OEM EPROM library

Rationale:
To preclude the possibility of conflict, when a sketch loaded into a previously used processor attempts to use inappropriate values left behind in the EEPROM.
To ease EEPROM management.
To provide a mechanism for saving the ‘most recent’ energy-related values.

Contents

1. Overview:

The library provides a method of managing the EEPROM. When data is saved to an unused EEPROM, or one that appears to have incompatible data, the EEPROM is erased and an “OEM-specific” signature that also identifies the sketch is written at the start of the data block.
The available memory (1024 bytes) is split into two areas, the lower ¼ (approx) is reserved for the configuration and calibration parameters, the upper ¾ is used to store the energy values.

2. Use – configuration and calibration section

Each sketch must be identified by an unique signature. The sketch may read only data that is identified by a signature that it recognises. If the data in EEPROM does not carry the correct signature, it may be erased but it must not be used. A sketch may save its configuration and calibration data to EEPROM. If there is no signature in EEPROM, or one it does not recognise, the EEPROM is erased, formatted and the data is written.
A register of signatures and the corresponding sketches will be maintained centrally. Registered signatures are guaranteed to represent compatible (but not necessarily correct) data. A sub-range has been allocated for personal and experimental use, signatures in this range are not guaranteed, and might contain invalid or inapplicable settings.
The signature is 2 bytes. The values 0x0000, 0xFFFE and 0xFFFF are reserved. 0xFFFE shall indicate a 'worn' area that must not be used (this is not yet, and might not be, implemented). 0xFFFF is the natural 'erased' value, 0x00 might have been written by published sketches to 'zero' EEPROM. The range 0xFF00 - 0xFFFD is reserved for personal and 'experimental' use.

3. Use – wear-levelled energy values section

The actual length of this section is determined by the size of the data structure containing the variables to be saved, it will usually be a little less than 768 bytes.
Each time data is ‘reported’ (transmitted via radio or serial), the present energy values – assumed to include the pulse count, are sent to EEPROM. In order to prevent excessive wear, the set of values is saved to EEPROM only when there is a change in one or more values of 200 or more units set by the value of `WHTHRESHOLD`. A unit will normally be 1 Wh.
At startup, the sketch will retrieve the last saved values so that it may resume accumulating the energy and pulse totals from where it left off.

4. Installation.

Copy the directory “emonEProm” and its contents into the usual Arduino library directory.
Add `#include <emonEProm.h>` to the sketch source file as normal, or use the Arduino IDE menu: Sketch > Import Library

5. API

This section lists all the functions that are included in the library.

unsigned int eepromSize(void)
Returns the total size of the EEPROM, i.e. duplicates EEPROM.length( )
(Note: Not the size available to either section.)

void eepromFormat(void)
Unconditionally erases all EEPROM contents and writes the OEM signature.

bool eepromValidate(void)
Returns true if the "OEM" signature is present, else false.

unsigned int eepromFindSignature(uint16_t signature)
Returns the offset from the beginning of EEPROM of the data block identified by signature, else zero.

unsigned int eepromRead(uint16_t signature, byte *dest)
Copies the EEPROM data of the data block identified by signature to the struct "dest", which MUST be big enough. Returns the number of bytes transferred, or zero if the signature is not found or the data is marked as deleted.

unsigned int eepromWrite(uint16_t signature, byte *src, int length)
Formats and writes unconditionally if the signature is not found or if the EEPROM is not recognised; or if the EEPROM is formatted, finds the block identified by signature and updates it, whether or not that block is marked as hidden. The source data is in the struct “src”, of length “length” bytes. Returns the number of bytes written (=length + block header), or zero if unable to write.

unsigned int eepromHide(uint16_t signature)
Finds and marks the block identified by signature as hidden. (This is equivalent to erasing the data.) Returns the number of data bytes in the block (=length), or zero if unable to find the signature.

void eepromPrint(void)
Print to Serial the entire used contents of EEPROM. (Assumes Serial printing is available via the host sketch.) If the format is unrecognised, the complete content of EEPROM is printed, else each block is printed individually along with its signature.

void storeEValues(long E1, long E2, long E3, long E4, unsigned long pulses1)
(for the emonTx)
void storeEValues(long E1, long E2, unsigned long pulses1, unsigned long pulses2)
(for the emonPi)
The energy variables to be periodically saved to EEPROM. A save is performed when any one of them changes by 200 units or more from the last saved value.

bool recoverEValues(long *E1, long *E2, long *E3, long *E4, unsigned long *pulses1)
(for the emonTx)
bool recoverEValues(long *E1, long *E2, unsigned long *pulses1, unsigned long *pulses2)
(for the emonPi)

Recovers the last saved values from EEPROM and assigns the values to the named variables. Use NULL if that variable is not used. Returns true if valid data is found, else false.

void zeroEValues(void)
Sets all energy values and pulse counts to zero.

Debugging.

Three functions are available to aid debugging the wear-levelling mechanism. All assume the Serial port is available.
`void dump_control(void)` Prints the block size & number of blocks, the address & marker for the active block, the start & end addresses of the memory allocated to the wear-levelling section.
`void dump_buffer(void)` Prints the contents of EEPROM allocated to the wear-levelling section.
`void dump_stats(void)` Prints the number of times all of the allocated EEPROM has been written and the number of blocks saved during the current wear-levelling write-cycle.

Note:
The frequency that the variables are saved at was chosen to minimise the number of writes to EEPROM whilst also minimising the amount of energy that could be 'lost' in the event of a supply failure. The value 200 Wh was chosen because the monetary value is small – about £0.03 (GBP) / $0.03 (USD) yet offers a realistic EEPROM life.
The EEPROM life is claimed to be approximately 100,000 writes. The largest consumer of electricity is the USA, with an average domestic annual consumption of 11 MWh.
One data block represents a minimum of 200 Wh. No distinction is made between energy consumed or generated. It comprises 21 bytes and thus 36 blocks can be stored in the 768 bytes allocated.
If the entire consumption is on one leg of the supply, there will be 55,000 blocks written per year.
The entire allocated EEPROM range will be written a little over 1500 times.
That yields a projected life in excess of 65 years.

Acknowledgement

The wear levelling mechanism is taken from EEWL EEPROM wear level library and is entirely the work of Fabrizio Pollastri, Torino, Italy<mxgbot@gmail.com> and released under the GNU General Public License, (c) 2017 Fabrizio Pollastri