Detailed guide for assembling and using your MightyWatt

Board revision: 2.5
Minimum firmware version: 2.5.7 (main), 2.5.7 (calibration)
Windows control program version: 1.3.1.0
Guide revision: I (2017-02-12)

Assembly

1) Solder the pin headers.
2) Put a thin but continuous layer of thermal grease on the transistor. Use really good, non-electrically-conductive thermal grease. Many CPU thermal greases have high performance but low thermal stability. Remember that temperatures at the MOSFET will be much higher than those at the CPU.
3) Assemble the heatsink. Start with the screws upside down, with their heads lying on a desk. Put the finger guard, then the fan (label facing up so it will be facing the transistor), heatsink (fins facing down), brass (5 mm) spacers and plastic washers (one per screw). Observe where the cable from the fan is. Then, attach the main board, transistor interfacing the heatsink, in such a way that the cable from the fan is in the same corner as the header on the board. Put another layer of plastic washers and finally, the nuts. Screw tightly so that the interfacial contact of the MOSFET and the heatsink is good.
4) If you are using Arduino Due, you have to set its AREF (BR1) jumper to EXT (external). Unfortunately, this means soldering on the Arduino Due board.

Calibration

Your MightyWatt uses a precision DAC but there are offsets that should be removed by calibration. Because MightyWatt uses the Arduino's ADC, each combination of a MightyWatt and an Arduino should be calibrated separately. If you strive for the best precision, calibrate MightyWatt every year. The calibration sketch is for Arduino Zero (M0/M0 Pro), Uno and Due. However, you have to select the board also in the sketch.

You will need one multimeter (voltmeter/ammeter) and an adjustable power supply, preferably capable of delivering voltage and current to match the range of MightyWatt. If you have a less powerful supply, do not worry because the linearity is usually quite good.

1) Open the calibration sketch in Arduino IDE.
2) Select the correct board in Arduino IDE in **Tools** → **Board**.
3) Look at the top of the sketch. Uncomment the line that defines your board (ZERO, UNO or DUE).
4) Uncomment section 1 in the calibration sketch.
5) Adjust the **for cycle** in section 1 to match the capabilities of your power supply and the number of points you want to use for calibration (4–10). For example, you have 12-bit DAC (standard) and the MightyWatt range is 10 A but the power supply delivers only 5A maximum, then set the maximum value to 4095/2 = 2047. If you want 6 calibration points, then set the increment to (2047 – 41)/(6 – 1) = 401. Don't go below 1 % of the

range (41) as the lowest value and above 99 % of the range (4054) as the highest value (provided your power supply can deliver it).
6) Upload the sketch to the Arduino.
7) Connect the power supply to the load. Put your ammeter in series with the load.
8) Start the Serial Monitor on the Arduino IDE. It should show two kinds of values: the ADC value and the DAC value plus you should have a reading on your ammeter. Write down the ammeter value and do not close the serial monitor.
9) Send one character (any) to the Arduino via the serial monitor. This will advance one iteration in the **for cycle**, raising the current. Repeat until all points are measured.
10) Put the values into the Excel calibration aid file, worksheet 1. It will show the calibration values, which you will then put into the main sketch.
11) Parts 2 and 3 follow the same principle (comment sections you don't need, uncomment the one you use), so briefly:
12) Part 2: Use the voltmeter in parallel; no current will be flowing in this part. Adjust the voltage on the external power supply; write down the values on the voltmeter and the corresponding ADC value. If you send '0' or '1' on the serial line, it will set the voltmeter gain. So two sets of ADC+voltmeter values are needed. One for the lower range (gain) and one for the high range (no gain). It is best not to overlap the ranges, that is, if you have ranges 5 V/30 V, do not run the full-scale from 0–30 V but do it from 5–30 V. Put the measured values into the Excel calibration aid, worksheet 2.
13) Part 3: It is better to use the 4-wire connection this time because current will be flowing. Connect the cables from the sense (SENS) terminals to the same point where you connect your voltmeter. Set the supply to the maximum value that MightyWatt can measure (30 V in standard units when using no gain, 5 V for gain) and some current that does not violate the maximum power dissipation (V × I). It will be running in current limited mode but high current is not necessary.
14) Measure voltage while decreasing the DAC value (load is in constant voltage mode) by sending any character over serial line. Do this twice, once for gain and once for no gain (in load's voltmeter; see calibration sketch). Again, you will have two sets of data, both containing a DAC value and the true voltage (from the voltmeter). Put them into the Excel calibration aid, worksheet 3, to get the calibration values.

Arduino sketch

The main sketch is for Arduino Zero (M0/M0 Pro), Uno and Due. However, you have to select the board also in the sketch.

1) Open the sketch in the Arduino IDE.
2) Select the correct board in the Arduino IDE in **Tools** → **Board**.
3) Look at the top of the sketch. Uncomment the line that defines your board (ZERO, UNO or DUE). Comment out the other line.
4) If you have calibration values, modify the constant values. You will find all the constants at the top of the sketch (before calling *void* ***setup()***).
5) You might want to save your sketch at this moment; it is configured specifically for your Arduino/MightyWatt combination.
6) Upload the sketch into your board. Use *programming USB* on Arduino Due boards and *native USB* on Arduino Zero (M0/M0 Pro) boards.

Windows control program

MightyWatt can be controlled over the virtual COM port. This is used in the Windows software.

The control is straightforward. First, you need to connect the load to the computer, then start the program and in the **Connection** menu select the Arduino board you are using and then the correct COM port.

Some comments on the program features are below:

Manual control immediately sends the desired setting to MightyWatt. **Stop** button will set the load to zero current. Continuously measured values are displayed in **Values** box.

Watchdog can stop the load in case current, voltage, power or resistance rises above or drops below the set value. **Voltage** can be **sensed** either locally at the power (PWR) terminals or remotely using another pair of cables – from the sense (SENS) terminals. Remote voltage sense (4-wire, Kelvin) effectively removes the voltage drop introduced by cable resistance from measurements.

More complex behaviour can be programmed using the **Program** box. You can select any combination of controllable variables (current, voltage, power, resistance, voltage with inverted phase) and assign it any **value** for any **duration**. Moreover, a condition similar to the watchdog function can be set to each program item. The **skip if** condition will terminate the current program item and jump to the next one. The Watchdog function is independent and will stop the load completely upon tripping.

It is possible to create several program items which will be executed consecutively after you click the **Start** button. You can use either **constant** value or linear **ramp** from an initial value to a final value. The initial value can be copied from **previous** program item. In this case, the last measured value of the preceding program item will be used as the starting value for the next item. You can repeat the program procedure by enabling **loop** and setting it to either a specific amount of loops or run the program in an infinite loop. It is possible to save the current program procedure to a XML file. To do so, click **Program items → Save**. Loading a program procedure can be done in two ways: Either adding the items from a XML file or overwriting existing items with those from a XML file. Adding does not change loop settings, overwriting will also overwrite the loop setting, logging period setting and 2-wire/4-wire mode (if the load is connected at that moment). To add items, select **Program items → Add**. To overwrite items, select **Program items → Replace.** If **Watchdog** is enabled, it is active in program mode as well and will terminate the program if it sets off.

Data can be saved into a simple plaintext file that can be easily imported into Excel. A data file has to be created before logging is possible. Use the **Logging** menu to create a new file. Sampling period can be adjusted in **Logging → Settings**. The shortest true logging period is around 56 ms (17–18 Hz) when set to zero. Logging is activated and deactivated separately for manual control (**Log Manual**) and for program (**Log Program**). Program logging will start after clicking the **Start** button and will terminate when the program finishes. Manual logging will be active when a user program is not running. The time value in logged data is referenced to the time when the file was created. Data is written into the file continuously so you won't lose them in case the program crashes.

When you run the load in 4-wire mode, the cable resistance is not reflected in the maximum power dissipation. The load will actually dissipate less power because some of it is dissipated in the cables. You can also deliberately add a power resistor in series with the load to lower the heat burden of the load. It is possible to compensate for the maximum power dissipation value for this series resistance. Select **Advanced → Series resistance**. A pop-up window will appear that will let you specify the series resistance. The load will now assume that you are dissipating some of the power ($R \cdot I^2$) externally. Use this with caution as wrong values may cause MightyWatt to overheat.

Select **Tools → Integrators and statistics** to show a toolbox that displays statistical information for the measured values and allows the integration of charge and dissipated energy (heat). In the toolbox, press **Start (Stop)** to initiate (halt) the data gathering. Press **Log snapshot** to log the displayed values to the log file. The button is enabled only when a log file is opened. You can enter an additional header for the data in the **User note** text box. Check **Offset log** to offset the logged data to the right so they are in different columns than the regular log data. This may help with the subsequent processing in your favourite spreadsheet application.

Because power dissipation can be high in MightyWatt, it constantly monitors the temperature of the main MOSFET. The temperature displayed is lower than the true junction temperature so a healthy safety margin is preferable. That is why the maximum temperature is set at 110 °C, which corresponds to about 145 °C junction temperature at 25 °C ambient temperature. MightyWatt will automatically stop when overheat is detected.

Additional info

MightyWatt has one orange LED that is on when there is current flowing. The amount of current to trigger the LED depends on the maximum current. For the 10 A version, the LED will turn on at about 50 mA and turn off at about 35 mA. You can change the behaviour of the LED in the sketch.

A note about the inverted phase voltage

MightyWatt is internally operated in either constant current or constant voltage mode. Under constant current mode, the analog feedback closes the main transistor if the current is too high and opens it when it is too low. Under constant voltage mode, the analog feedback closes the main transistor if the voltage is too low and opens it when it is too high. This works fine for most of the loads. However, there can be certain situations and arrangements when the voltage actually increases with the current. With such behaviour, the hardware constant voltage mode would saturate the load, meaning it will be either fully open or fully closed. To make it stable, the phase of the signal must be inverted. Since this is not implemented in hardware, it has to be done by the firmware. The inverted phase voltage mode keeps constant voltage under such conditions by operating the load in constant current mode internally and keeping the constant voltage by software feedback loop. This mode of control is also applied in constant power and constant resistance modes.

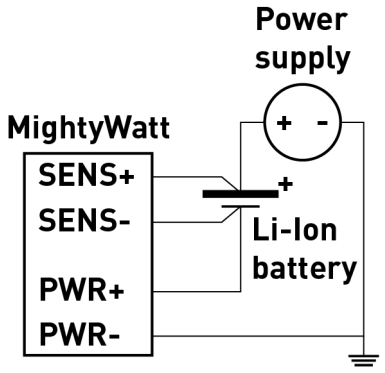

Figure 1: Connection schematic to create a programmable CC/CV Li-Ion battery charger.

An example of when the inverted phase voltage mode is useful is when MightyWatt is used as a Li-Ion battery charger. A generic power supply charges the battery and MightyWatt controls the voltage and current, making a programmable power supply out of a fixed one. Typically, a Li-Ion battery is charged first with constant current (CC) until a certain voltage is reached, which is then kept constant (CV) until the current falls below some small value. The battery is then charged.

Using the setup on Figure 1, the constant current will be maintained in standard constant current mode, the voltage at the battery will be sensed and MightyWatt will switch to constant voltage mode when the voltage reaches a set threshold. Then, the inverted phase voltage mode will keep a constant voltage until the current drops below a set value. A correct program setting is shown in Figure 2.

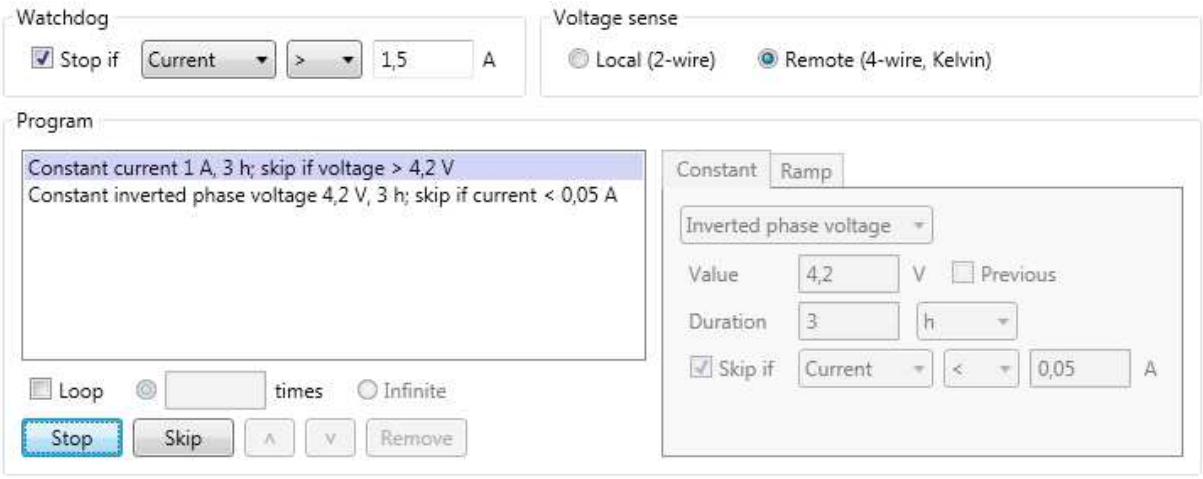

Figure 2: Setting of a CC/CV mode for battery charger. Watchdog is enabled for safety.

A note about the maximum power point tracker

Maximum Power Point Tracker (MPPT) is a feature used predominantly in testing of photovoltaic modules (solar cells). This function will try to find and keep the point on the current-voltage (I-V) curve, where the product of I and V – power – is maximal ($dP/dI = 0$ and $dP/dV = 0$). Since I and V are dependent, the two differentials are equal.

Practically, the user interface has a **Max power point tracker** as a selectable mode in manual and program-constant regimes. A MPPT ramp has no meaning since the tracker should keep maximum power, therefore, in program-ramp, MPPT is not in the menu.

Inside the Arduino sketch, the MPPT algorithm is implemented to use constant-current mode. It is possible to specify the starting current as the entry point for the algorithm. That is why the MPPT mode shows amperes as the unit. If you don't have an estimate for the current, start from zero.

There are many algorithms that can be used to find maximum power point. The one implemented here belongs to the "hill-climb" category, also called "perturb and observe". In each iteration of the control loop, the algorithm will choose between increasing and decreasing the current by 1 LSB. The decision is made from the outcome of two previous actions (current up or current down).

If both actions were the same, then if the power has increased, the next action will be the same. Otherwise it will reverse.

If the actions were different, the algorithm will choose the one which produced the more favourable result; i.e., the one which led to a larger increase or smaller decrease in power.

While the algorithm is intended especially for photovoltaic modules, it can be used for any source of electricity such as a fuel cell or battery. Be aware that maximum power point of a battery or fuel cell usually means a significant drain and therefore low efficiency and a risk of overheating.