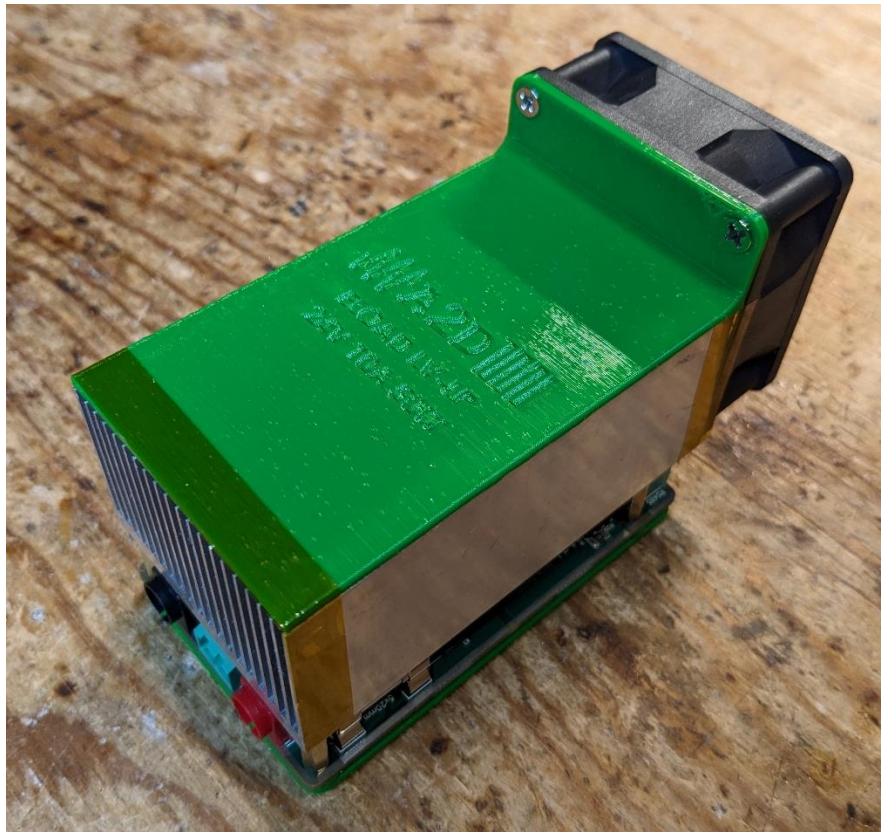




Eload LV-LP Spec Sheet



Description

This board was designed as a simple electronic load for discharging batteries for computer-controlled single-cell tests. The goal was to do 1 thing pretty well – constant-current discharging for Low-Voltage, Low-Power cells.

I did not add extra features such as a high output slew rate or extra measurement ports, as this will be controlled by a computer that can have other devices hooked up for that (such as the A2D 4ch Isolated ADC). There is no front-panel interface either to keep complexity and cost to a minimum, and all control happens through a SCPI interface on the USB port. Examples to get up and running with a python script are provided (see the Communication section).

Specifications

Channels	1
Eload Voltage Input Range	0-24V
Eload Current Range	0-10A
Eload Current Input Connector	5.08mm pitch 2-pin pluggable terminal block, and 4mm banana plugs
Eload Power Input	50W max continuous *must be managed by user
Power - Input Voltage	24V
Power - Input Source	3.81mm pitch 2-pin pluggable terminal block
Power - Input Current	TBD
Communication Interface	SCPI (through USB COM Port), RS485
RS485 Connector	3.81mm pitch 2-pin pluggable terminal block Up to 32 boards connected to a single USB interface

Included Parts

- Eload
- 3x 3.81mm pitch 2-pin pluggable terminal block for RS485 and power input
- 1x 5.08mm pitch 2-pin pluggable terminal block for eload input
- USB-A to USB-C cable

Communication

SCPI interface is available on the USB port. This device can interface with Python through PyVisa. I have created a python library for this device as well as a GUI for calibrating it automatically using another programmable multimeter and power supply.

GUI and SCPI example are available here: https://github.com/mbA2D/Test_Equipment_Control

RS485 communication interface is available (though not currently supported in the firmware). It can be used to chain up to 32 boards together with a single USB interface. The GND of all the boards (including the one powered by USB) must be connected as the RS485 interface is not isolated.

Firmware is available on Github. You are free to modify firmware and upload your own through the serial header pins and a BOOT0 jumper available (there is no USB bootloader in the firmware provided).

SCPI Commands

The device will show up as a COM port on the computer. You can use PyVisa to connect to it using the PySerial backend. Improper commands or parameters will not have any negative effects, they will simply be ignored by the board and will do nothing unless type conversion fails.

<ch> parameter

For all the commands below, the <ch> parameter is an integer that specifies which channel of the A2D Eload to use. Each A2D Eload only has 1 channel, but multiple devices can be connected together with an RS485 bus. Each device must first be assigned an RS485 address over the SCPI interface from the USB. The RS485 address assigned is the channel of the device minus 1. The device connected to the computer gets the 'base' RS485 address of 0, and its channel number <ch> is 1. When adding a second device over RS485, set its RS485 address to 1. The channel number parameter <ch> for the second device will be 2.

PyVisa Settings

The following connection settings should be used for the PyVisa connection:

read_termination	'/r/n'
write_termination	'/n'
baud_rate	115200
query_delay	0.001

Once connected, you can send commands and read responses from the instrument using PyVisa's *query*, and *write*.

*IDN?

Returns the instrument identification: Manufacturer, Model, Serial Number, Firmware Version

Syntax	*IDN?
Response Example	A2D Electronics,Eload,1234,V1.0.0
PyVisa Example	instrument_idn = isnt.query('*IDN?')

*RST

Resets the instrument. Turns off the LED, relay, fan, sets target current to 0, and sets the active calibration values to the values stored in EEPROM.

Syntax	*RST
PyVisa Example	isnt.write('*RST')

MEAS:VOLT <ch>?

Returns the voltage at the power supply input (the 24V input that powers the eload, not the eload's input voltage). The active calibration values are used to convert the voltage measured at the ADC to the voltage at the power supply input.

Syntax	MEAS:VOLT <ch>?
Response Example	23.456

PyVisa Example	<code>ch1_voltage = float(isnt.query('MEAS:VOLT 1?'))</code>
----------------	--

MEAS:TEMP <ch>?

Returns the temperature of the thermistor (glued to the outside of the MOSFET case).

Syntax	MEAS:TEMP <ch>?
Response Example	25.12
PyVisa Example	<code>ch1_temp = float(isnt.query('MEAS:TEMP 1?'))</code>

MEAS:VOLT:ADC <ch>?

Returns the voltage at the ADC (power supply input voltage with voltage divider) for the specified channels, used to calibrate the measurement.

Syntax	MEAS:VOLT:ADC <ch>?
Response Example	1.2345
PyVisa Example	<code>ch1_adc_voltage = float(isnt.query('MEAS:VOLT:ADC 1?'))</code>

INSTR:LED <ch,state>

Turns the LED on or off according to *state*.

Syntax	INSTR:LED <ch>,<state>
<state>	Integer from 0-1. 0 turns the LED off. 1 turns the LED on.
PyVisa Example	<code>inst.write('INSTR:LED 1,1')</code>

INSTR:LED <ch>?

Returns the LED state.

Syntax	INSTR:LED <ch>?
Response Example	1
PyVisa Example	<code>led_state = inst.query('INSTR:LED 1?')</code>

INSTR:FAN <ch,state>

Turns the fan on or off according to *state*.

Syntax	INSTR:FAN <ch>,<state>
<state>	Integer from 0-1. 0 turns the FAN off. 1 turns the FAN on.
PyVisa Example	<code>inst.write('INSTR:FAN 1,1')</code>

INSTR:FAN <ch>?

Returns the fan state.

Syntax	INSTR:LED <ch>
Response Example	1

PyVisa Example	fan_state = inst.query('INSTR:FAN 1?')
----------------	--

INSTR:RELAY <ch>,<state>

Turns the relay on or off according to *state*. Also sets the current target to 0.

Syntax	INSTR:RELAY <ch>,<state>
<state>	Integer from 0-1. 0 turns the relay off. 1 turns the relay on.
PyVisa Example	inst.write('INSTR:RELAY 1,1')

INSTR:RELAY <ch>?

Returns the relay state.

Syntax	INSTR:RELAY <ch>?
Response Example	1
PyVisa Example	inst.write('INSTR:RELAY 1')

CAL:V:RST <ch>

Sets the active calibration values for the specified channels to the default values.

Syntax	CAL:RESET <ch>
PyVisa Example	inst.write('CAL:RESET 1')

CAL:V:SAV <ch>

Saves the active calibration values for the specified channels to EEPROM. Upon reset (*RST) and power-up, the calibration values stored in EEPROM are set as the active calibration values.

Syntax	CAL:SAVE <ch>
PyVisa Example	inst.write('CAL:SAVE 1')

CAL:I:RST <ch>

Sets the active calibration values for the specified channels to the default values.

Syntax	CAL:RESET <ch>
PyVisa Example	inst.write('CAL:RESET 1')

CAL:I:SAV <ch>

Saves the active calibration values for the specified channels to EEPROM. Upon reset (*RST) and power-up, the calibration values stored in EEPROM are set as the active calibration values.

Syntax	CAL:SAVE <ch>
PyVisa Example	inst.write('CAL:SAVE 1')

CAL:V <ch>?

Returns the gain and offset calibration for the requested channels. Return is <offset>,<gain>.

Syntax	CAL:V <ch>?
Response Example	0.004,2.81562
PyVisa Example	cal_values = [float(val) for val in inst.query_ascii_values('CAL:V 1?')]

CAL:I <ch>?

Returns the gain and offset calibration for the requested channels. Return is <offset>,<gain>.

Syntax	CAL:I <ch>?
Response Example	0.004,2.81562
PyVisa Example	cal_values = [float(val) for val in inst.query_ascii_values('CAL:I 1?')]

CAL:V <ch,m1,a1,m2,a2>

Generates new calibration values for the selected channel and sets them as the active calibration values. Calibration is done with a linear calibration with offset and gain values.

Calibration points should be chosen relative to the range you want it to be accurate in. If you want to measure in a range of 2.5 to 4.2V for battery testing, then choose your calibration points to be near the two edges of that range.

Syntax	CAL:VOLT <ch,m1,a1,m2,a2>
<m1>	Float. The measured voltage at the ADC (MEAS:VOLT:ADC <ch>?) when applying the 1 st calibration point to the input.
<a1>	Float. The actual voltage applied to the input when measurement <m1> was taken. This could be from a calibrated DMM or a known voltage reference.
<m2>	Float. The measured voltage at the ADC (MEAS:VOLT:ADC <ch>?) when applying the 2 nd calibration point to the input.
<a2>	Float. The actual voltage applied to the input when measurement <m2> was taken. This could be from a calibrated DMM or a known voltage reference.
PyVisa Example	inst.write('CAL:VOLT 1,1.02,2.34,1.98,4.34')

CAL:I <ch,m1,a1,m2,a2>

Generates new calibration values for the selected channel and sets them as the active calibration values. Calibration is done with a linear calibration with offset and gain values.

Calibration points should be chosen relative to the range you want it to be accurate in. If you want to measure in a range of 0.1A to 5.0A for battery testing, then choose your calibration points to be near the two edges of that range.

Syntax	CAL:VOLT <ch,m1,a1,m2,a2>
<m1>	Float. The voltage that the DAC is programmed to (CURR:CTRL <ch>?) when the 1 st current is measured with a multimeter.

<a1>	Float. The actual current that is being drawn by the eload when measurement <m1> was taken, as measured by a calibrated multimeter.
<m2>	Float. The voltage that the DAC is programmed to (CURR:CTRL <ch>?) when the 2 nd current is measured with a multimeter.
<a2>	Float. The actual current that is being drawn by the eload when measurement <m2> was taken, as measured by a calibrated multimeter.
PyVisa Example	inst.write('CAL:CURR 1,1.02,2.34,1.98,4.34')

CURR <ch>?

Returns the current that the selected channel is programmed to. Note that this is not the current measured across the shunt resistor, but the target current set by the DAC.

Syntax	CURR <ch>?
Response Example	1.23
PyVisa Example	current = inst.query('CURR 1?')

CURR <ch,val>

Sets the target current on the specified channel.

Syntax	CAL <ch>?
<val>	Float. Target current that the output will be set to.
PyVisa Example	inst.write('CURR 1,1.23')

CURR:CTRL <ch>?

Returns the voltage that the DAC is set to (controls the current target).

Syntax	CURR:CTRL <ch>?
Response Example	0.54
PyVisa Example	current_ctrl_voltage = inst.query('CURR:CTRL 1?')

INSTR:RS485?

This command has no channel parameter. It is only available if communicating with this device over USB.

Returns the RS485 address of this device.

Syntax	INSTR:RS485?
Response Example	0
PyVisa Example	rs485_addr = inst.query('INSTR:RS485?')

INSTR:RS485 <addr>

This command has no channel parameter. It is only available if communicating with this device over USB.

Sets the rs485 address for this device.

Syntax	INSTR:RS485 <addr>?
<addr>	Integer. 0 to 31, inclusive.
PyVisa Example	inst.write('INSTR:RS485 0')

INSTR:RS485:SAV

This command has no channel parameter. It is only available if communicating with this device over USB.

Saves the current rs485 address to EEPROM.

Syntax	INSTR:RS485:SAV?
PyVisa Example	inst.write('INSTR:RS485:SAV')

INSTR:KICK <ch>

Kicks the watchdog for the specified channel. Use this command to keep the output alive when no other SCPI commands are being sent. The watchdog is set for a 10s timeout.

Syntax	INSTR:KICK <ch>?
PyVisa Example	inst.write('INSTR:KICK 1')

Firmware and Arduino IDE Settings

Firmware for this board is under the following GitHub repository:

https://github.com/mbA2D/A2D_Eload

The STM32 board files will need to be installed in the Arduino IDE under the “Additional Boards Manager URLs” in the File->Preferences menu. Instructions are available here:

https://github.com/stm32duino/Arduino_Core_STM32/wiki/Getting-Started

The following settings should be used in the Arduino IDE to upload any modifications to the firmware:

Board	Generic STM32F1 series
Debug symbols and core logs	None
Optimize	Smallest (-Os default)
Board part number	BluePill F103C8
C Runtime Library	Newlib Nano (default)
Upload method	S32CubeProgrammer (Serial)
USB support (if available)	CDC (generic 'Serial' supersede U(S)ART)
U(S)ART support	Enabled (generic 'Serial')
USB speed (if available)	Low/Full Speed

Hardware and Schematic

The full schematic can be found here: https://a2delectronics.ca/wp-content/uploads/2024/06/Schematic_A2D_Electronics_Eload_LVLP_V1_2024-03-10.pdf

Programming

This board was designed with a similar schematic to a standard ‘Blue Pill’ board while fixing common issues (i.e. USB D+ resistor among others). This should make it easy for others to understand the schematic and to develop custom firmware for it should they want to. The H2 header connects BOOT0 to 3V3 to enter bootloader mode.

Power Architecture

The board is split into 2 isolated sections, control side and eload side. The control side is powered from the USB input, and has an STM32 micro, USB interface, RS485 interface, a relay and a fan. The eload is powered from an isolated 24V to +/-12V converter, and several rails are derived with LDOs for the DAC and opamp control circuits. An isolated I2C transceiver is used for the micro to communicate with the DAC on the isolated side. There is no feedback from the eload side to the control side, so whatever current is set in the DAC is what the load will be regulating to.

The control side measures the voltage of the 24V rail, and if it is not within 20-26V, then the output is not allowed to turn on. If the voltage falls out of this range while the output is on, then the output will be turned off.

ESD and Reverse Polarity Protection

A unidirectional TVS diode (SMCJ24A) is across the fused eload input and 24V power terminals, such that if the input is connected in reverse polarity, the diode will conduct. This will blow the fuse if the power source can provide enough current to blow it.

Fan and Temperature Control

The control side uses a thermistor to check the temperature of the MOSFET case. It will use the fan to regulate the temperature to 40C using ON/OFF control with some hysteresis. The "INSTR:FAN" SCPI command will only affect the state of the fan until the next time the temperature control loop runs (at 1Hz). If the temperature of the MOSFET case reaches 80C, then output will be turned off to protect the MOSFET.

Eload

In an effort to keep this board as simple as possible, there is no measurement circuitry on the board. So the current setpoint is assumed to be perfectly regulated when using the "MEAS:CURRENT?" SCPI command. This also means that the amount of power the eload is sinking cannot be measured as the eload voltage is not measured. It is up to the user to limit the eload voltage so the power stays within the 50W max for the set current. The device will turn off the eload in the case that the MOSFET case temperature gets too high, but this is not guaranteed to protect under very high power levels as the die will heat up much quicker than the case.

Watchdog

A 10s watchdog timeout has been implemented on this board. If a valid SCPI command has not been received within 10s, then the output will turn off. This is to protect any load such as a battery from fully discharging if the control computer fails.

RS485 Interface

The solder bridges on the board go to GPIO pins of the STM32 microcontroller and could be used to set an address for RS485 communication if you want to modify the firmware for it. Currently setting the RS485 address is only available through the USB communication port. 5 solder bridges can be set for up to 32 devices. TX3/RX3 of the micro are used for the RS485 transceiver. The RS485 bus operates differentially with a 3V3 power supply. A 120 Ohm termination resistor should be added to both ends of the bus. Putting a jumper over the middle pins of H3 connects a termination resistor on the board. The interface is not isolated, so all boards connected with RS485 must have the same GND reference on their power input. The outer pins of H3 will connect RS485 bias resistors and should only be connected on the base device. These are to provide a fixed voltage on the RS485 lines so that noise is not recognized as a start of frame bit.

V1.0 Hardware Errors

Please note the following mistakes on the V1.0:

1. There is no bulk decoupling capacitor on the 24V input supply. A 10uF or greater input capacitor (C38) is placed across D2 on the input supply rails.
2. +/-15V on the schematic is mistakenly labelled. A +/-12V converter is being used for U11.