EmonLibCM

This *Continuous Monitoring* library provides a means of using advanced features of the Atmel ATMega328P microprocessor to give a way to measure voltage, current and power on a single-phase electricity supply. The physical quantities are measured continuously, the average values are calculated and made available to the host sketch at user-defined intervals.

The library incorporates temperature monitoring using the DS18B20 sensor and pulse counting.

This document has the following main sections:

Key Properties – important points about this library

Using EmonLibCM – a very brief explanation of how to use this library

Application interface – power and energy (including pulse counting)
Application interface – temperature monitoring

List of required supporting libraries

Initial configuration

Calibration notes

Example Sketches

Alphabetical index of Application Interface functions

Key Properties

- Continuous monitoring of one voltage channel and up to 5 current channels.
- Gives an accurate measure of rapidly varying loads.
- Better than 1900 sample sets per second – using 4 channels of an emonTx V3.4 @ 50 Hz
- Calculates rms voltage, rms current, real & apparent power & power factor.
- Accumulates Wh.
- Pulse input for supply meter monitoring.
- Integrated temperature measurement (up to 6 DS18B20 sensors).
- User-defined reporting interval.
- Suitable for operation on a single phase supply at 50 or 60 Hz.
- Can be calibrated for any voltage and current up to a maximum of 32 kVA per input. (Default calibration is for emonTx with 100 A CT & UK a.c. adapter).
- Includes functions to enable on-line recalibration.

Using EmonLibCM

You will need a sketch that globally

1. Includes the emonLibCM library and any others that might be needed.
2. Declares & defines arrays to receive temperature data (if temperature sensors are being used).

In **`setup( )`**

1. Sets up any parameters where the default value is not suitable.
2. Initialises emonLibCM.

and in **`loop( )`**

1. Checks that the library has ‘logged’ the data, then
2. Extracts, prints or forwards the data to wherever it is needed.

Setting parameters and options.
All settings are made with a “setter” function. For example, the default voltage channel is AI 0, and the calibration is set for the emonTx V3.4. To set this for the emonTx V2, using AI 2 and the nominal calibration for that, you must use:

```
EmonLibCM_SetADC_VChannel(2, 227.59);
```

A list of all available setter functions, and a description of what they do, follows in the Application Interface sections.

Extracting the data.
The function

```
EmonLibCM_Ready( );
```

must be called very frequently inside **`loop( ).`** If no new data is ready, this returns immediately with the value **`false.`** If however new data is ready, it returns the value **`true`** and you may then use the “getter” functions to retrieve the data. For example, to retrieve the real power measured on logical channel 0 (normally CT1) and assign the value to the floating point variable **`power1`**, you would use:

```
power1 = EmonLibCM_getRealPower(0);
```

A list of all available getter functions, and a description of what they do, follows in the Application Interface sections.
Example sketches are available: one shows the minimum sketch needed, and a second illustrates the use of every function available (even though in most cases the default value is set again, by way of illustration).

EmonLibCM Application Interface

Power & Energy

void EmonLibCM_SetADC_VChannel(byte ADC_Input, double _amplitudeCal)
void EmonLibCM_SetADC_IChannel(byte ADC_Input, double _amplitudeCal, double _phaseCal)

Sets the physical channels used for the inputs. The first sets the input pin and amplitude calibration constant for the voltage input, the second sets the input pin, amplitude calibration constant and phase error compensation for the current inputs.

Generally, **ALL** the input channels that will be used must be fully defined. Current inputs not in use should not be defined, and omitting unused inputs will improve the sample rate of the remaining channels. Note that if **NO** ADC channels are defined, then one voltage and 4 current input channels will be set, and all inputs will use the default values.

In any set of measurements, the voltage channel is always the first to be read. Thereafter, the current inputs are read in the sequence in which they are defined in the sketch. This sequence becomes the *logical order* used thereafter. For example, if (in the case of the emonTx V3) the input labelled CT3 [= ADC Input 3] on the pcb legend is the first current channel to be defined, it will be accessed as channel 0.

Voltage amplitude calibration constant: This is the unitless ratio of mains voltage to the alternating component of the voltage at the ADC input, and will depend on the voltage divider ratio and the a.c. adapter (transformer) used. Default: 268.97

Current amplitude calibration constant: This is the ratio of mains current to the alternating component of the voltage at the ADC input. Default: 90.9 for all channels except logical channel 3: 16.6 and provided that the default channel sequence is unaltered. The unit is the Siemens (1 / Ω).

Phase error compensation: This is the phase lead of the voltage transformer *minus* the phase lead of the current transformer, in degrees. (It will be negative if the c.t. leads the v.t.). The defaults and the approximate values for the combination of Ideal/TDC UK adapter & YHDC SCT-013-000 on a 240 V 50 Hz supply for the emonTx channels 1-3 is 4.2°, and for channel 4 it is 1.0° at maximum current. (Hint: use steps of 0.2° or larger if setting by trial & error.)

The physical channels must be set before the library is initialised with EmonLibCM_Init( ). To adjust the calibration while the library is running, use the functions below.

There is no return value.

void EmonLibCM_ReCalibrate_VChannel(double _amplitudeCal)
void EmonLibCM_ReCalibrate_IChannel(byte ADC_Input, double _amplitudeCal, double _phaseCal)

The first resets the amplitude calibration constant for the voltage input, the second resets the amplitude calibration constant and phase error compensation for the related current input. These are intended for calibrating the sketch whilst it is running. They must only be called after the library had been initialised with EmonLibCM_Init( ).

The parameters are the same as for EmonLibCM_SetADC_VChannel(…) and EmonLibCM_SetADC_IChannel(…) above, there is no return value.

void EmonLibCM_setPulsePin(int _pin [, int _interrupt])
Sets the active pull-up on the I/O pin on which the interrupt pulse is accepted. The interrupt number associated with that pin is optional (but deprecated) and need only be specified if it is not the default for that pin. (Active pull-up is required to prevent spurious triggering.) The pulse input is active low: if a volt-free contact is used, it must be connected between the pulse input pin and GND. The pulse is recognised on the rising edge (i.e. the opening of mechanical contacts). If a voltage source is used, it must be capable of overriding the input pull-up. Defaults: pin = 3, interrupt = 1.

void EmonLibCM_setPulseMinPeriod(int _period)
Sets the minimum period of time that must elapse between successive pulses for the second pulse to be recognised, in ms. This should be longer than the contact bounce time expected of a mechanical switch contact, but shorter than the time between pulses at the maximum rate expected. For electronic switches that do not exhibit contact bounce, zero may be used. Default: 110.

void EmonLibCM_setPulseEnable(bool _enable)
Enables pulse counting. Pulse counting is initialised when the library is initialised, and thereafter may be turned on or off as required. The change is applied at the next datalogging. The pulse count is frozen whilst pulse counting is disabled. Default: false.

void EmonLibCM_setADC(int _ADCBits, int _ADCDuration)
Sets the ADC resolution (bits) and informs the library of the time taken for the conversion (µs). For an emonTx V3, the values are 10 and 104 respectively. There is no return value. Defaults: ADCBits = 10, ADCDuration = 104

void EmonLibCM_ADCCal(double _Vref)
Sets the ADC reference voltage, nominally 3.3 V for the emonTx and 5 V for an Arduino. If the precise voltage is known, that can be used. There is no return value. Default: 3.3

void EmonLibCM_cycles_per_second(int _cycles_per_second)
Sets the mains frequency, used to calculate the datalogging interval. There is no return value. Default: 50.

void EmonLibCM_min_startup_cycles(int _min_startup_cycles)
Sets the number of complete mains cycles to be ignored before recording starts. There is no return value. Default: 10

void EmonLibCM_datalog_period(float _datalog_period_in_seconds)
Sets the interval (seconds) over which the power, voltage and current values are averaged

and reported. There is no return value. It has not been tested below 0.1 s nor above 5 minutes (300 s). Note that this *may* be called after the library had been initialised with EmonLibCM_Init( ) so that the datalogging period can be changed whilst the sketch is running. Note also that emoncms.org will not accept data faster than once every 10 s. Default: 10

double EmonLibCM_getVrms(void)
Returns the decimal value of the rms average voltage in volts over the reporting period.

int EmonLibCM_getLogicalChannel(byte ADC_Input)

Returns the logical current channel (zero-based) given the physical input as defined by the processor. The return value is meaningless if the voltage input or an unused physical ADC input is given. This can be used to convert the physical input number to the logical current channel required by the following functions.

double EmonLibCM_getIrms(int channel)
Returns the decimal value of the rms average current in amperes over the reporting period for that channel. 'channel' is the logical current channel (zero-based) according to the order in which the current input channels were defined by multiple instances of the statement EmonLibCM_SetADC_IChannel (not the physical channel defined by the pcb legend).

int EmonLibCM_getRealPower(int channel)
Returns the nearest integer value of the average real power in watts over the reporting period for that channel. 'channel' is the logical current channel (zero-based) according to the order in which the current input channels were defined by multiple instances of the statement EmonLibCM_SetADC_IChannel (not the physical channel defined by the pcb legend).

int EmonLibCM_getApparentPower(int channel)
Returns the nearest integer value of the average apparent power in volt-amperes over the reporting period for that channel. 'channel' is the logical current channel (zero-based) according to the order in which the current input channels were defined by multiple instances of the statement EmonLibCM_SetADC_IChannel (not the physical channel defined by the pcb legend).

double EmonLibCM_getPF(int channel)
Returns the decimal value of the average power factor over the reporting period for that channel. 'channel' is the logical current channel (zero-based) according to the order in which the current input channels were defined by multiple instances of the statement EmonLibCM_SetADC_IChannel (not the physical channel defined by the pcb legend).

unsigned long EmonLibCM_getPulseCount( )
Returns the accumulated count of pulses since the library was initialised, provided that pulse counting has been enabled.

long EmonLibCM_getWattHour(int channel)
Returns the integer value of the accumulated energy in watt-hours since the library was initialised, for that channel. 'channel' is the logical current channel (zero-based) according to the order in which the current input channels were defined by multiple instances of the statement EmonLibCM_SetADC_IChannel (not the physical channel defined by the pcb legend).

EmonLibCM_Init( )
Initialise the library. This function must be called once only, and then only after all other set-up functions have been called, typically it will be the last line of setup( ). It assigns all the set-up and calibration constants to the appropriate internal variables and starts the ADC in free-running mode. There is no return value.

bool EmonLibCM_Ready( )
Returns *true* when a new result is available following the end of the reporting period, else returns *false*. Typically, it will be used in loop( ) to control a conditional branch that includes the 'get' functions that extract the required values. It must be called every time that loop( ) executes to ensure correct operation.

bool EmonLibCM_acPresent( )
Returns *true* when the a.c. voltage has been detected (greater than approx. 10% of the nominal input). If the a.c. voltage sensor is not being used, then only current measurements are valid; the real and apparent power, energy and power factor values are meaningless.

Integrity Check
(For debugging or as a performance check only)

int EmonLibCM_minSampleSetsDuringThisMainsCycle( )
Returns the lowest number of sample sets per mains cycle recorded during the last data logging period. The value should be 192 (50 Hz system) or 160 (60 Hz system) divided by 2 for 1 CT in use, 3 for 2 CTs in use, etc; but will depend on the exact mains frequency at the time.
The value 999 is returned if no mains is detected.
To make this function available, you must add the line

#define INTEGRITY

both at the top of your sketch and at the top of the **library** .cpp file.

EmonLibCM Application Interface

Temperature Monitoring

Temperature measurement parameters are set up prior to calling **EmonLibCM_TemperatureEnable**, which enables temperature measurements ('conversion'). The conversion, which can take up to 750 ms depending on the measurement resolution demanded, is triggered so that the measurement result is available to be retrieved the next time that data is logged. The resolution and datalogging periods must be chosen so that there is adequate time for conversion to take place. Depending on the number of sensors, temperature reporting is not reliable with a datalogging period of less than 1 s, and not permitted with a datalogging period of less than 0.2 s. The measurements can be accessed in the form required, either as integers (× 100) or as decimals. The theoretical maximum number of sensors is 127, the practical maximum is much less.
Temperature measurement can be enabled and disabled as necessary, but only the sensor addresses may be changed after the first time that measurements are enabled.

EmonLibCM_setTemperatureDataPin(byte _dataPin)
Sets the pin on which the OneWire temperature data is received. There is no return value.

EmonLibCM_setTemperaturePowerPin(char _powerPin)
Sets the pin that turns on power to the sensors (-1 = no pin is turned on). Setting this to a valid (i.e. non-negative) value turns on power to the powerPin for the minimum time to enable the temperatures to be read and reported, and minimises self-heating. This is available on the emonTx V3.4 only when the sensors are connected via the terminal block. There is no return value.

EmonLibCM_setTemperatureResolution(byte _resolution)
Sets the resolution of the measurement, permissible values are 9, 10, 11 & 12 (bits). The resolution will be set to 9 bits if the *datalog_period_in_seconds* is less than 1 second. There is no return value.

EmonLibCM_setTemperatureMaxCount(int _maxCount)
Sets a limit to the number of sensors that will be initialised. The order in which sensors are discovered is outside the scope of this document and is explained elsewhere. If the number of sensors detected is greater than maxCount, some sensors will not be used, their addresses will not be stored and they should be disconnected. There is no return value.

EmonLibCM_setTemperatureAddresses(DeviceAddress *addressArray [, bool keep])
Sets the array of sensor addresses. The array must be created in the sketch and must be large enough to accept at least maxCount addresses. See EmonLibCM_TemperatureEnable( ) for how to use keep. There is no return value.

EmonLibCM_setTemperatureArray(int *temperatureArray)
Sets the array to save the retrieved temperatures. The array must be created in the sketch

and must be large enough to accept at least maxCount temperatures. There is no return value.

EmonLibCM_TemperatureEnable(bool _enable)
Enables/disables temperature measurements using the DS18B20 sensor array. Temperature measurement will not be enabled unless the array to receive the data has been set using EmonLibCM_setTemperatureArray( ). If necessary, the temperature sensors' addresses are discovered and recorded in the address array. The order in which sensors are discovered is outside the scope of this document and is explained elsewhere. The same measurement resolution is set in each sensor. Sensor addresses can be changed or added to and temperature measurement can be enabled or disabled whilst the library is running (the change taking effect at the next datalogging). No other settings may be changed after the library has been initialised. If _enable is false, any call to EmonLibCM_getTemperature( ) or interrogation of the temperatures array thereafter might return old or invalid data. There is no return value.

> *It is possible to pre-load the sensor addresses, for example from a hard-coded list or from EEPROM. If* setTemperatureAddresses( ) *is used with keep set to true,* EmonLibCM_TemperatureEnable( ) *will not search and discover sensors, but will assume that the array contains a list of valid addresses. MaxCount should be set to a value not greater than the number of addresses in the array. If keep is not present or is set to false, then the sensors are discovered as described above.*
>
> *Hint: The sensor addresses can be found by allowing this function to search for the sensors, and including the* **printTemperatureSensorAddresses( )** *function at the end of* setup( ) *to show the addresses. Those addresses may then be copied manually into the array.*

bool EmonLibCM_getTemperatureEnabled(void)
Returns the state of temperature measurements.

int EmonLibCM_getTemperatureSensorCount(void)
If temperature measurement is enabled, returns the number of active temperature sensors, otherwise zero.

void printTemperatureSensorAddresses(void)
Prints to the serial port a report of the number of temperature sensors detected and their addresses. Assumes the serial port is open. This is intended as a debugging tool and to verify your sensor(s) can be detected, and should not be used in normal operation. Display of your sensor's serial number does not mean temperature reporting is enabled. You must enable temperature reporting with **`EmonLibCM_TemperatureEnable(true);`** There is no return value.

float EmonLibCM_getTemperature(char sensorNumber)
Returns the temperature in degrees Celsius as last recorded by the sensor at the sensorNumber position in the array. SensorNumber is zero-based and must be less that

maxCount. If temperature measurement is disabled, or has only just been re-enabled, the temperature reported might be the last value recorded prior to measurements being disabled.

Error values:
300.00 : Sensor has never been detected since power-up/reset.
302.00 : Sensor returned an out-of-range value.
304.00 : Faulty sensor, sensor broken or disconnected.
85.00 : Although within the valid range, if it is not close to the expected value, this could represent an error, and might indicate that the sensor has been powered but not commanded to measure ('convert') the temperature. It might be a symptom of an intermittent power supply to the sensor.

Getting the temperature as an integer.
To extract the temperature as an integer, e.g. to pack into the payload structure for the RFM radio module, it is possible to access the temperatures array directly, because this is declared in and so is available to the sketch. Viz:

```
emontx.temp1 = myTemperatureArray[0];
```

The value is the temperature × 100 as a signed integer. (e.g. 'Faulty Sensor' would be 30400)

REQUIRED LIBRARIES

These libraries are required to support emonLibCM:

JeeLib	(JeeLabs - https://github.com/jcw/jeelib)
Wire	[Arduino standard library]
OneWire	(Paul Stoffregen)
SPI	[Arduino standard library]
CRC16	[Arduino standard library]
DallasTemperature	(Miles Burton)

INITIAL CONFIGURATION

The following settings should be checked and included in your sketch as necessary.

Set the correct I/O channels according to your hardware, using **EmonLibCM_SetADC_VChannel** and an instance of **EmonLibCM_SetADC_Ichannel** for each current input in use. For best performance, you should not include any current channels that will not be used.

Set the interval at which you wish the values to be reported, using **EmonLibCM_datalog_period**
For emoncms.org, this may not be less than the default value of 10 s. For emoncms running on a private server, any value is permissible, provided that the remainder of the system is compatible. Short periods will give a large amount of data. The library has not been tested with values less than 0.1 s nor greater than 300 s (5 mins). Temperature reporting is not reliable with values less than 1 s.

If you are **not** in the 50 Hz world, set the mains frequency, using **EmonLibCM_cycles_per_second**.

If you are **not** using an emonTx, set the ADC resolution and conversion time, using **EmonLibCM_setADC.**

If you are **not** using an emonTx, set the ADC reference voltage, using **EmonLibCM_ADCCal.**

If you are using temperature sensors:

Set the pins used for the data (and power connection if appropriate), using **EmonLibCM_setTemperatureDataPin** and **EmonLibCM_setTemperaturePowerPin**
Define arrays to receive the sensor addresses and the temperatures.
Set the measurement resolution required using **EmonLibCM_setTemperatureResolution**

If you are using the pulse input:

Set the pin used for the pulse sensor, using **EmonLibCM_setPulsePin**. Set the debounce period using **EmonLibCM_setPulseMinPeriod**, and finally enable pulse counting with **EmonLibCM_setPulseEnable**.

CALIBRATION

Before calibrating a sketch that uses this library, read (but do not do) the calibration instructions in Resources > Building Block Resources. Those instructions contain the general procedure and safety warnings, which you must be familiar with. The detailed instructions that follow apply only to sketches using this library. The default values are given in the description of each function above. Follow *these* instructions for the order in which to make the adjustments and how to apply the values in the sketch, but follow the *general instructions* for how to proceed with the measurements.

Check the ADC reference voltage. The correct nominal value (3.3 or 5.0) should be set with **EmonLibCM_ADCCal(Vref)** so that the actual calibration coefficients will be closer to the calculated values. If desired, set the actual precise measured value. It is not essential to do this, as any discrepancy will be taken up by the individual voltage and current calibration constants.

Set the voltage calibration constant for the voltage input circuit, using **EmonLibCM_SetADC_VChannel(byte ADC_Input, double _amplitudeCal)**

Set the current calibration constant and the phase error compensation for the input circuit of each channel that is in use, using **EmonLibCM_SetADC_IChannel(byte ADC_Input, double _amplitudeCal, double _phaseCal)**

Calibration is neither required nor possible for both the temperature sensors and the pulse input.

EXAMPLE SKETCHES

Two example sketches are provided.

EmonTxV34CM_min.ino

This is the absolute minimum sketch that is needed to use the library. As the comment at the beginning of the sketch states, the sketch assumes that all the default values for the emonTx V3.4 are applicable, that no input calibration is required, the mains frequency is 50 Hz and the data logging period interval is 10 s, pulse counting and temperature monitoring are not required, and that 4 'standard' 100 A CTs and the UK a.c. adapter from the OEM Shop are being used as the input sensors.

This should be your starting point for using the library. If you find that you need to adjust any of the default settings, consult the Application Interface section and then copy the appropriate function either from there or from the full sketch `EmonTxV34CM_max.ino`, changing parameters as necessary.

EmonTxV34CM_max.ino

This provides an example of every Application Interface function. Many will be redundant in normal circumstances as they simply set again the default parameters, many of which are likely to be correct and will not need changing. If you do need to change a value, the Application Interface section above gives full details.

Alphabetical Index of Application Interface Functions

(Note: The function name has been abbreviated for clarity)

acPresent
ADCCal
cycles_per_second
datalog_period
getApparentPower
getIrms
getLogicalChannel
getPF
getPulseCount
getRealPower
getTemperature
getTemperatureEnabled
getTemperatureSensorCount
getVrms
getWattHour
Init
Integrity Check
min_startup_cycles
minSampleSetsDuringThisMainsCycle
printTemperatureSensorAddresses
Ready
ReCalibrate_Vchannel
ReCalibrate_IChannel
SetADC_IChannel
SetADC_VChannel
setADC
setPulseEnable
setPulseMinPeriod
setPulsePin
setTemperatureAddresses
setTemperatureArray
setTemperatureDataPin
setTemperatureMaxCount
setTemperaturePowerPin
setTemperatureResolution
TemperatureEnable