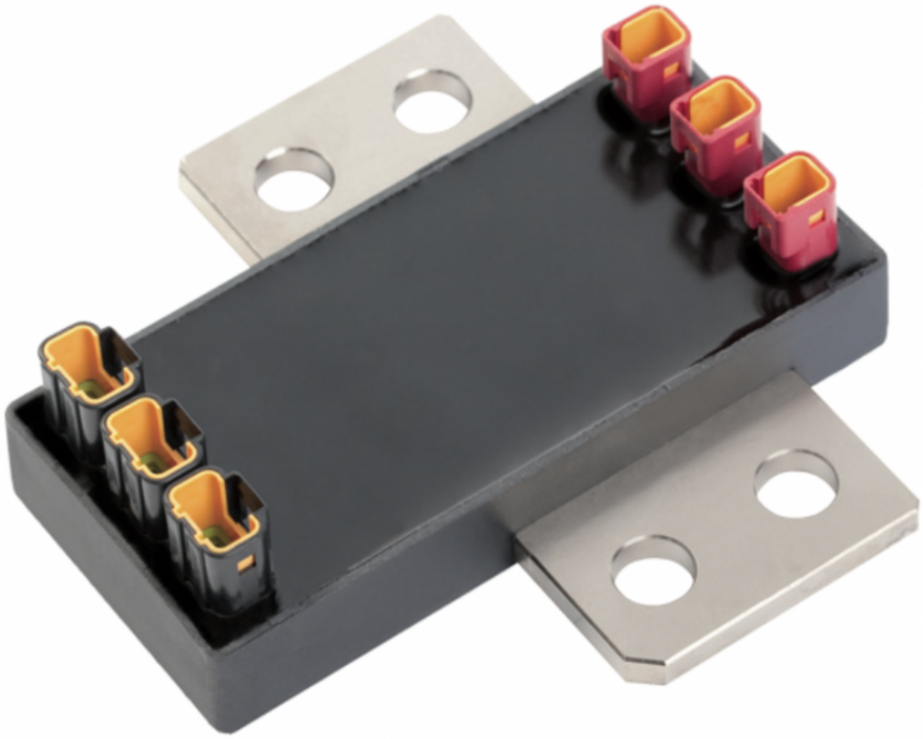
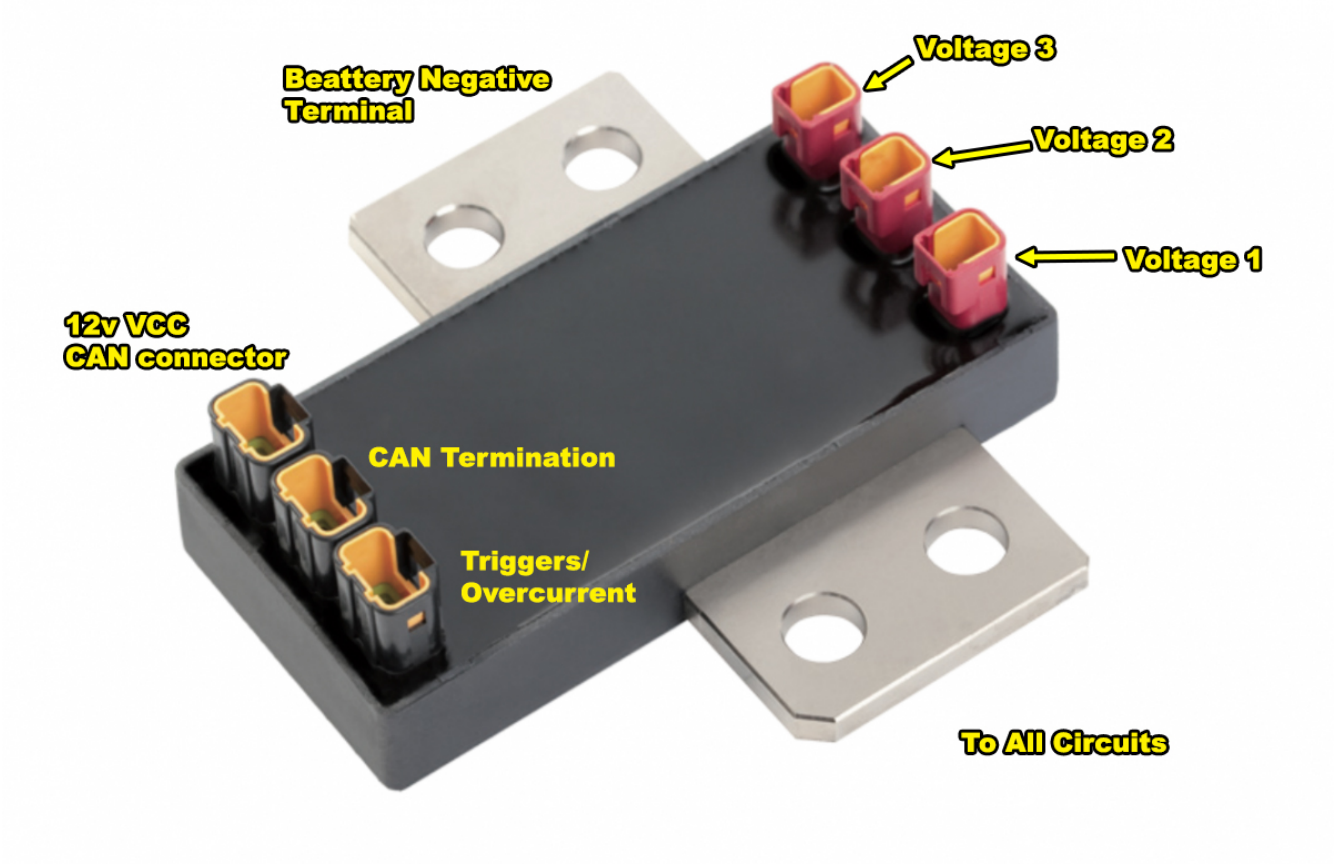
ISAscale
High Current Sensor Module
For
Electric Vehicles



INTRODUCTION

This User Manual describes use and operation of the ISAscale IVT-Modular high current sensor module for Electric Vehicles.

The IVT Module provides high accuracy temperature compensated shunt measurement of electrical current at levels up to 2500 amps on some models. EVTV makes available a model for 300A as well as a 1000 amp module.



Beyond current, the IVT Module measures up to three voltages of up to 800v between the battery pack negative terminal and the measurement point.

These measurements are galvanically isolated from the 12v power used and the Controller Area Network CAN communications bus where it reports these measured values.

As such, it performs as a very accurate mini Battery Monitoring System, reporting three pack-segment voltages and overall current with high accuracy over the CAN communications network.

To provide easy access to the basic functions of the IVT Module, EVTV also developed the ISA library for the Arduino Due. This library makes it easy to incorporate the functions of the IVT module in C++ programs written for an Arduino Due with a CAN port connection.

SPECIFICATIONS

Operating Voltage: 12vdc - 5.5-16.0vdc
Operating Temperature: -40 to 85C
Current Consumption: 20-95 ma
Startup time: 400ms max, 350 ms typical
Isolation: 4kv
CAN Communications: 250k, 500k, 1000k. Default 500k 2.0A
Max number of units on CAN bus: 6

CURRENT MEASUREMENT

Nominal Current Range: +/- 1000 amps
Overcurrent Measurement Range: +/- 12,200 amps
Extended load: 1400 amps for 30 seconds; 2000 amps for 10s.
Intiial accuracy: +/-0.1% of reading
Total Accuracy: +/-0.4% of reading
Offset: 125ma
Linearity: 0.01% of range
Noise: 70ma
Resolution: 47ma

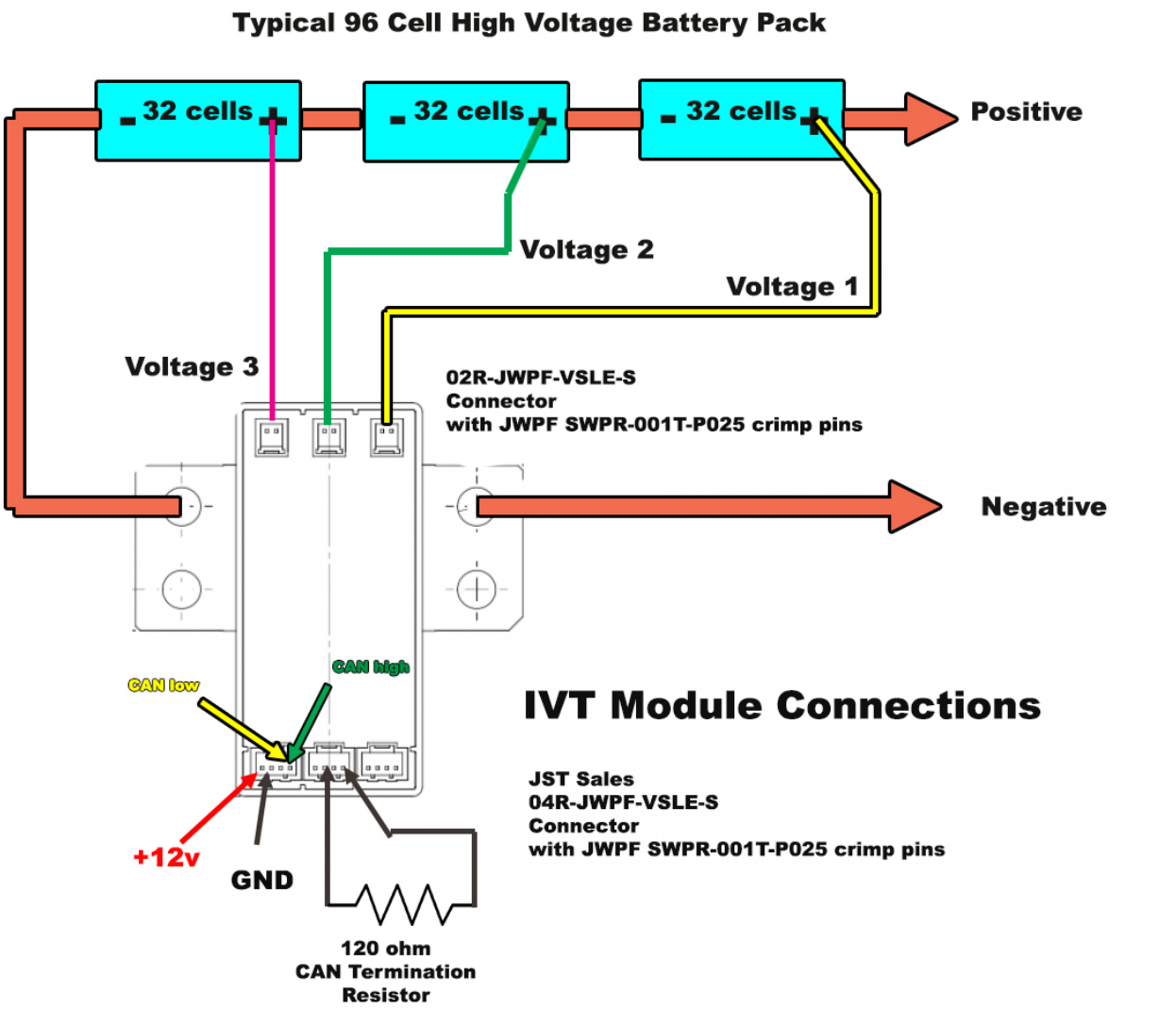
HIGH VOLTAGE MEASUREMENT

Nominal Measurement Range: +/-600v
Extended Range: +/-800v
Initial Accuracy: 0.1% reading
Offset: 100 mv
Linearity: +/-0.01% of range
Noise: 60mv
Resolution: 30mv

Temperature corrected.

CONNECTIONS

Connections to the IVT Module are depicted below:



1. Control connector is the left most 4-pin connector. Left to right are 12v, ground, CANLO and CAN HI.

2. The module is NOT CAN terminated. If you need to terminate, connect 120 ohm resistor between pins 3 and 4 of the center connector.

3. Voltage connectors are two-pin connectors but the pins are shorted internally. Either may be used.

4. Voltages should be connected in the order depicted with Voltage 1 at the most positive point of the battery pack. This is important. Power calculations are based on the voltage at Voltage1 and so should be the total pack voltage on that input.

5. The shunt is directional. Currents will be read as negative when charging and positive when discharging as depicted. You may reverse this if desired.

6. The module should be mounted between the pack negative terminal and ALL loads to ensure all currents into and out of the pack are measured.

ISA LIBRARY

The ISA library was written to allow easy access to the functions available in the IVT Module. But this library is fairly narrow in application.

It is written for the Arduino Due 84MHz SAM3X multicontroller with CAN port transceiver hardware installed on the native CAN0 port of the SAM3X chip and an EEPROM chip for storing variables between sessions. The EVTV Due Microcontroller has these features.

For other applications, more detailed descriptions of module function and configuration can be found in the ISAscale IVT Module Data Sheet and by examining the code in this ISA library.

To use this library, you must first include it in your Arduino program. You must also include the due_can library http://github.com/collin80.

```
#include <ISA.h>
#include <due_can.h>
```

Once included, these libraries allow creation of an object module specifically for the IVT Module. Instantiate this object in the following manner.

```
ISA Sensor;
```

This creates the object **Sensor** of class **ISA**. The IVT Module variables and functions will thereafter be available as **`Sensor.Voltage1`** for example or **`Sensor.Amperes`** or **`Sensor.AH`**, etc. Of course, the object name can be any C++ legal name combination.

The object module Sensor, is of course of "class" ISA. This is part of the object module hierarchy of the C++ language. However, it is also of class **`CANlistener`** from the **`due_can`** library.

```
class ISA : public CANListener
```

This takes advantage of a feature of the C++ programming language termed "inheritance" in that we can create an object of one class, in this case

CANlistener, and then add some features to it to create a "child" class that "inherits" all the features of the parent, but then has its own as well.

So ISA is actually an object of class CANlistener that has been enhanced with new features to make the child class ISA.

This bit of esoterica is interesting because it allows us to an interesting, thing with the ISA module.

Sensor.begin(0,500);

If we place this line in our setup section, we initialize Sensor to use CAN 0 at a speed of 500 kpbs (500,000 bits per second).

The ISA hardware sends CAN messages in the 0x520 to 0x529 message ID range. The ISA object module, as a CANlistener submodule, sets an interrupt to trap all 0x520-0x529 messages and automatically process them to extract current, voltage and power data.

You can still use CAN 0 in your other program code, but it is already initialized at 500 kbps. Note too that filter 6 is already in use by the ISA object and should NOT be redefined in your program.

Bottom line, in this way, you don't have to decode CAN messages at all to get the info out of an ISA module.

ISA KEYWORDS

When accessing ISA keywords and variables, case IS important.

Amperes

Certainly the primary function of the IVT Module is to measure current. This is available continuously at the variable Amperes, a floating point variable.

```
float current=Sensor.Amperes;
```

Note that the IVT Module is capable of very high accuracy and resolution at very high current levels. Smaller values can be accessed as **Sensor.milliamps.**

Voltage

This variable holds the voltage last measured between the shunt itself and the Voltage1 input terminal. This should be the most positive terminal of your battery pack as this voltage is used for all power calculations in the IVT Module. As such it represents total pack voltage

```
float packvoltage=Sensor.Voltage;
```

VoltageHI

This variable holds the highest voltage measured since the last reset.

```
float packvoltageHigh=Sensor.VoltageHI;
```

VoltageLO

This variable holds the lowest voltage measured since the last reset. But it does ignore the first 50 frames after reset.

```
float packvoltageLowest=Sensor.VoltageLO;
```

Voltage1

This variable holds VOLTAGE minus the sum of the voltages measured at Voltage 2 and Voltage3. As such, this is the voltage of the most positive SEGMENT of the pack measured.

```
float V1=Sensor.Voltage1;
```

Voltage1HI

This variable holds the highest voltage measured since the last reset.

```
float V1High=Sensor.Voltage1HI;
```

Voltage1LO

This variable holds the lowest voltage measured since the last reset. But it does ignore the first 50 frames after reset.

```
float V1Lowest=Sensor.Voltage1LO;
```

Voltage2

This variable holds VOLTAGE between the second point of the pack and the shunt - minus the voltage measured at Voltage 3. As such, this is the voltage of the middle segment of the pack.

```
float V2=Sensor.Voltage2;
```

Voltage2HI

This variable holds the highest voltage measured since the last reset.

```
float V2High=Sensor.Voltage2HI;
```

Voltage2LO

This variable holds the lowest voltage measured since the last reset. But it does ignore the first 50 frames after reset.

```
float V2Lowest=Sensor.Voltage2LO;
```

Voltage3

This variable holds the voltage last measured between the shunt itself and the Voltage3 input terminal. This would be the lowest segment of the pack.

```
float V3=Sensor.Voltage3
```

Voltage3HI

This variable holds the highest voltage measured since the last reset.

```
float V3High=Sensor.Voltage3HI;
```

Voltage3LO

This variable holds the lowest voltage measured since the last reset. But it does ignore the first 50 frames after reset.

```
float V3Lowest=Sensor.Voltage3LO;
```

Note that many packs contain a number of cells easily divisible by three while other packs are only divisible by two. In this way, you can have equal pack segments for almost any sized pack.

And it is important that these voltages each represent an EQUAL number of cells. In our example diagram, we should three taps each of 32 cells.

This is important and actually one of the key elements of this device. You can detect individual cell failures quite easily and in fact somewhat before they occur with just a few pack segment measurements by COMPARING those values, and specifically comparing them under HIGH LOAD conditions.

When lithium cells are put under load, their terminal voltages diminish or "sag" as a function of how much current they are asked to produce. However, in a series pack of lithium cells, the current through all cells will be identical and so the resulting voltage from one cell to another, while it will actually vary, should do so at a very close level. Differences between one pack segment and another greater than just a volt or two are signs that at least one cell in the lower measured segment is struggling to produce that current and will likely fail in the near future.

Specifically which cell is failing is a maintenance function of no particular import while driving the car. But using these voltage inputs, you can easily detect variations between groups of cells that warrant investigation.

Power

This variable holds the current instantaneous power discharging from the battery or charging into it (negative). This is Amperes* Voltage and is expressed in kiloWatts (1000 watts). The voltage at Voltage input 1 is used for this calculation.

```
float power=Sensor.KW
```

This is also available as Sensor.watt representing smaller values.

KiloWatt-Hours

Power is expended from the battery pack in a constantly varying manner and both voltage varies with discharge as well as current. Additionally, as the State of Charge diminishes, so does the pack voltage. Because of this, the current necessary to produce the same amount of power will increase.

The most accurate way to measure pack capacity and discharge then is with the amount of power, expressed in watts, over time, normally expressed in hours. This kiloWatt-Hour variable then holds the cumulative power discharged from the pack or charged into the pack over time.

```
float totalDischarge=Sensor.KWH
```

Note too that this value accumulates over time and use as long as 12v power is applied to the IVT Module. It is also periodically written to EEPROM to be persistent when power is removed from the module. It reloads automatically when power is again applied. To zero, it must be manually reset

```
Sensor.KWH=0;
```

This value is also available as Sensor.wh representing smaller values.

Ampere-Hours

An alternate way of measuring power into and out of the battery over time is to use current only. One Ampere of current flowing from the battery for 60 minutes would then be referred to as one Ampere-Hour of energy.

```
float ampDischarge=Sensor.AH
```

pack. Note too that this value accumulates over time and use and is also periodically written to EEPROM. So similarly you must manually zero it to reset:

```
Sensor.AH=0;
```

This is also available as Sensor.As representing ampere-seconds.

OTHER FUNCTIONS

There are a few housekeeping functions available when using the IVT Module.

INITIALIZATION

The IVT Module has a wealth of other features allowing you to measure things only when a hardware trigger is applied, or to produce a hardwire output in the event of an overcurrent condition. Different applications require different measurement functions, etc. Configuration can become somewhat detailed and is described in the ISAscale IVT Module Data sheet available at the EVTV web site.

For the ISA library, we have already worked out a configuration to produce the functions included in the library. Note that this initialization only needs to be performed once and is persistent from use to use. But if you receive a new IVT Module, you will likely have to perform this function once to get proper readout of the variables discussed.

Once an ISA object has been created:

```
Sensor.initialize();
```

There may be a delay of a few seconds while this function is performed. Again, this does NOT need to be run each time you use the device or power up a program referring to it.

FRAMECOUNT

The ISA library keeps a running total of CAN message frames received and processed in the variable framecount.

```
int frames=Sensor.framecount;
```

DEBUG

Normally, the ISA library does not print anything out the USB port itself. However, there are two separate debug levels that can be set to observe internal data processed by the ISA library.

These variables will normally be set to 0. But if set to 1, will cause the ISA library to print out ASCII data via the USB port showing various details of operation for troubleshooting purposes.

```
Sensor.debug=1;
```

This will cause ISA to display individual CAN frames received or sent via the CAN port. A timestamp is provided along with the message ID and eight data bytes of the received or transmitted message.

```
00:00:34.834 Rcvd ISA frame: 0x523 02 03 08 34 00 00 00 00
00:00:34.834 Rcvd ISA frame: 0x523 02 03 08 34 00 00 00 00
00:00:34.840 Rcvd ISA frame: 0x526 05 01 00 00 00 00 00 00
00:00:34.840 Rcvd ISA frame: 0x526 05 01 00 00 00 00 00 00
00:00:34.870 Rcvd ISA frame: 0x521 00 03 0C 00 00 00 00 00
00:00:34.870 Rcvd ISA frame: 0x521 00 03 0C 00 00 00 00 00
00:00:34.884 Rcvd ISA frame: 0x522 01 0D A4 67 00 00 00 00
00:00:34.884 Rcvd ISA frame: 0x522 01 0D A4 67 00 00 00 00
00:00:34.886 Rcvd ISA frame: 0x527 06 08 AC FB FF FF 00 00
```

A second debug will cause display of data calculated from those frames.

```
Sensor.debug2=1;
```

```
KiloWattHours: -0.01  Watt Hours: -8 frames:15355
Voltage: 26.54 vdc Voltage 1: 13.23 vdc 26539 mVdc frames:15356
Voltage: 26.54 vdc Voltage 3: -0.04 vdc -43 mVdc frames:15357
Current: 0.01 amperes 7.00 ma frames:15358
```

```
Power: 0 Watts  0.00 kW  frames:15359
Temperature: 32.00 C  frames:15360
Voltage: 26.53 vdc Voltage 1: 13.22 vdc 26532 mVdc frames:15361
Voltage: 26.53 vdc Voltage 2: 13.35 vdc 13306 mVdc frames:15362
Current: 0.02 amperes 23.00 ma frames:15363
Amphours: -0.31  Ampeseconds: -1104 frames:15364
KiloWattHours: -0.01  Watt Hours: -8 frames:15365
```

TEMPERATURE

The IVT Module uses temperature to correct current calculations through the shunt. This shunt temperature is also available:

```
int temp=Sensor.Temperature;
```

STOP

You can halt operation of the ISA lib at any time with the STOP command:

```
Sensor.STOP();
```

START

Operation may be resumed with the START command:

```
Sensor.START();
```

RESTART

RESTART completely resets and restarts the IVT Module. This has the effect of resetting AH and KWH to zero.:

```
Sensor.RESTART();
```