EmonPiCM

The history of emonLib, and why we want Continuous Monitoring

The original emonTx was designed as a small, battery-powered device for monitoring whole-house power, which transmitted its data to the emonGLCD – a liquid crystal display. As such, it was essential that it operated at minimum power and the “discrete sample” monitoring mode was created (though it was never called that until much later). In this mode, current is sampled for approx. 200 ms, and the data sent by radio to the GLCD. The emonTx then goes to sleep for about 10 s, using very little power, before repeating the measurement sequence.

This works surprisingly well provided the power demand doesn’t fluctuate rapidly. But there’s a problem if a load is switched on and off rapidly, with only a few seconds in either state (e.g. an oven or hob). If the load is switched on while the emonTx is "asleep", the energy consumed during that interval won't be recorded, hence the average power measurement will be incorrect.

Converting the emonPi to continuous monitoring

A few years before emonLibCM became standard for the emonTx, the emonTx V3 had been introduced. V3 uses the RFM69CW radio, which is capable of handling the transmitted data as one complete message. The emonTx 3-phase sketch needs to use this capability because the processor is too busy taking and processing measurements to be able to handle the radio traffic one character at a time. Neither the 3-phase sketch, nor the standard emonTx sketch, now use JeeLib, but the transmitted message is totally compatible with JeeLib – or JeeLib in *compatibility mode,* as the mode for the original message format is now called.

Implementing continuous monitoring on the emonPi is difficult because of the need to handle incoming radio traffic while continuously monitoring two power channels (and optionally, reading as many as 6 temperature sensors.) The RFM69CW employs what is called “packet mode”. The transmitter can accept the entire message into its internal buffer, then transmit it independently of the processor. Meanwhile, the receiver can, independently of its processor, receive, validate and hold the message, ready for processing when the processor can handle it. What JeeLabs could not predict when they created the ‘classic’ JeeLib packet format was that Hope’s packet format for the RFM69CW would be different, making it impossible to use the RFM69CW’s ‘native’ format, yet be compatible with anything using the ‘classic’ JeeLib format.

Therefore, *if you want to use emonPiCM* ***and*** *other sensing nodes sending their data by radio,* there is no choice, the format that OEM uses must change. If you only have an emonPi, you can use CM if you wish, there is nothing else to change. If you have other nodes sending by radio and don’t want or need CM on the emonPi, or you use an emonBase, there’s no need to change.

Options

There appeared to be three main options:

1. Use the LowPowerLabs library. It is well-established, it lives up to its name by using a low data rate of 4799 bits per second, which allows the transmitter power to be turned down. That's eminently suitable for environmental monitoring, but the regulations regarding the use of the radio band mean that the emonTx running the standard CM sketch would be breaking the law if it transmitted more than once every 8.5 s.

2. Write our own software library. As mentioned above, both the emonTx 3-phase sketch and the standard emonTx CM sketch use our own transmitter library, but an equivalent receiver library has not been written.

3. Use the JeeLabs RFM69 driver. This is, compared against JeeLib, a very simple piece of software and is easy to understand. It has a couple of shortcomings: it does not check that the channel is free before transmitting, and there was a bug in the receiver code that resulted in an incorrect RSSI value being returned.

Decision

On the basis of these factors, the decision was made to go with the JeeLabs RFM69 driver, patched to correct the RSSI bug, for the emonPi and emonTH. For the mains-powered emonTx and emonTx Shield, we modified the OEM transmitter software to check that the channel is clear before transmitting.

The emonPiCM front-end sketch

The sketch that runs in the Atmel ATMega 328P primarily provides the energy monitoring function and handles the radio traffic coming in from other sensor nodes.

It uses emonLibCM, set up for two current channels, to obtain and process current and voltage readings and, from external sensors, temperature readings and the pulse input.

It is now possible to fully calibrate the emonPi. Calibration commands come in via the serial data connection from the Raspberry Pi. The interface has been designed so that these commands can be generated manually and routed to the front end via the Raspberry Pi, and so that it might be possible at some future time to set up and calibrate the front end via the emonHub/emonCMS GUI.

emonPiFrontEndCM.zip (55.6 KB)
https://community.openenergymonitor.org/uploads/short-url/ew2L5uXCqJTHs0Yw6IeyoY8PV5A.zip

MD5 Hashes:

emonPiFrontEndCM.zip	9de88226b6376cdaab49e5f97bcfcbab
emonPiFrontEndCM.ino	6be0b5c1c7a4d305b3970ed0d6e0806a
emonPiFrontEndCM_config.ino	7c2e2ba7937f5b7de47d9fecb6a92ff8
emonPiFrontEndCM.pdf	9dbe63138121b5842bb5eb1779a1f6be

Loading the front-end sketch

The file must be compiled with the “Export compiled binary” command and the resulting “.hex” file transferred to the emonPi. The procedure to get from there it into the “emon” front-end processor is given in section 8 of the emonPi Wiki: https://wiki.openenergymonitor.org/index.php/EmonPi#Uploading_Arduino_Firmware

Supporting sketches for monitor nodes

The sketches listed below are available. They send r.f. data in the new ‘RFM69 native’ format to the emonPiCM. The single-phase energy monitoring ones have also been updated to use emonLibCM and include a standardised on-line configuration and calibration section.

emonTx V3.4 single phase: **EmonTxV34CM_rfm69n.ino**
EmonTxV34CM_rfm69n.zip (71.4 KB)
https://community.openenergymonitor.org/uploads/short-url/zkNTP5oBC1yQ1GjoFfHJE868RP7.zip

MD5 Hashes:

EmonTxV34CM_rfm69n.zip	034f48fa8c65e7e3587f3359841539d6
EmonTxV34CM_rfm69n.ino	92f516f0547c5d440ae8a867d17cde8d
EmonTxV34CM_rfm69n_config.ino	c0cb3167db3c7ae161f91e305cd57030
EmonTxV34CM_rfm69n.pdf	4e14bb8b09deac0a173eb669b064f0f7

emonTx Shield single phase: **EmonTxShieldCM_rfm69n.ino**
EmonTxShieldCM_rfm69n.zip (69.0 KB)
https://community.openenergymonitor.org/uploads/short-url/7gyGlH4HatbevgvXtHdNan5hiCE.zip

MD5 Hashes:

EmonTxShieldCM_rfm69n.zip	f3c59f2fec041597c70912a557312acd
EmonShieldCM_rfm69n.ino	96ab9d0359ad61f7bdda81414f521fc1
EmonTxShieldCM_rfm69n_config.ino	d0faf73e18e7a715a2b318423eb97f38
EmonTxShieldCM_rfm69n.pdf	ef9be26c14c035e01e08903e69dbc439

emonTH: **EmonTH_V2_rfm69n.ino**
(note: This sketch offers no protection against r.f. collisions – protection may be enabled but battery life is shortened, possibly to below 75%, depending on r.f. traffic)

EmonTH_V2_rfm69n.zip (54.8 KB)
https://community.openenergymonitor.org/uploads/short-url/8hKrhYZWu5XUSEI33yP2gLm1xvD.zip

MD5 Hashes:

EmonTH_V2_rfm69n.zip	d5bbacbbbeaa93d15daf30d9fb1055e0
EmonTH_V2_rfm69n.ino	e420a993e2ccffcef19dbcd113a9a538
EmonTH_V2_rfm69n_config.ino	a426e90d982a49c30d3140fe86bce2c3
EmonTH_V2_rfm69n.pdf	cb5488cb4ee5811326ac7a3e28161dcd

emonTx V3.2: **EmonTxV32CM_rfm69n.ino**
EmonTxV32CM_rfm69n.zip (45.4 KB)
https://community.openenergymonitor.org/uploads/short-url/lHCxqHOoP6a4yZE3MeWuqa5vjSX.zip

MD5 Hashes:

EmonTxV32CM_rfm69n.zip	9d8091c2219481c99fecbf824c91b0a9
EmonTxV32CM_rfm69n.ino	0266d4f8a5ccbe4f123674fa10a6e11b

MD5 Hashes:

EmonTxV32CM_rfm69n_config.ino	8be37cb658ea94664924212f7ce2c4e0
EmonTxV32CM_rfm69n.pdf	356a6735361c56a814fa22b9f0f8bcb2

emonTx_V2 (RFM12B or RFM69CW): **emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n.ino**
emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n.zip (59.5 KB)

MD5 Hashes:

emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n.zip	39e3e4a5ecdabb2bf54481259941c47e
emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n.ino	df10b3a4e87d1f0f0e1e47501f0cfe50
emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n_config.ino	8bdaf7b9998734cbd6b7f2b9dce7511e
emonTx_V2_CT123_Voltage_Temp_Pulse_rfm69n.pdf	fe88dcd340dfcbbc07cce4b2db1dd02f

emonTx 3-phase **emonTx_3Phase_PLL_rfm69n**
(all hardware variants, both Classic and RFM69 Native formats)
emonTx_3Phase_PLL_rfm69n.zip (137.2 KB)
https://community.openenergymonitor.org/uploads/short-url/zngBXa75W5OiTSRJ1dVBWbzAG6T.zip

MD5 Hashes:

emonTx_3Phase_PLL_rfm69n.zip	a68c0657806781d59ca344beae22d09d
emonTx_3Phase_PLL_rfm69n.ino	0a8b68a69fa10b7faac55dbb86535d0f
emonTx_3Phase_PLL_rfm69n_config.ino	e90bcffbaaa8f87296f69f8491f2e2da
emonTx 3-phase PLL User Doc.pdf	b5356c2b7ab96a4855a3c5b2ac56f8a3

GLCD

Existing sketches for the GLCD can be recompiled using a patched version of JeeLib.

EMONBASE

There should be no need for an emonBase to use the new ‘RFM69 native’ format – it will only be required if and when the RFM69 native’ format becomes the standard. Until then, the only use case will be if it replaces an emonPiCM, thereby removing the need to convert the monitor nodes back to the ‘RFM69 classic’ format.

emonBase_rfm69n.zip (42.2 KB) https://community.openenergymonitor.org/uploads/short-url/zK9lk6Edl5OIvFMlja9FVYhDUm4.zip

MD5 Hashes:

emonBase_rfm69n.zip	36c77071de009140440e51245dde66f1
emonBase_rfm69n.ino	1ee9c827c85e76762f37ff80067dfee7
emonBase_rfm69n_config.ino	b01d0e4d902aa29f74a621c05d8dd562
emonBase_rfm69n.pdf	860269ae8f68f506303a85c13005b49a

Required Libraries

emonPi front end:
- emonLibCM
- emonEProm
- rf69

emonTx V3.4 (single phase)
emonTx Shield (single phase)
emonTx V3.2 (single phase)
emonTx_V2: (single phase):
- emonLibCM
- emonEProm
- rfm69nTxLib

EmonTH V2:
- *power
- *sleep
- rfm69nTxLib
- emonEProm
- JeeLib (JeeLabs)
- OneWire (Paul Stoffregen)
- DallasTemperature (Miles Burton)

emonLibCM requires:
- *Arduino
- *Wprogram
- *SPI
- *crc16
- OneWire (Paul Stoffregen)

emonEProm requires:
- *EEPROM

* This is a standard Arduino library.

Sketches for emonGLCD

It is possible to convert JeeLib to operate in RFM69 Native mode, however great care must be taken as it means making a change to one of the JeeLib files itself. When this is done, most of the emonGLCD sketches should work with sensor nodes and emonPi that are using the RFM69 ‘native’ message format.

In the file RF12.h, near the top, you will find these lines:

```
/// RFM12B driver definitions

// Modify the RF12 driver in such a way that it can inter-operate with RFM69
// modules running in "native" mode. This affects packet layout and some more.
#define RF12_COMPAT 0
```

Change the last line to read

```
#define RF12_COMPAT 1
```

Now, for the IDE to recognise the change, you must completely exit the Arduino IDE, restart it and compile the sketch as normal.

If for any reason you need to revert to the JeeLib ‘Classic’ mode, you must restore the line to its original state and restart the IDE. Due to the way the Arduino IDE works, it is NOT possible to have two versions of the library - even in different directories.

Reference Documents

JeeLabs publications:

JeeLabs published 5 separate documents describing the RMF69 Native format. Those and a chart showing the various formats have been consolidated for convenience into a single PDF document:

JeeLib Radios RFM69 Native.pdf
https://community.openenergymonitor.org/uploads/short-url/i2rXeCszsm3OE12ZSAxqoisu2rk.pdf

The original JeeLabs documents are:
RFM69 on ATmega » JeeLabs:
https://jeelabs.org/2015/05/27/rfm69-on-atmega/index.html

Classic vs native packets » JeeLabs:
https://jeelabs.org/book/1522a/index.html

RF compatibility options » JeeLabs:
https://jeelabs.org/book/1522b/index.html

RF69 native on ATmega's » JeeLabs:
https://jeelabs.org/book/1522c/index.html

Using RFM12's with RFM69 native » JeeLabs:
https://jeelabs.org/book/1522d/index.html

Design of the "old" RF12 Driver:

Inside the RF12 driver » JeeLabs:
https://jeelabs.org/2011/12/10/inside-the-rf12-driver/

Inside the RF12 driver – part 2 » JeeLabs:
https://jeelabs.org/2011/12/11/inside-the-rf12-driver-part-2/index.html

Inside the RF12 driver – part 3 » JeeLabs:
https://jeelabs.org/2011/12/12/inside-the-rf12-driver-part-3/index.html

RF12 packet format and design » JeeLabs:
https://jeelabs.org/2011/06/09/rf12-packet-format-and-design/index.html

RF12 broadcasts and ACKs » JeeLabs:
https://jeelabs.org/2011/06/10/rf12-broadcasts-and-acks/index.html

Data Sheets:

RFM12B Radio Module:
https://www.hoperf.com/data/upload/portal/20190306/RFM12B%20Datasheet.pdf

RFM69CW Radio Module:
https://www.hoperf.com/data/upload/portal/20190307/RFM69CW-V1.1.pdf