC

8.1 Sleep mode

The module will go to sleep
- On an explicit sleep command from serial port or i2c
- When Vin drops below 2.5V

The module wakes up from sleep mode on activity on either the serial port or i2c interface.

In sleep mode some power regulators are still active, resulting in a load of 7mA at 9V in, but the controller draw considerably less.

The microcontroller draws approximately 10mA when idle, without display, and 265µA in sleep mode (firmware 10.02). The MCU can operate down to 2V. A diode drop in the power path gives VCC=3V normally in sleep mode. With $i = C\frac{dU}{dt}$ we get the expected run time on this capacitor of $dt = 220mF\frac{1V}{265\mu A}$ ≈ 13 minutes.

8.2 RTC

The RTC is backed up (in sleep mode) by a 220mF capacitor which is charged in normal operation. The RTC can be set from i2c by sending a 32-bit UNIX EpochTime timestamp or, from the serial port, using the "date <n>" command where <n> is a 32-bit epoch timestamp.

The current date and time can be read back with the "date" command, or read back from from the i2c register (which reads back the updated epoch timestamp). The RTC is initialized by firmware to some default date, e.g 1920-11-26 22:35:43.

Command prompt

Command	`date <n>`

i2c control registers

Setting the i2c register

DANFE_REG_EPOCH	<n>	(read or write)

The 32-bit timestamp number <n> can be calculated using an on-line service, e.g https://www.epochconverter.com/, or using the standard c library functions localtime(&n) and n=mktime().

9 Characteristics

9.1 General

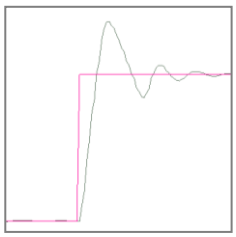
Item	Min	Typ	Max	Unit	Note
VIN voltage range	5	9	12	V	Recommended
	3		15	V	Absolute maximum
Current consumption		25		mA	Vin=9V, no display, idle
		7		mA	Vin=9V, no display, sleep
Heater output current			2	A	Vin>9V (from datasheet)
			1	A	Vin<9V (from datasheet)
			1.5	A	Limited by regulator firmware
Heater output voltage	0.5		12	V	Limited by regulator firmware, see below
Heater output power			3.5	W	Limited by regulator firmware
Heater power loss			0.6	W	9.3W to the load, Vin=12V, load=11Ω
PT100 Temperature noise			0.05	°Cp-p	Measured on a 200°C level
SMU0 Amplification		2.2		kΩ	
SMU1,2 Amplification		4.7		kΩ	
SMUx output voltage	0.07		4.950	V	
SMU1,2 output current			425	µA	At set voltage 5V
			1.4	mA	Short circuit, @ max ADC output
SMU1,2 ADC bit resolution		177.546		pA	OSR=20480
		23		Bit	
SMU1,2 DAC bit resolution		1.221		mV	
		12		Bit	
SMU1,2 noise		6.8		nApp	No load, OSR=20480, Vout=2.5V
		5.25		Bit	
BIAS output voltage	-5		5	V	No load
BIAS output current			20	mA	Approximate value, short circuit
BIAS resolution		16		Bit	
ADC update period		150		ms	

9.2 PI regulator output voltage

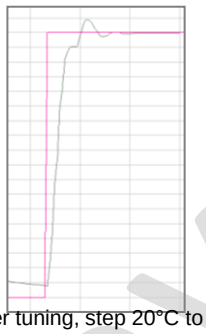
Vin min	Vin max	Heater max output	Unit	Note
>=10		VIN	V	
8	<8.5	8.5	V	
7	<8	8	V	
6	<7	7	V	
5	<6	6.5	V	
4	<5	5.5	V	

9.1 PI regulator response

The heater regulation consist of a full power pulse to get close to the setpoint fast (when the setpoint is changed), then a PI regulation. If not tuned, only the PI regulation is used.

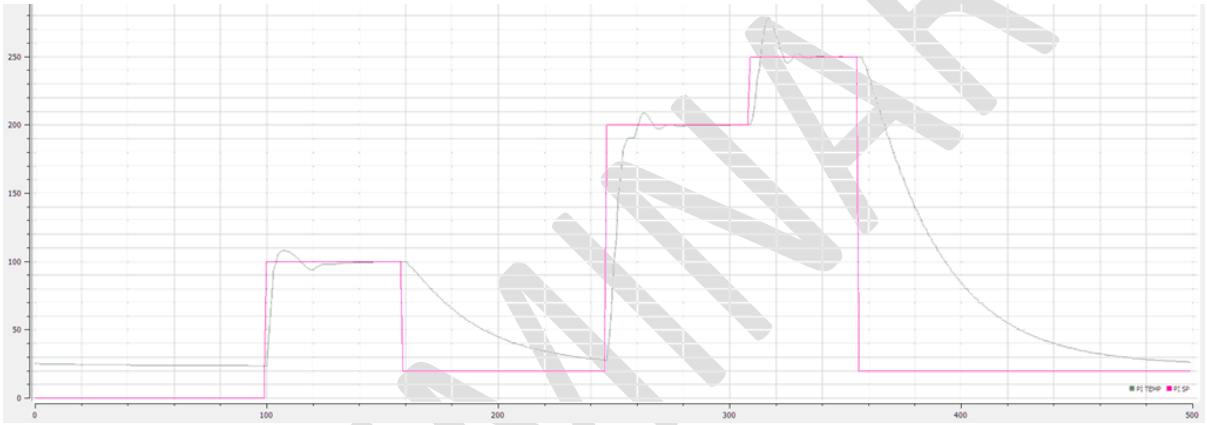


Before tuning, step 100°C to 200°C



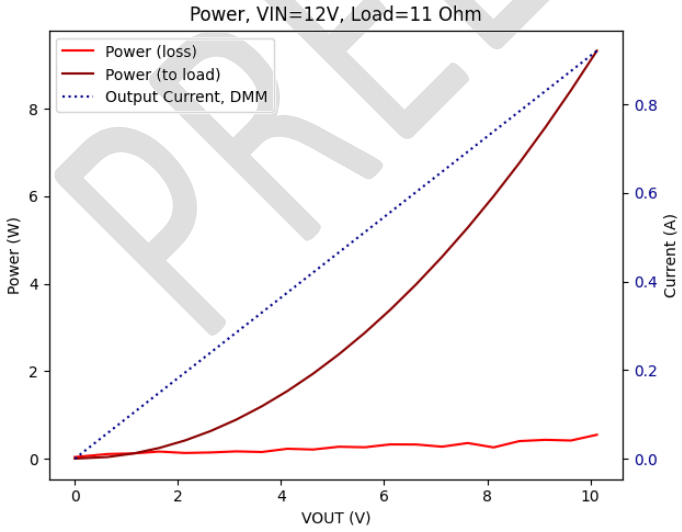
After tuning, step 20°C to 200°C

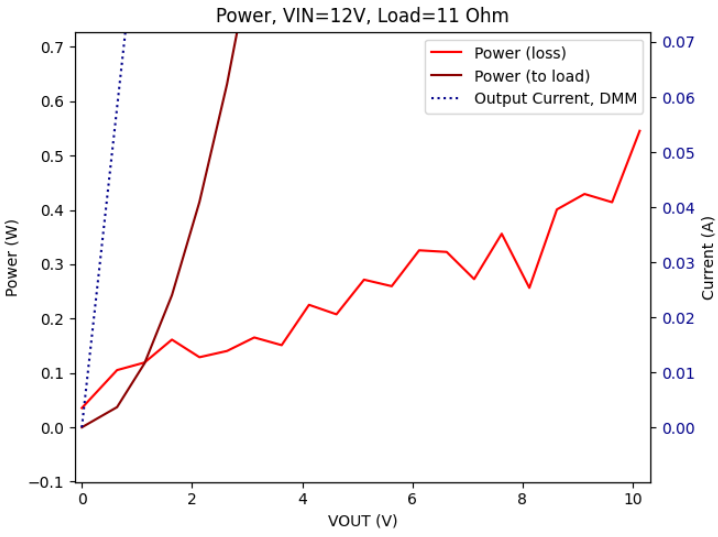
The response for some temperature steps, after tuning, is shown below



9.2 Heater power

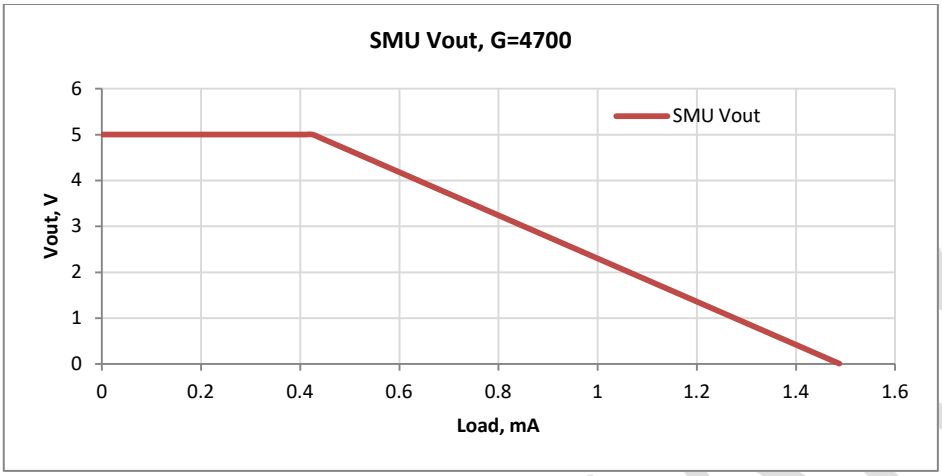
The heater works best when VOUT<=VIN. The power loss is then quite small, <0.6W for 9.3W delivered to the load.
Even so the current path in to the module needs to be well sized and decoupled.
If the power regulator starves, unpredictable behaviour might occur.





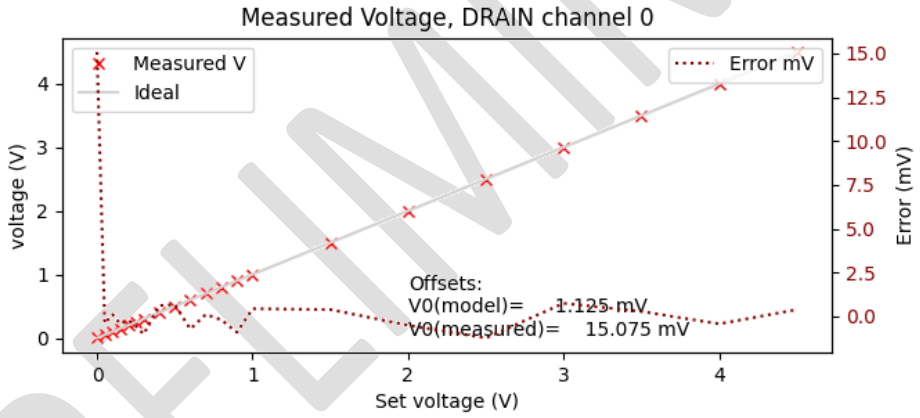
9.1 SMU output voltage vs current

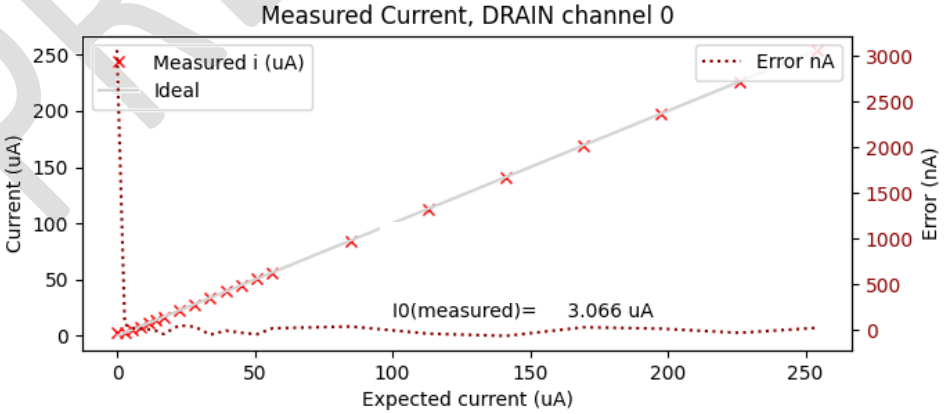
The SMU outputs with amplification 4700Ω has a load/vout characteristic as below.
This means that for 1mA load the output should not be set higher than 2V.



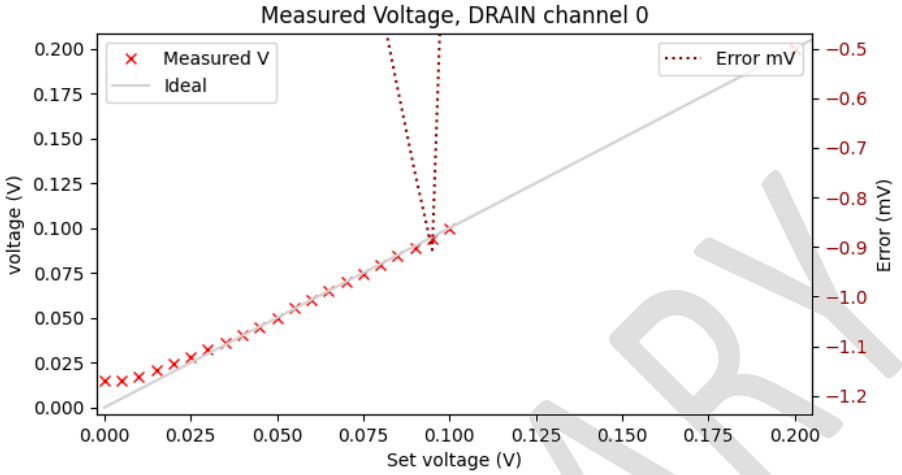
9.2 SMU output voltage vs set voltage and current error

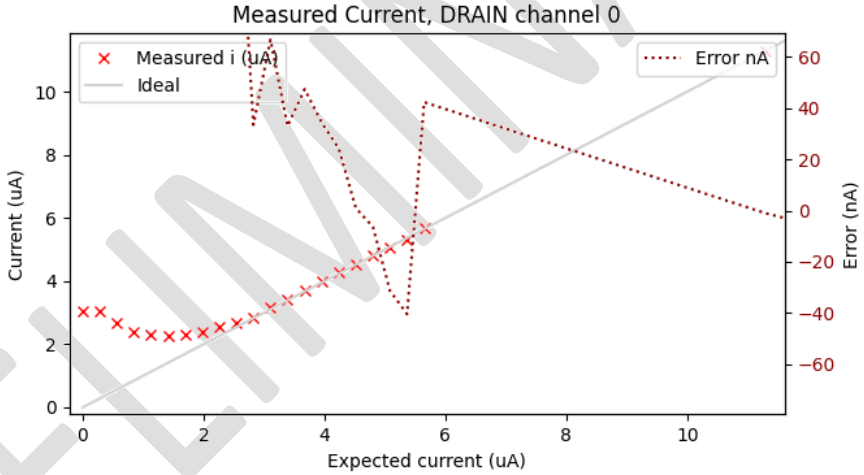
The graph below shows SMU1 response to a 18k//1MΩ Load when the programmed setpoint is increased from 0...5V.





The programmed value and the value at the output differ close to 0V, as does the measured current. The close-up below shows that programmed voltage differs from the output below approximately 70mV. This is caused by the output stage.
The error of the read back current will also differ in this region. It is therefore advised that the SMU voltage is always programmed higher than 70mV and <4.95V, to keep away from the power rails.



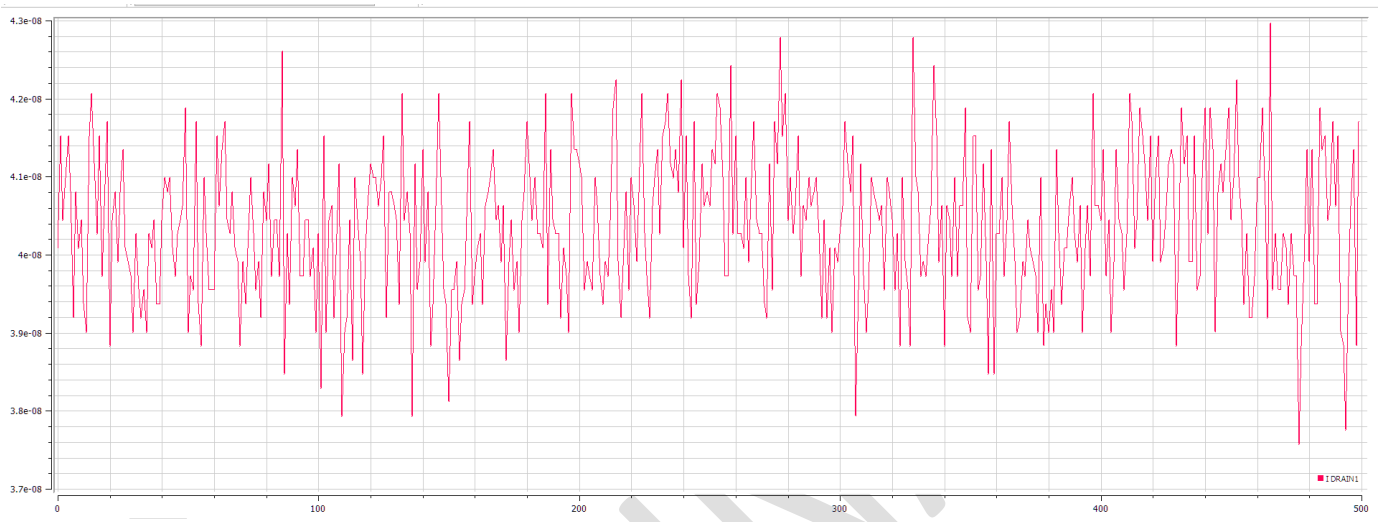


9.1 SMU current noise

The noise of the input current read from an SMU channel depends on:

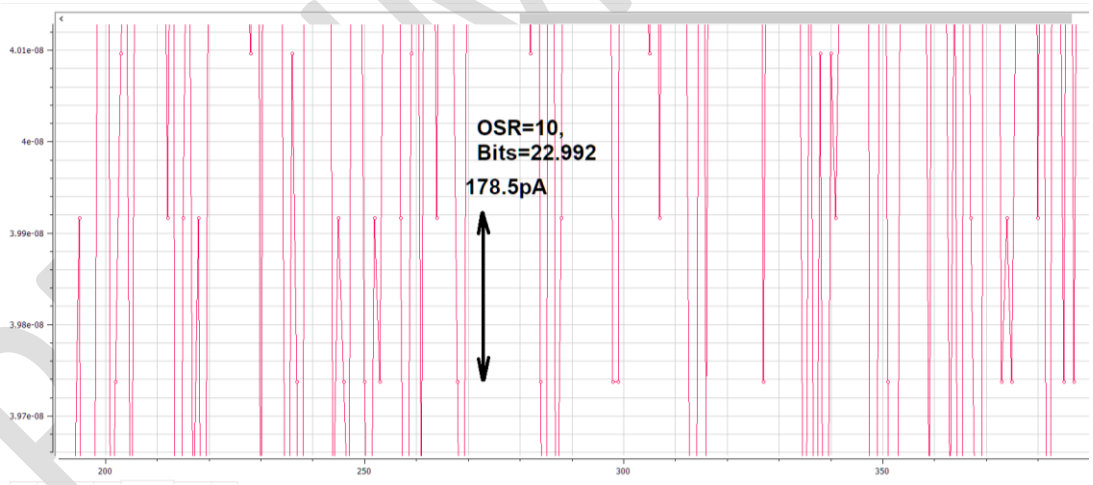
- SMU output voltage (needs to be within recommended range 70mV...4.95V)
- ADC OSR (oversampling rate)

With the default DANFE_REG_OSR =10 (oversampling 20480) and Vout set to 2.5V and no load, we get the noise as pictured below:

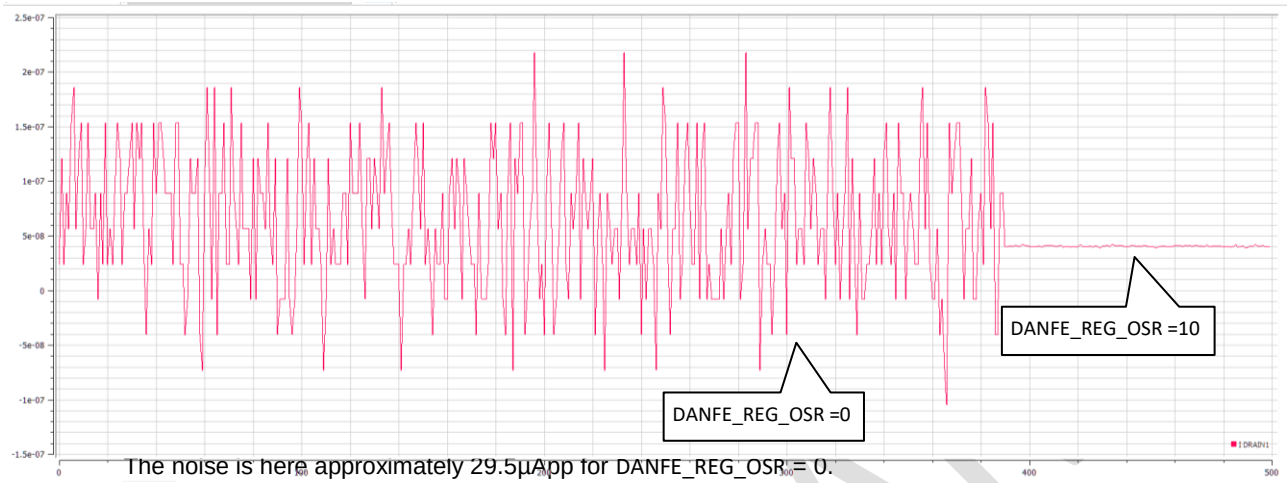


Typical noise is here 6.8nApp. Zooming in we can see the bit levels at 178.5pA which corresponds to 23bits as expected.

The noise of 6.8nApp corresponds to log(6.8n/178.5p)/log(2) = 5.25 bits.

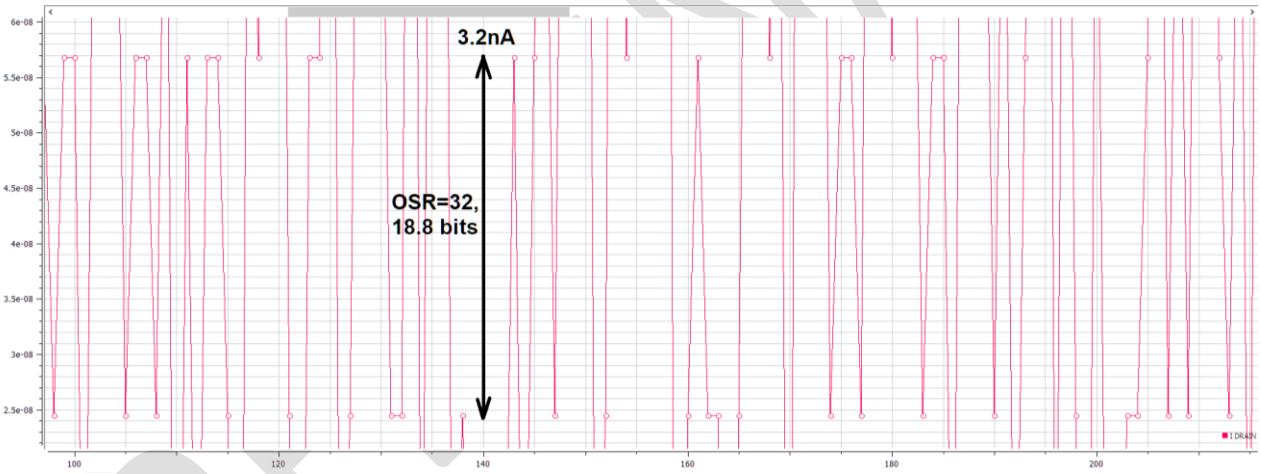


Setting the DANFE_REG_OSR to 0 (oversampling 32) we get a different view:



The noise is here approximately 29.5µApp for DANFE_REG_OSR = 0.

Zooming in on the bit noise and calculating the number of bits we get *log(*1.489mA/3.2nA*)/log*(2) =18.8 bits



9.2 BIAS channels

Bias channels can go from -5V to +5V, and if the set voltage is within a 0.2V margin of the power rails the error is lower than 25mV.
Output current capabilities has not been investigated, but the capacity expected from the LMP7702 used on the output is <20mA.