Software style guide for avrtoolbox

This document has two main parts, a C programming style guide, and documentation style guide.

C Programming Style

The C programming guide is modified from the avrcore library suggested guide by Ruddick Lawernce: http://www.nongnu.org/avr-libc/avr-corelib/style_guidelines.html
We will try to use this guide as much as possible since we would like to have our software usable with avr-corelib project, however several good (or at least adamant) suggestions from AVRFreaks led to several differences. [Link: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=101132&highlight=]

Naming Conventions

- **lower_case**: Has all lower case letters, with words separated by underscores.
- **UPPER_CASE**: Has all upper case letters, with words separated by underscores.

Public versus private
We will divide our code into public and private modules, functions, and variables. The public materials are meant to be used in .a type libraries and are documented in the .h file supplied with the library. The private materials are those used to build the libraries and are not exposed for use other than for creating libraries.

Modules
Each module will have a short descriptive name such as spi or uart. Short module names is a subjective requirement with the example of spi_init() versus serial_peripheral_intrface_init()

Files
Public files will use **lower_case** and will have the module name as a prefix. For example: spi.c and spi.h. Private files will have the first letter of the module in **UPPER_CASE.**

Functions
Public function names and arguments will all use **lower_case** and have the module name as a prefix: module_init(). Private functions will have the first letter of the module in **UPPER_CASE**: Module_init()**.**

Module initialization functions will all have _init as a suffix. For example: uart_init(uint32_t baud_rate).

Variables
Variable names will all use **lower_case.**
Keep global variables to a minimum and make sure they are justified and well documented.

Data Structures

Data typedefs use **lower_case** and end in _t (ie, long_timer_t). Any member variables follow the guidelines for variables specified above.

Formatting

As a general guideline, code will use the BSD Allman formatting.

Tab characters are allowed, and the suggested setting is 4 spaces for consistent appearance especially in the documentation.

Use braces for all statements that could have them, even for enclosing a single line.

Example from Wikipedia:

```
while (x == y)
{
    something();
    somethingelse();
}
finalthing();
```

Constant Values

Public constants (in header files) use #define for single values because it will not reserve memory space. Private constants use static const values to limit the scope to the module.

Constants will be **UPPER_CASE** and have the module name as a prefix. For example: #define UART_BAUD 19200.

Registers

Whenever possible, use the read/modify/write paradigm to change registers in order to avoid overwriting other parts of the register. This is best done by using standard bitwise operator techniques.

Scope

Although C has no "private" definition, any functions and otherwise global variables not meant to be used by client code should be declared static.

API

Each module's API is defined in a single header file, named after the module (ie, uart.h).

Documentation Style Guide

Documentation requirements vary with the type of user. The developer will be intimately familiar with the code and not need much other than the code itself. A maintainer should have the same level of programming skill as the developer, but will need information about the software that may not be obvious from reading the raw code, so he will want in-line comments sufficient to help him quickly understand what the code is doing. A user may just want to use the functions and not care how they were generated, so all he'll need is an overview of the module and the specifics of how to use each function – what goes in and what comes out – but generally he doesn't care what happens in between. This leads to two conceptually separable types of documentation: inline comments in the C source for the maintainer, and a separate API document for the user. The developer should just use common sense and let the obvious things be self-documenting while adding comments on things that might not be immediately obvious. We will use Doxygen comments in the header file to generate the API document and will try to anticipate the real needs of an average user.

C file documentation.

Each C module will begin with a title block containing the file name, the author, the date, and the version on one line with the avrtoolbox directory location on the next line as follows:

```
/*****************************************************
      spi.c Joe Pardue September 18, 2010 rev 001
      Location: avrtoolbox\Libraries\Peripheral\spi
*****************************************************/
```

The author is encouraged to add any additional comments he thinks might be helpful to another developer.

All dates will be written with the month name, the date, a comma, and the year, for example: December 11, 2010. This will prevent misunderstandings about numeric date styles between various regions.

Each title block will be followed by this license block:

```
/*
 *  BSD License
 *  -----------
 *
 *  Copyright (c) YEAR, AUTHOR NAME, All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are met:
 *
 *  - Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 *  - Neither the name of the AUTHOR NAME nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *  ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 *  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 *  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 *  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 *  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 *  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 *  POSSIBILITY OF SUCH DAMAGE.
 */
```

The following function is provided as an example of reasonable in-function comments:

```
void spi_init_master(void)
{
    // Set pullups with output high
    PORTB |= (1<< MISO_HARDWARE_PIN) \
        | (1<< MOSI_HARDWARE_PIN) \
        | (1<< SCLK_HARDWARE_PIN) \
        | (1<< SS_HARDWARE_PIN);

    //Set MOSI, SCK AND SS to outputs
    DDRB |= (1<< MOSI_HARDWARE_DDR) \
        | (1<< SCLK_HARDWARE_DDR) \
        | (1<< SS_HARDWARE_DDR);

    // Set Miso to input
    DDRB &= ~(1<< MISO_HARDWARE_DDR);

    // Enable SPI, Set as master, set clock to fosc/16
    SPCR = ( 1 << SPE ) | ( 1 << MSTR ) | ( 1 << SPR0 );
}
```

Using Doxygen to generate the API document.

Doxygen comments will only be used in the header file.

Module title block:
This block contains a general description of the module along with whatever API style information seems appropriate for easing use.

This block will always include:

- \mainpage with module name
- \brief - that 'brief'ly describes what the module does.
- \author – tells who wrote this function.
- \license New BSD
- \date – when it was first released.
- \version – function specific to be incremented each time the function is changed along with the date and a note explaining the change.
- todo – items that remain to be done.
- The /* ***…*** */ dividers to make the file more readable.

The following example block is from SPI.h:

```
/* ********************************************************* */
/*!
      \mainpage SPI (Serial Peripheral Interface) Functions Library

      \brief This is for single master only hardware SPI for either
      8 or 16-bit read-write.

      There are three SPI functions that you will normally use:\n
      void spi_init_master(void);\n
      uint8_t spi_master_rw8(uint8_t to_slave);\n
      uint16_t spi_master_rw16(uint16_t to_slave);\n

      The user is reminded that SPI reads and writes in the same
      operation, for example 8-bits are clocked out to the slave while
      8-bits are clocked in from the slave. For the spi_master_rw8
      function an 8-bit byte is taken as a parameter to send to the
      slave and a byte is returned from the slave. The _rw16 function
      sends and receives 16-bits.

      This code was tested on the ATmega169 (AVR Butterfly, ATmega328
      (Arduino), and ATmega644 (BeAVR)(TODO)

      Location: avrtoolbox\Libraries\Peripheral\spi

      \todo test it for the ATmega644

      \author Joe Pardue
      \license New BSD
      \date December 1, 2010
      \version 1.01 December 11, 2020 corrected initialization.
*/
```

Documenting each function:

Each function declaration in the *.h file will have a preceding block of doxygen comments.

This block will always include:

- \brief - that 'brief'ly describes what the function does. The final line will contain the avrtoolbox directory location for the module.
- \param – name data type and use for each parameter.
- \return – name data type and use for return value.
- The /* ***…*** */ dividers to make the file more readable.

This block may include:

- \note – provides extra information that might prove useful.
- \todo – items that remain to be done.

The following example block is from SPI.h:

```
/* ********************************************************** */
/*!
      \brief Writes and reads an 8-bit byte via SPI.

      \param to_slave 8-bit byte to send to slave

      \return 8-bit byte from slave
*/
/* ********************************************************** */
uint8_t spi_master_rw8(uint8_t to_slave);
```

In the above case, stating the obvious that the \param and \return are 16-bit may be overkill, but is used to differentiate between this function and the nearly identical 8-bit version of the function.

These comments will be used by Doxygen to generate the avrtoolbox user manual. That process is explained elsewhere.