NVSRAM Library

This library enables you to read and write into Microchip NVSRAM 23LCV512 (64K x 8 bit) and 23LCV1024 (128K x 8 bit), battery-backed SPI RAM.

The NVSRAM is great for moving data in and out for applications, without worrying about read-write cycles, wear and power consumption. By simply adding a battery, the NVSRAM can retain data even after power loss and data retention is limited only by the battery capacity.

Microchip's battery-backed SRAM devices 23LCVxxxx have true unlimited read and write cycles, lower standby current (about 4µA), a wider voltage range and can be accessed via an SPI (or SDI) serial bus for fast data transfers.

Library usage and initialization

Memory selection

Firts of all, you have to choose if the library should compile for 23LCV512 (64K x 8 bit - *default*) or for 23LCV1024 (128K x 8 bit). To do this, it is necessary to remove / insert a comment in front of the two-line that you can found at the beginning of the NVSRAM.h file of the library.

By default you have:

```
#define NVSRAM_512
// #define NVSRAM_1024
```

which is the configuration for the **23LCV512** (64K x 8 bit). If you want to compile for the 23LCV1024 (128K x 8 bit) you have to move the comment from the second line to the first line to have:

```
// #define NVSRAM_512
#define NVSRAM_1024
```

To use this library you have to add, at the beginning of your program:

```
#include <NVSRAM.h>
```

To **instantiate** the classe you have to use the NVSRAM constructor that takes two parameters: the first one is the **CS** pin number that you have chosen and is mandatory, the second one is a boolean to indicate if you want to initialize the SPI bus (*and configure the chip*) or not, it is optional and its default value is **true**.

Example:

```
NVSRAM myNVSRAM( 10 );
```

... or, if you **don't want** to initialize the SPI bus (*since you do this manually in your program*) :

```
NVSRAM myNVSRAM( 10, false );
```

... but, in this case (*and only in this case*), **before** using the NVSRAM, **you MUST call** the begin() method to configure the chip:

```
myNVSRAM.begin();
```

Library methods

erase( )

Write 0x00 into all cells of the NVSRAM erasing the memory.

Example:

```
myNVSRAM.erase( );
```

length( )

Return the address of the last memory byte or the size of the memory.

e.g. myLong = myNVSRAM.length( );

read( )

Read a byte from NVSRAM. Requires the address of byte to read.

Example:

```
myByte = myNVSRAM.read( myAddress );
```

write( )

Write a byte into NVSRAM. Requires the address and the value to write.

Example:

```
myNVSRAM.write( myAddress, myByte );
```

update( )

Only for compatibility with the methods of the EEPROM library; does the same thing as write() and have the same syntax.

Example:

```
myNVSRAM.update( myAddress, myByte );
```

get( )

Read any data type or object from the NVSRAM. Requires the address and return the read data or object.

Example:

```
myData = myNVSRAM.get( myAddress );
```

put( )

Write any data type or object to the NVSRAM. Requires the address and the data to write.

Example:

```
myNVSRAM.put( myAddress, myData );
```

crc( )

Compute and return the crc16 (uint16_t) of a block of bytes in NVSRAM. Requires the starting address of the block and the number of bytes to include in the computation.

Example:

```
myCRC16 = myNVSRAM.crc( myAddress, myLength );
```

NVSRAM [ ]

This operator allows you using your **instance**, to the class NVSRAM, like an array to **read** a byte.

Example:

```
myByte = myNVSRAM[ myAddress ];
```