emonTx_3Phase_PLL.ino

This sketch is a development of the 3-phase 'discrete sample' sketch and MartinR's PLL energy diverter. It is intended for use on a 3-phase, 3-wire or 4-wire system. It utilises advanced features of the Atmel328P microprocessor to provide continuous monitoring of voltage of one phase and the currents in all three phases, thereby allowing a good estimate of real power to be calculated. The physical quantities are measured continuously, the average values are calculated and transmitted at user-defined intervals.

Pulse counting and temperature monitoring using a single DS18B20 sensor is supported.

This document has the following main sections:

> Key Properties – important points about this sketch
> Limitations
> Initial configuration
> Calibration
> Installation
> Explanation of operation of the sketch
> List of required supporting libraries

Key Properties

- Continuous monitoring of one voltage channel and up 4 current channels.
- Gives an accurate measure of rapidly varying loads.
- 1800 sample sets per second – using 4 channels of an emonTx V3.4 @ 50 Hz
- Calculates rms voltage, rms current, real & apparent power & power factor.
- Pulse input for supply meter monitoring.
- Integrated temperature measurement (one DS18B20 sensor).
- User-defined reporting interval.
- Suitable for operation on a three-phase, 3-wire or 4-wire supply at 50 or 60 Hz.
- Can be calibrated for any voltage and current (default calibration is for emonTx with 100 A CTs & UK a.c. adapter).

Limitations

Because the voltage of only one phase can be measured, the sketch must assume that the voltages of the other two phases are the same. This will, in most cases, not be true, therefore the powers calculated and recorded will be inaccurate. However, this error should normally be limited to a few percent. Voltage imbalance might be the result of unbalanced loads within your installation, or it might result from the actions of other consumers on the same supply.

If you find that one or other phase voltage is consistently wrong, you might want to consider a small adjustment to the *current* calibration of the affected phase.

Because of the additional power required for continuous operation, a 5 V USB power supply is likely to be required, even if only the on-board RFM69CW radio is used. The 5 V power will always be required when the ESP8266 WiFi module is added.

The sketch is not compatible with the RFM12B radio module, nor the Arduino Due.

The baud rate for serial communications must not be set greater than 9600 – if an ESP8266 WiFi module is used, its baud rate must be changed to suit.

Initial Configuration

The following parameters must be set in the sketch for correct operation. This is done using both *pre-processor directives* and normal variable definitions. There are three ways to change the setting. For example:

1. `#define USEPULSECOUNT`
    As it stands, pulse counting is enabled. To disable it, make this line a comment.
2. `#define PULSEINT 1`
    This defines Interrupt 1 as the interrupt for pulse counting. Change the '1' to '0' to make the sketch suitable for the emonTx V2, which counts pulses using Interrupt 0.
3. `int nodeID = 11;`
    This declares the variable `nodeID` and defines the initial value (11). Change the number as required.

EEPROM Memory

The internal EEPROM memory can be used to retain some of the system settings and calibration constants. When the settings have been saved to EEPROM memory, those values are read at each power-up thereafter and override the values in the sketch. You can however choose to erase the EEPROM settings, which will restore the default values. How to store, change and revert the settings is explained later.

System Settings

`EMONTX_V34`
Defines the emonTx model for the I/O pin mapping. Permissible values are:

`EMONTX_V2`	for the emonTx V2
`EMONTX_V32`	for the emonTx V3.2
`EMONTX_V34`	for the emonTx V3.4
`EMONTX_SHIELD`	for the emonTx Shield

Default: EMONTX_V34

`SERIALPRINT`
Turns on 'human-friendly' print statements for calibration and commissioning - comment this line to exclude. You should probably comment this when the installation is working satisfactorily. This output format is automatically turned off if either of the other serial output formats is selected.

`USEPULSECOUNT`
Turns on the ability to count pulses. Comment this line if pulse counting is not required.

`PULSEINT`
Defines the interrupt number for pulse counting:
    EmonTx V2 = 0,
    EmonTx V3 = 1,
    EmonTx Shield – selectable, see the Wiki.

`PULSEPIN`
Defines the interrupt input pin:
    EmonTx V2 = 2,
    EmonTx V3 = 3,
    EmonTx Shield – selectable, see the Wiki.

PULSEMINPERIOD

Defines the minimum period between pulses, in milliseconds. This should be made smaller than the minimum time between switch operations where a mechanical switch is used (e.g. a magnetic reed switch). Set this to 0 for an electronic sensor with a solid-state output, where there is no risk of contact bounce.

Default value: 110

RFM69CW

Defines the mode of transmitting the output data. Permissible settings are:

      RFM69CW   Hope RFM69CW radio module

      SERIALOUT Wired serial connection via the FTDI port (suitable for direct
                    connection to the emonBase or emonPi
                    (see https://github.com/openenergymonitor/emonTxFirmware/
                     blob/master/emonTxV3/noRF/
                     emonTxV3_DirectSerial/emonTxV3_DirectSerial.ino)

      EMONESP   ESP8266 WiFi module
                    (see https://github.com/openenergymonitor/EmonESP)

If neither radio nor serial connection is required, do not define anything - in which case, the serial output will be 'human readable' for information and debugging only.

  NOTES:

  1. The sketch cannot be used with the Hope RFM12B radio, and will hang if the wrong radio module is specified, or if one is specified and not fitted.

  2. The baud rate for serial communications must not be set to a value greater than 9600. (For the ESP8266 module, change Line 41 or thereabouts in the ESP8266 file "src.ino".)

RF12_433MHZ

Defines the frequency of RFM module. This can be

      RF12_433MHZ
      RF12_868MHZ
      RF12_915MHZ

You should use the one matching the module you have.

  NOTE: This is different from the normal OEM definition.

RFPWR 0x99

Sets the transmitter power.

The control range for transmitter power is 0x80 to 0x9F = -18 dBm to +13 dBm.

  NOTE: Ensure a correctly matched antenna is used when operating at or near maximum power, otherwise the module may be damaged. The RFM12B equivalent power is: 0x99 (+7 dBm).

  Default: 0x99 = +7 dBm

A 5 V d.c. USB supply is likely to be required if the transmitter is operated above minimum power.

nodeID

Sets the node ID for this emonTx. If DIP switch 1 was ON at power-up, then the nodeID is one more than the number specified. This value can be changed at power-up and the new value stored in EEPROM memory.

Default: 11

networkGroup

Sets the wireless network group. This must be same as the emonBase / emonPi and the emonGLCD. The OEM default is 210. This value can be changed at power-up and the new value stored in EEPROM memory.

Default: 210

Sensor Calibration Constants

These values can be changed at run time and the new value stored in EEPROM memory. Thereafter, the stored value is used instead of the values in the sketch.

VCAL
The voltage at the sensor input that gives 1 V rms as measured by the ADC input. It is a function of the a.c. adapter / transformer voltage ratio and the resistor divider network. For the emonTx V3 using the 'standard' UK a.c. adapter, the calculated value is 240:11.6 x 13:1 = 268.97
For the emonTx Shield, the calculated value is 234.2
This value depends on component tolerances and should be adjusted for best accuracy.
If DIP switch 2 is ON at power-up, then the calibration constant for the EU a.c. adapter is used.
Default: 268.97

I1CAL - I4CAL
The current at the sensor input that gives 1 V rms as measured by the ADC input. It is a function of the current transformer ratio and the burden resistor value (or the current transformer's current:voltage ratio for c.t's with an internal burden resistor).
For the emonTx V3 inputs 1-3 and the 'standard' current transformer, the calculated value is 100A:0.05A for current transformer ÷ 22 Ω for resistor = 90.91
For the emonTx Shield, the calculated values are 60.6 & 16.67
This value depends on component tolerances and should be adjusted for best accuracy.
Default: 90.91 for inputs 1 – 3, 16.67 for input 4.

WIRES
Defines the number of wires in the system. Permissible values are:
        3-WIRE
        4-WIRE
Use 3-WIRE where no neutral is available and the a.c. adapter and all loads are connected Line – Line,
use 4-WIRE where there is a neutral and the a.c. adapter is connected Line – Neutral, and loads are connected mainly Line – Neutral, though Line – Line is permissible.
Default: 4-WIRE

CT1Phase
CT2Phase
CT3Phase
CT4Phase
Attaches c.t.1 - 4 to one of the phases to allow the correct voltage reference to be used for real power calculations. Permissible values are:
        PHASE1
        PHASE2
        PHASE3
Phase 1 is defined as being the phase that feeds the a.c. adapter, irrespective of the wire colours or labelling. Phases 2 & 3 follow phase 1 in that order.
C.T's 1 – 3 must be specified, only c.t.4 is optional. If c.t.4 is not needed, comment the appropriate line. This permits a faster sampling rate to be selected – see NUMSAMPLES below.
Defaults:
        c.t.1: PHASE1   c.t.2: PHASE2
        c.t.3: PHASE3   c.t.4: PHASE1

I1LEAD - I4LEAD

The number of degrees by which the a.c. adapter / v.t. phase error leads the c.t. phase error. This can be obtained from the manufacturers' or from test data, or can be established by trial and error during calibration.

NOTE: Input 4 of an emonTx will probably be significantly different, even with the same c.t., due to the different burden resistor.

Default: 2.00 for c.t's 1 – 3, 0.2 for c.t. 4.

PLL & ADC Constants and Settings

LEDISLOCK

The on-board LED lights to indicate that phase lock has been obtained, and flickers very briefly to indicate a data transmission (this might not be readily visible). Comment this line for the 'standard' LED behaviour where a flash indicates data transmission.

SUPPLY_VOLTS

Specifies the reference voltage for the ADC. This will normally be 3.3 for any emonTx but 5.0 for an Arduino (including the emonTx Shield). If the accurate voltage is known, it may be specified. This value affects the calibration of every analogue input.

Default: 3.3

SUPPLY_FREQUENCY

The nominal supply frequency, either 50 or 60.

Default: 50

NUMSAMPLES

The number of times to sample each 50/60Hz cycle. The number chosen must be a multiple of 3 for a 4-wire system, and a multiple of 6 for a 3-wire system. Permissible maximum values are:

	3 c.t's	4 c.t's
50 Hz	45	36
60 Hz	36	33

Lower values are permissible, but in general should not be used without good reason. (E.g. if phase lock is lost.)

Default: 36

ADC_BITS

Defines the resolution of the ADC. 10 for the Atmel 328P (emonTx, many Arduino boards).

Default: 10

ADC_RATE

The time between successive ADC conversions in microseconds. This is calculated from the clock frequency and the divider setting. Do not change this unless either of these is changed.

Default: 64

LOOPTIME

Sets the time between data transmissions in milliseconds.

Default: 5000

PLLTIMERRANGE
Sets the limits for the PLL timer. 100 = ~ +/-0.5Hz. This should not normally be changed.
Default: 100

PLLLOCKRANGE
The allowable ADC range to enter locked state. This should not normally be changed.
Default: 40

PLLUNLOCKRANGE
The allowable ADC range to remain locked. This should not normally be changed.
Default: 80

PLLLOCKCOUNT
The number of cycles to determine if PLL is locked. This should not normally be changed.
Default: 100

Calibration

Calibration might not be necessary provided that the 'standard' components are used.
However, if there is any deviation from this, and especially if the optimum accuracy is
desired, then the unit must be carefully calibrated using known good test instruments.
If test instruments are not available, then by adjusting the various calibration constants by
small amounts, over a period of time, it should be possible to obtain agreement with the
energy supplier's meter.

Before calibrating the sketch, read (but do not do) the calibration instructions in Learn >
Electricity Monitoring > Current & Voltage. Those instructions contain the general
procedure and safety warnings, which you must be familiar with. The detailed instructions
that follow apply only to this sketch. 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.

1. Obtain a resistive load that will draw a current that is close to but less than the
maximum that your multimeter can read and less than the rating of your c.t. Steps 2 – 5
that follow can be done on the test bench.
2. Plug in the a.c. adapter. Set the output to use SERIALPRINT, i.e. if you have enabled
SERIALOUT or EMONESP, temporarily disable those. Using your multimeter (or a reliable
voltage measurement), adjust VCAL so that the voltage measured is the same as your
meter indicates.
3. Set all three main c.t's (inputs 1 – 3) on the cable to your test load, and all facing the
same way. If you are using c.t. 4, put this on the cable too. Label the c.t's, because the
calibration is for the combination of c.t. and input. From now on, each c.t. must stay with
the same input.
Temporarily set the sketch to 4-wire, and set each CT phase to PHASE1.
4. Insert your multimeter in series with the test load, to measure current. The power factor
for every c.t should be nearly +1.0. If it is negative, reverse all the c.t's on the cable, or
reverse your a.c. adapter – if it is of the type that can be reversed. (But if you do reverse
your a.c. adapter, from now on ensure you do not reverse it again.) Adjust the current
calibration factors I1CAL, I2CAL, I3CAL & if appropriate I4CAL to read the same current as
your meter. Ignore the power values and power factor. When done, take the meter out of
circuit. If you are using (say) an inverter's power meter to calibrate, set I1CAL to give the
correct c.t.1 power, but set I2CAL, I3CAL and I4CAL to give the same currents as c.t.1.

5. Adjust I1LEAD, I2LEAD, I3LEAD and if appropriate I4LEAD so that the power factor for each c.t is as close as possible to 1.000.
6. Set the sketch to 3-wire or 4-wire according to your power system, and in the sketch set each CT to the phase it will be used on. Leave the c.t's on your test load.
Check (but do not re-adjust) the power factors:

> For a 4-wire system, the power factor for the c.t's on phase 1 should be nearly +1.0, the c.t's on phase 2 and phase 3 should be close to -0.5
> For a 3-wire system, the combined power for inputs 1 & 2 should be 1.5 times the previous value, and the combined power factor should be close to +0.75, a c.t on phase 1 should be +1.0, and a c.t on phase 2 should be close to +0.5

Transfer the c.t's to their final positions.

No calibration is required for the temperature sensor.

Temperature Sensor 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.

Pulse sensor.
If multiple pulses are counted where only one is expected, and a mechanical reed switch is in use, then it might be necessary to increase the value in `PULSEMINPERIOD`.

Installation

It is possible that there will be exposed live metal parts where you install the current transformers. If you think this might be the case, take the greatest possible care and ensure that you, or nothing that conducts electricity, touches a live part.

At the place where you will install the c.t's, identify the phase sequence and the phase on which the a.c. adapter will be installed. This is most important. If you cannot identify the phase sequence from the wire colours or the labelling, you will need to do this by trial and error:
Plug in the a.c. adapter. This will define phase 1 as far as the sketch is concerned, even though that might not be phase 1 as indicated by the phase colours or labels or terminal numbers on the meter or main circuit breaker.

Use your test load, and check its power rating. Switch off all other loads. Put the c.t. for phase 1 (c.t. input 1) onto one of the main phase conductors. Plug the test load into each phase in turn. If, at some point, the test load reads the correct power, c.t.1 is on the correct phase. If it reads half the correct power, the c.t. is not on the correct phase, so try another.

When you have identified phase 1, repeat with the remaining c.t's and the remaining phase conductors.

Our convention is consumed power is positive. Reverse the c.t. on its cable if the power is negative. If you use an a.c. adapter that is reversible, that may be reversed if all the c.t's indicate the wrong polarity. Take care not to reverse the a.c. adapter subsequently.

Pulse Input
A reed switch or S0 sensor should be wired between the input pin and GND.

On-line calibration & configuration of RF Module

On-line calibration & configuration is not available when the serial output is being used for machine-read data (i.e. when `SERIALOUT` or `EMONESP` has been set).

The user has the opportunity during the Power-On-Self-Test (POST) procedure to change the Network Group and the Node ID, and to adjust the sensor calibration when running.

To enter configuration mode at start-up, using the serial monitor part of the Arduino IDE, enter "+++" followed by the [Enter] key when prompted. You must respond within 10 seconds. If you do nothing, the start-up procedure continues normally after the timeout has expired.

You will then see a short menu:

```
Available commands for RF config during start-up:
<nn>i      - set node ID (standard node ids are 1..30)
             (i - reports the node ID)
<nnn>g     - set network group (1-250, RFM12 only allows 210)
             (g - reports the network group)
r          - restore sketch defaults
s          - save config to EEPROM
v          - Show firmware version
x          - exit and continue

Available commands when running:
k<x> <yy.y> <zz.z>
           - x = a single numeral: 0 = voltage calibration,
                 1 = ct1 calibration, 2 = ct2 calibration, etc
           - yy.y = a floating point number for the voltage/current
                 calibration constant
           - zz.z = a floating point number for the phase calibration
                 for this c.t.
                 (z is not needed, or ignored if supplied, when x = 0)
           e.g. k0 256.8
                k1 90.9 2.00
l          - list the config values
s          - save config to EEPROM
```

For the RF configuration commands changed during start-up, if you change one or more of the settings, the change will take effect when you continue (option 'x').

For the settings you change when running, you will not normally see any response to a 'k' command, but you will see the displayed values change. You will see confirmation when you save the changes. If you change one or more of the settings, the change will take effect immediately.

Option ('s') will save all the changes. If you do not do this, the settings will revert to the previous values at the next restart. After you save ('s') the changes, the new settings will be used forever, or until changed again.

If you restore the sketch default values ('r' during start-up), the EEPROM data is erased and the sketch restarts immediately, using the values set in the sketch. There is then no means of recovering the EEPROM data.

If the sketch starts and all the calibration values are zero, then it is possible that the EEPROM has been used previously and has had zero values written. In that case, restart the sketch, enter configuration mode and choose 'r' - this will erase the EEPROM to the manufacturing default values and the sketch will restart using its own set values.

Explanation of operation of the sketch

General Principle

The sketch measures samples of voltage and current many times in each cycle of mains electricity. To be able to measure real power and energy, which is the quantity that energy suppliers generally charge for, the relationship between voltage and current must be known. The emonTx has only one voltage input, and as this sketch is designed for 3-phase use, it is necessary to use artificial methods to obtain a voltage reference for the other phase(s).

In the case of a 4-wire (3 phases + neutral) supply, there are two other phases. The voltage reference for these is obtained by ensuring that the number of voltage readings in one mains cycle is a multiple of three, and by storing the voltage readings for one third and two thirds of a cycle, so that they can be made to coincide with the second and third phases of the mains supply.
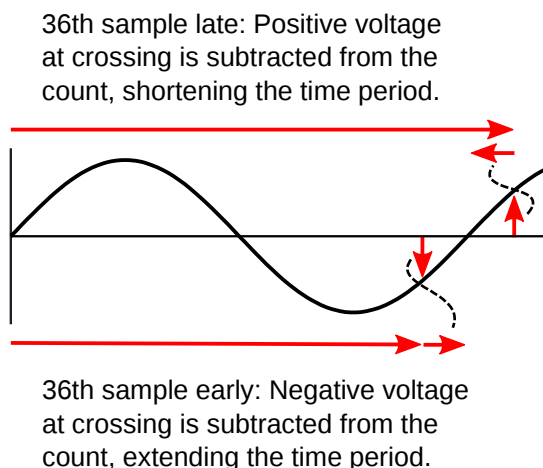
In the case of a 3-wire (no neutral) supply, one phase conductor is treated as the neutral and so there is only one other 'phase' and it is displaced one sixth of a cycle behind the first. In this case, the voltage reference for the second phase is obtained by ensuring that the number of voltage readings in one mains cycle is a multiple of six, and by storing the voltage readings for one sixth of a cycle, they can be made to coincide with the second phase of the mains supply.

The sketch uses a Phase-Locked Loop (PLL) to ensure that the readings always happen at the same place on each phase on each mains cycle, even though the mains frequency can change.

Pairs of readings (voltage and current) are multiplied together and added in an accumulator. At the end of the 'reporting period', the average power is calculated and transmitted. The rms averages for the voltage and current are calculated in a similar way.

What is a PLL – operating principle

A Phase Locked Loop (PLL) is a particular form of a closed loop control system. It is often used, as here, to follow the changes in frequency of some incoming signal. In our case, that is the mains electricity supply.

36th sample late: Positive voltage at crossing is subtracted from the count, shortening the time period.

36th sample early: Negative voltage at crossing is subtracted from the count, extending the time period.

If we set up our emonTx to measure the mains voltage and current 1800 times per second, (that's 36 times per cycle in the UK) then as long as the mains frequency remains at 50 Hz, and as long as the clock in our emonTx is accurate and doesn't drift, then we will indeed take exactly 36 measurements in each cycle, and on successive cycles each measurement will be at exactly the same place on the wave. Clearly, in practice even if we can start out like this, it won't stay like it for long. Both the mains frequency and our clock can drift, and so the measurements will slide

backwards and forwards along the wave as the two frequencies change relative to each other.

To stop this from happening, we must control the frequency at which we make the measurements so that they happen at exactly the same place on the wave each time. If we say we start measuring when the mains voltage crosses zero going positively, then the 37th measurement should again be zero. If our clock is running fast, we will get there early and the voltage will be negative and we need to slow our clock; on the other hand, if we are late the voltage will be positive and we need to speed our clock up. So we are not only locking our clock and the frequency of our measurements to the mains, we are also locking every 36th measurement to the positive-going zero crossing of the voltage. We have a phase locked loop.

The ADC and interrupts

The analogue to digital converter (ADC) in the Atmel processor can run in one of two modes. The data sheet lists features that are key to how we use the ADC:
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete

This sketch uses the "Single Conversion Mode". The first conversion is started by an interrupt from Timer 1, which is the timer that controls the PLL. This "interrupts" the main program, and a special function, the "interrupt handler" or "Interrupt Service Routine" (ISR) then runs. It is from here that the instruction to start the first conversion is sent. When that conversion completes, we tell the ADC which conversion to perform next. While the ADC is converting, the main program is free to carry on with whatever it was doing.

When the ADC finishes the first conversion, it puts the result in a special place. It also generates an "Interrupt on ADC Conversion Complete". This again interrupts the main program, and a second ISR then runs to pick up the result and do something useful with it (and it's this function that tells the ADC which conversion to do next).

This second ISR is at the heart of the sketch. In addition to directing the operation of the input multiplexer and the ADC, it performs the initial calculations on each sample to remove the initial offset, multiplies or squares the samples and accumulates them over the measuring period.

The maths – removing the bias offset

The way that the ADC input works means that we must 'bias' the input signal to mid-way between the power supply rails in order to measure an alternating voltage. But that means we must then remove the effect of that bias in order to read the correct value. That can be done by measuring the average input over a long time (a whole number of mains cycles) and then subtracting that average from the readings. This has the advantage that we do not need to do complicated maths using decimal numbers on each reading – we can simply multiply voltage and current (for the power) or multiply the current or voltage by itself for the rms value – and add the result to an accumulator. This inherently uses integer mathematics, which is a simpler and faster operation than calculating decimal fractions in a software filter.

Purely to reduce the size of the numbers, we actually subtract the nominal offset – 512 for a 10-bit ADC – first, then perform the multiplication and accumulation, finally removing the remaining offset at the end of the measurement period. For the real power, we simply subtract the "offset power", which is the product of the voltage and current offsets. For the

rms current and voltage, we make use of a simplified version of the formula for the rms value of a wave with many components:

$$[rms \ of \ signal + offset] = \sqrt{(signal^2 + offset^2)}.$$

The maths – removing the effect of timing and transformer phase errors

The physical properties of all transformers means that there will be a phase error between input and output. Careful design and choice of materials can minimise this, but there will always be some error. We also introduce a timing error because, having only one ADC, it is necessary to measure voltage and current one after the other. Fortunately, we can use the same software to compensate for these two effects together. The first step is to read the current first, because generally, the current transformer has the smaller error – which is almost always a 'phase lead' – meaning the current appears to have been measured before it actually was. The voltage transformer (a.c. adapter) has a much greater phase lead, so by measuring it later, we have already partly compensated for its phase lead. But we have no control over the time between readings, that is decided by the ADC. However, it is possible to interpolate between voltage samples to generate a new voltage wave that we can move about in time at will.

The formula used by emonLib, which does this calculation for each voltage sample and then multiplies the 'shifted' voltage sample by the current sample to give the instantaneous power, which is then averaged over the sampling period, is

$$average \ power = \frac{k}{n} \sum_0^{n-1} [(lastSampleV + PHASECAL * (sampleV - lastSampleV)) * sampleI]$$

where *k* is a calibration constant depending on the voltage and current calibration constants, the ADC reference voltage and the ADC resolution, and *n* is the sample count.

Because `PHASECAL` is a constant, the equation can be rearranged as

$$average \ power = \frac{k}{n} . (PHASECAL . \sum_0^{n-1} [sampleV \times sampleI] + (1 - PHASECAL) . \sum_0^{n-1} [lastSampleV \times sampleI])$$

This implies that the two multiplications can still be done with integer arithmetic, it follows that the summation is integer arithmetic, and the only calculation that requires decimal precision is multiplication by `PHASECAL` or (`1-PHASECAL`) and the final scaling by *k/n*. And that is done after summation, in the main program where time is not critical.

One final refinement is to adjust the values of `PHASECAL` and (`1-PHASECAL`) to compensate for the amplitude change that interpolation creates. For a pure sine wave, the coefficients are modified thus:

$$average \ power = \frac{k}{n} . (X \times \sum_0^{n-1} [sampleV \times sampleI] + Y \times \sum_0^{n-1} [lastSampleV \times sampleI])$$

where
   $\Delta = sampling \ interval$
   $\Phi = phase \ error \ difference \ between \ two \ transformers$
   $Y = \sin \Phi / \sin \Delta$
   $X = \cos \Phi - Y \times \cos \Delta$

While interpolation still introduces some more distortion to already distorted waves, such as the 'flat-topped' mains wave shape commonly encountered, it is much less than when extrapolation is used.

Detailed description of the sketch

1. emonTx_3Phase_PLL.ino

The first part, lines 1 – 150, contain comments and the normal settings that the user might need to change. There should be no need to change anything below line 150.

Debugging pins – lines 159 – 164
These are pin allocations for attaching an oscilloscope, to monitor timing etc. Due to the limited availability of accessible output pins, it might be necessary to re-use pins already allocated. This will create a conflict, proceed with care.

Line 167 - Arduino pin use.
Definitions of I/O pins for various hardware.

Line 230 - constants calculated at compile time.
Values substituted by the pre-processor, placed here primarily to reduce clutter in the code.

Line 253 - Temperature sensor definitions.

Line 268 - Pulse counting definitions and variables.

Line 288 - Include the required libraries.

Line 293 - Data structure for RFM transmitted data.

Line 299 – Variables for rarely changed intermediate calculations.

Line 305 - Values accumulated over 1 mains cycle. These are shared between the ISR and the main program.

Line 311 - Values accumulated over the reporting period. These are of necessity 64-bit values, and used exclusively in the main program.

Line 317 - Function and variable declarations used in final stages of calculation.

Line 340 - setup( ). Set I/O pins, serial comms, initialise RF module. The first temperature measurement is started, the ADC is set up (but not started) and the phase correction coefficients calculated. Finally, the timer is started.

Line 486 – loop( ). The main loop processes no time-critical operations. As each mains cycle completes, it add the cycle accumulated values to the period accumulators, and at the end of the reporting period - 'nextTransmitTime', it calculates the average values, fetches the temperature reading, and transmits the data. It then starts the next temperature conversion. (Note: temperature readings may therefore be "old" by nearly the full reporting interval. Normally, temperature will change only slowly so this is of little consequence.)

Line 488 – getCalibration( ). If user input is present (via the IDE serial monitor), then this allows sensor calibration to be achieved without reloading the sketch.

Line 525 - Timer 1 interrupt handler. Timer 1 is the main timer for the PLL. It runs at a frequency that is nominally the mains frequency times the number of samples per cycle

(i.e. between 1440 and 2250 per second), and each time starts the ADC with the conversion of the first (c.t.1) current.

Line 538 - ADC interrupt handler. This is the heart of the sketch. It executes when the ADC has completed a conversion. First (lines 549-552) it fetches the result of the conversion just completed and removes the nominal offset of half the ADC range. Then, depending on which conversion has just completed, it starts the next one.
For the current samples, it stores the I value and the I² value in their respective accumulators.
For the voltage sample, it stores the V value and the V² value in their respective accumulators, then it goes on to accumulate the 'partial' power values (which will eventually give the totals for using in the phase & timing correction formula) for each of the three or four c.t's.
Finally, it calls the function that updates and corrects the PLL timing so that synchronism is maintained.

Line 622 - PLL update function. This is called from and is therefore part of the ISR. It looks for the end of a cycle, and checks the voltage that was measured. If it is rising, then the Timer 1 count is adjusted as described earlier. If lock has been lost, then the timer is set so that lock can be regained as quickly as possible. The 'locked' flag is controlled from here.

Line 674 - addsumCycle( ) This function is called by the main loop when a mains cycle ends, and transfers the cycle totals into the 'period' accumulators.

Line 727 - removeRMSOffset( ). Applies the formula to remove the residual offset from the rms value that includes the offset.

Line 733 - removePowerOffset( ). Subtracts the "offset power" from the average power that includes the offset.

Line 738 - Converts degrees to radians for the trigonometric functions.

Line 743 - applyPhaseShift( ). This function calculates the real power from the two 'partial' power values.

Line 752 - Calculate voltage, current, power and frequency.
This is the final calculation to produce the result of the measurement in engineering units. First, the frequency is calculated, then the residual offsets are removed and the true value of real power, apparent power and power factor calculated for each c.t.
The data structure for the RFM module is filled (line 817) and the period averages are zeroed.

Line 864 – calculateConstants( ). Calculate the internal, rarely changed, ratios used in the calculations.

Line 887 – calculateTiming( ). Calculate the coefficients for applying the phase error correction by interpolation.

Line 903 - Send the results. The results are formatted according to the eventual destination (screen display, ESP8266 or emonBase/emonPi) and sent to the serial output.

Line 1018 - isTemperatureSensor( ) Checks for the presence of the temperature sensor.

Line 1023 - convertTemperature( ) Issues the command to the DS18B20 to measure the temperature.

Line 1030 - readTemperature( ) This commands the DS18B20 to send the temperature reading on the OneWire bus, processes and converts the data to a temperature. An error value is returned if a fault is detected.

Line 1064 - onPulse( ). This is the ISR triggered by a pulse from a reed switch, Hall effect or optical detector. Pulses occurring sooner than PulseMinPeriod after the previous one are ignored, this effectively removes contact bounce. PulseMinPeriod may be set to zero if the input device does not exhibit contact bounce.

2. config.ino

This handles the 'on-line' configuration commands.

Line 21 – loads the EEPROM standard library and sets up a structure 'eeprom' to facilitate the interface between the sketch variables and EEPROM memory.

Line 53 – load_config(bool verbose) Reads data from EEPROM and assigns it to the relevant variables. If the first byte of EEPROM is 0xFF (255), it is assumed that the EEPROM memory is either unused or erased, and no action is taken, therefore the values set in the sketch are used. If 'verbose' is true, the values are printed to the serial user interface. The values should not be printed when using SERIALOUT or EMONESP.

Line 89 – list_calibration( ) Prints the calibration constants.

Line 103 – save_config( ) Transfers the configuration variables into EEPROM.

Line 135 – wipe_eeprom( ) Writes the value 0xFF (255) to the area of EEPROM in use. (This is the default value before first use of the EEPROM memory.)

Line 145 – softReset( ) Restarts the processor

Line 150 – readInput( ) This handles user input via the IDE Serial Monitor during the start-up phase. Interpretation of the commands is handled by config( ).

Line 182 – config( ) Here the incoming characters are processed. Numeric input is built into an integer value, and the alpha character that follows determines the action taken. The present RF configuration is printed.

Line 273 – getCalibration( ) This handles user input via the IDE Serial Monitor at run time. If no serial input is available, no further time is wasted and the function returns immediately. If a character is present, then, if it is 'k', the following 2 or 3 values are read and the values are loaded into the appropriate constants, and the necessary follow-up actions (calculation of other variables, including the phase calibration) is done. If the first character was 'l', then the calibration values are listed; if the first character was 's', then the configuration is saved.

Line 370 – showString( ) Prints the help text.

3. rfm.ino

The interface to the Radio Module.

The data format is fully compatible with JeeLib. The function names match those used in JeeLib (from which the major part of the RFM69 code is derived), but the rfm_send( ) function has been modified to accept the node ID and network group, thus allowing these to be changed without requiring rfm_init( ) to be used again. The code is heavily commented and requires no further description.

REQUIRED LIBRARIES

These libraries are required to support the sketch:

Wire	[Arduino standard library]
OneWire	(Paul Stoffregen)
SPI	[Arduino standard library]
CRC16	[Arduino standard library]
EEPROM	[Arduino standard library]

Documentation Version 1.6 for Sketch Version 1.6	No change. 24 February 2019
Documentation Version 1.5 for Sketch Version 1.4	Note added to advise that the serial baud rate in the ESP8266 module must be changed to 9600. 27 January 2019
Documentation Version 1.4 for Sketch Version 1.4	Extended to accommodate 3-wire system. 22 October 2018
Documentation Version 1.3 for Sketch Version 1.3	Declaration of showString( ) added to sketch. 25 September 2018
Documentation Version 1.2 for Sketch Version 1.2	Temperature fault codes changed. 12 March 2018
Documentation Version 1.0 for Sketch Version 1.0	10 January 2018