EEPROM - eeprom_update, eeprom_write

Example Code

```
/* EEPROM Update method
 *
 * Stores values read from analog input 0 into the EEPROM.
 * These values will stay in the EEPROM when the board is
 * turned off and may be retrieved later by another sketch.
 *
 * If a value has not changed in the EEPROM, it is not overwritten
 * which would reduce the life span of the EEPROM unnecessarily.
 *
 * Released using MIT licence.
 */

#include <EEPROM.h>

/* the current address in the EEPROM (i.e. which byte we're going to write to next) */
int address = 0;

void setup() {
  /* Empty setup */
}

void loop() {
  /*
   * need to divide by 4 because analog inputs range from
   * 0 to 1023 and each byte of the EEPROM can only hold a
   * value from 0 to 255.
   */
  int val = analogRead(A7) / 4; // Use A7 for example because all supported parts have it: tinyAVR, Dx, and Ex.

  /*
   * Update the particular EEPROM cell.
   * these values will remain there when the board is
   * turned off.
   */
  EEPROM.update(address, val);

  /*
   * The function EEPROM.update(address, val) is equivalent to the following:
   *
   * if( EEPROM.read(address) != val ){
   *   EEPROM.write(address, val);
   * }
   */
  /*
   * Iterate through each byte of the EEPROM storage.
   *
   * Larger AVR processors have larger EEPROM sizes, E.g:
   * tinyAVR 0/1/2-series 2k flash:   *   64b
   * tinyAVR 0/1/2-series 4-8k flash:   * 128b
   * tinyAVR 0/1/2-series 16-32k flash:  256b
   * megaAVR 0-series:   *    *    *    *    256b (all flash sizes)
   * DA, DB, EA-series:   *    *    *    *   512b (all flash sizes)
   * DD-series:   *    *    *    *    *    *   256b (all flash sizes)

   * Rather than hard-coding the length, you should use the pre-provided length
function.
   * This will make your code portable to all AVR processors.
  */

  address = address + 1;
  if (address == EEPROM.length()) {
    address = 0;
  }

  /*
   * As the EEPROM sizes are powers of two, wrapping (preventing overflow) of an
   * EEPROM address is also doable by a bitwise and of the length - 1.
   *
   * ++address &= EEPROM.length() - 1;
  */

  delay(100);
}
```

```
/* EEPROM Write
 *
 * Stores values read from analog input 0 into the EEPROM.
 * These values will stay in the EEPROM when the board is
 * turned off and may be retrieved later by another sketch.
 */

#include <EEPROM.h>

/* the current address in the EEPROM (i.e. which byte we're going to write to
next) */
int addr = 0;

void setup() {
  /* Empty setup. */
}

void loop() {
  /* Need to divide by 4 because analog inputs range from
```

```
   * 0 to 1023 and each byte of the EEPROM can only hold a
   * value from 0 to 255.
   */

  int val = analogRead(A7) / 4; // Use A7 for example because all supported parts have it: tinyAVR, Dx, and Ex.

  /* Write the value to the appropriate byte of the EEPROM.
    these values will remain there when the board is
    turned off.
   */

  EEPROM.write(addr, val);

  /* Iterate through each byte of the EEPROM storage.
   *
   * Larger AVR processors have larger EEPROM sizes, E.g:
   * tinyAVR 0/1/2-series 2k flash:      64b
   * tinyAVR 0/1/2-series 4-8k flash:    128b
   * tinyAVR 0/1/2-series 16-32k flash:  256b
   * megaAVR 0-series:                   256b (all flash sizes)
   * DA, DB, EA-series:                  512b (all flash sizes)
   * DD-series:                          256b (all flash sizes)
   *
   * Rather than hard-coding the length, you should use the pre-provided length function.
   * This will make your code portable to all AVR processors.
   */

  addr = addr + 1;
  if (addr == EEPROM.length()) { // Okay, we've written gibberish over the entire EEPROM
    while (1); // Wait forever - no need to sit there wasting rewrite longevity.
  }

  /* As the EEPROM sizes are powers of two, wrapping (preventing overflow) of an
   * EEPROM address is also doable by a bitwise and of the length - 1.
   *
   * ++addr &= EEPROM.length() - 1;
   */

  digitalWrite(LED_BUILTIN, HIGH); //briefly flash LED as activity indication.
  delay(2000);
}
```

```
Sketch uses 1082 bytes (0%) of program storage space. Maximum is 131072 bytes.
Global variables use 6 bytes (0%) of dynamic memory, leaving 16378 bytes for local
```

```
variables. Maximum is 16384 bytes.

avrdude: Version 6.3-20201216
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is
"C:\Users\ivanFernandez\AppData\Local\Arduino15\packages\Microchip\hardware\megaav
r\1.0.0/avrdude.conf"

         Using Port                    : usb
         Using Programmer              : curiosity_updi
avrdude: Found CMSIS-DAP compliant device, using EDBG protocol
         AVR Part                      : AVR128DA48
         Chip Erase delay              : 0 us
         PAGEL                         : P00
         BS2                           : P00
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 0
         StabDelay                     : 0
         CmdexeDelay                   : 0
         SyncLoops                     : 0
         ByteDelay                     : 0
         PollIndex                     : 0
         PollValue                     : 0x00
         Memory Detail                 :

                                  Block Poll               Page
Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW
ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- -----
---------
           signature      0     0     0    0 no          3    0      0     0     0
0x00 0x00
           prodsig        0     0     0    0 no        125  125      0     0     0
0x00 0x00
           fuses          0     0     0    0 no          9   16      0     0     0
0x00 0x00
           fuse0          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse1          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse2          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse4          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse5          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse6          0     0     0    0 no          1    0      0     0     0
0x00 0x00
```

```
           fuse7          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           fuse8          0     0     0    0 no          1    0      0     0     0
0x00 0x00
           lock           0     0     0    0 no          4    1      0     0     0
0x00 0x00
           data           0     0     0    0 no          0    0      0     0     0
0x00 0x00
           flash          0     0     0    0 no     131072  512      0     0     0
0x00 0x00
           eeprom         0     0     0    0 no        512   32      0     0     0
0x00 0x00

         Programmer Type : JTAGICE3_UPDI
         Description     : Microchip Curiosity in UPDI mode
         ICE hardware version: 0
         ICE firmware version: 1.17 (rel. 514)
         Serial number   : MCHP3280031800001901
         Vtarget         : 3.31 V
         JTAG clock megaAVR/program: 0 kHz
         JTAG clock megaAVR/debug:   0 kHz
         JTAG clock Xmega: 0 kHz
         PDI clock Xmega : 100 kHz

avrdude: Partial Family_ID returned: "    "
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9708 (probably avr128da48)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "0b11001001"
avrdude: writing fuse5 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse5 written
avrdude: verifying fuse5 memory against 0b11001001:
avrdude: load data fuse5 data from input file 0b11001001:
avrdude: input file 0b11001001 contains 1 bytes
avrdude: reading on-chip fuse5 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse5 verified
avrdude: reading input file "0x00"
avrdude: writing fuse7 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse7 written
```

```
avrdude: verifying fuse7 memory against 0x00:
avrdude: load data fuse7 data from input file 0x00:
avrdude: input file 0x00 contains 1 bytes
avrdude: reading on-chip fuse7 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse7 verified
avrdude: reading input file "0x00"
avrdude: writing fuse8 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse8 written
avrdude: verifying fuse8 memory against 0x00:
avrdude: load data fuse8 data from input file 0x00:
avrdude: input file 0x00 contains 1 bytes
avrdude: reading on-chip fuse8 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse8 verified
avrdude: reading input file
"C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_177937/eeprom_write.ino.hex"
avrdude: writing flash (1082 bytes):

Writing | ################################################## | 100% 0.47s

avrdude: 1082 bytes of flash written
avrdude: verifying flash memory against
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_177937/eeprom_write.ino.hex:
avrdude: load data flash data from input file
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_177937/eeprom_write.ino.hex:
avrdude: input file
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_177937/eeprom_write.ino.hex
contains 1082 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.26s

avrdude: verifying ...
avrdude: 1082 bytes of flash verified

avrdude done.  Thank you.
```

Notes

1. Each of these sketches compile and upload scuccessfully to the AVR128DA48 board. Testing complete.