Overcoming erase/write-endurance limitations in EEPROMs

Peter Buitenkant - September 28, 2000

EEPROMs are useful, nonvolatile devices that store data. However, they suffer from a serious deficiency that is absent in other nonvolatile storage devices. A magnetic disk, for example, has no limit on the number of erase/write cycles for each location, but the finite number of erase/write cycles for any byte location in EEPROM often limits the disk's performance and utility. Today's technology limits the most commonly used EEPROMs to 100,000 to 1 million erase/write cycles because the erase function degrades the oxide barrier on the silicon and eventually leads to failure.

You can put the problem into perspective by considering what happens when you use 1 byte of EEPROM as a scratchpad RAM in a CPU-based system. If you erase and write to this byte once per second, the location exceeds its endurance rating in 100,000 to 1 million sec, or approximately 27.7 to 277 hours. Thus, the useful life of the equipment containing this device is to three and one half to 35 days at 8 hours per day of usage. In contrast, general-purpose RAM does not limit the number of write cycles to any location.

Fortunately, most applications use EEPROM not as a scratchpad but rather as a device on which to record data or save settings to survive a powering off of the system. You can use EEPROM to log data, which entails writing data to consecutively addressed bytes rather than constantly erasing and writing to 1 byte. However, a problem arises in tracking the last write address when the unit powers off. Functionally, you need 1 byte of nonvolatile scratchpad RAM to maintain a last write address.

Exceeding the state of the art

With a little careful planning, you can achieve data-storage capability and performance that seems more advanced than that of available state-of-the-art EEPROMs. For example, a recent design of an EEPROM-based data logger for an RF personal-safety monitor achieves three major design goals. It logs data into the EEPROM at 1 data byte/sec with a full date-and-time stamp to the nearest second from the real-time clock. (The logger must store 32,000 data bytes and overwrite the old data with the new data.) It provides a minimum five-year equipment life, assuming 8 hours per day operation. And it minimizes power drain, real estate, software complexity, and cost using only one EEPROM IC (a serial I/O device in an surface-mount-technology package).

The design includes one important power-cycling constraint: The equipment operates from a battery and is subject to random on/off cycling a few times per day. The real-time clock has separate, continuous battery power. Despite the power-cycling constraint, the design meets the above goals and in some cases exceeds them by a large margin.
Using a single EEPROM location as a pointer to track where you write data results in an almost immediate endurance failure in the EEPROM. Typically, a data logger writes data to sequential addresses. After writing to the highest address, the logger wraps around to the lowest address and continues writing, and the data-storage endurance of the entire device increases by the number of bytes that the EEPROM contains. A typical EEPROM that includes 32 kbytes of consecutively written bytes begins to suffer endurance problems only after 32 kbytes×(27.7 to 277 hours), which equals 907,646 to 9,076,459 hours or 453.8 to 4538 years at a usage of 2000 hours per year.

More than 400 years of useful life seems decent, but two problems remain. First, the software needs a pointer to determine where to write the current data. This same pointer serves as a delimiter to indicate where to stop reading when retrieving the data. The software should update this pointer for every logged data byte, but this amount of updating exceeds the endurance limit. RAM is an excellent place to locate this pointer, except for a second problem: Data-logging equipment is often battery-powered, and consequently, it is subject to turn-off at unpredictable times. Thus, you can lose a RAM-based pointer when the power turns off. On the next power-on cycle, the software cannot determine where to begin storing data or where to start and stop reading logged data. Hence, the data pointer must be in EEPROM and is therefore subject to endurance limitations.

An additional problem involves date-and-time stamping. The device must record the exact date and time that every data byte is written into the EEPROM, to the nearest second. A real-time clock in the system provides this information, and an independent five-year lithium battery maintains the clock.

The goal of the design is to answer the following questions: Considering the limited erase/write-cycle endurance of EEPROMs, how can you design battery-powered equipment and maintain the required pointer for writing new data and retrieving stored data? How can you date-and-time stamp, to the nearest second, every data byte that the EEPROM stores without having an excess of overhead bytes to record the date and time?

The following design offers a solution. As a benefit, the design requires no additional hardware and uses a small amount of software and only 17 bytes of the EEPROM for housekeeping. The technique is adaptable to EEPROMs of different types, sizes, and endurance ratings.

For the first step of the design, you can group a reasonable number of bytes into so-called "pages." EEPROM manufacturers use this technique to simplify EEPROM design. You can use a 32-kbyte EEPROM and divide it into 128 pages each containing 256 bytes (256×128=32,768). You might notice that the byte address ranges from 0x0000 to 0x7FFF or 0 to 32,767 decimal. Because each page contains 256 bytes, the byte address of each page ranges from 0x--00 to 0x--FF (0xFF=255 decimal). Also, the address of each page ranges from 0x00-- to 0x7F-- (0x7F=127 decimal).

Grouping bytes into pages

Figure 1 is a graphical depiction of the design's data structure. Note that the complete address of any byte is the concatenation of the 7 bits contained in the EEPROM pointer to form the high address, with the lower 8 bits held in RAM to form the individual byte address within the page. The loss of the RAM's contents on power-off is insignificant because a new page always starts upon power-up and, by design, the RAM's content initializes to the first data-byte location on the new page.

You can reserve 1 byte of EEPROM as a pointer to indicate only the current written page. Because this page pointer requires updating only every 256 data-byte writes, the effective endurance of the EEPROM increases from approximately 27.7 to 277 hours to 7091 to 70,912 hours. This increase is an improvement, but only high-endurance parts can meet the design goal of five years of operation.
or 10,000 hours at 2000 hours per year usage.

Now you can get really sneaky. Instead of only 1 byte, you can reserve 16 bytes of EEPROM as page pointers numbered 0 to 15 (Figure 1b). The first page pointer, 0, tracks the written page as the EEPROM fills the first time. As each page fills up and a new page starts, the value in page-pointer 0 increments starting at 000 and eventually reaches 0x7E as the entire EEPROM fills. Note that the design now reserves page 0x7F for the page pointers. Thus, you lose the use of 256 bytes out of 32,768, which is an insignificant 0.78% loss. After the EEPROM fills the first time, rather than continue using pointer 0, the software begins using pointer 1. When the memory fills again, the software switches to pointer 2, and so on. After the memory has filled up 16 times, the software again returns to using pointer 0. The design implements this technique using a 17th byte of EEPROM. The lower nibble of this byte points to the one active pointer among the 16 pointer bytes.

This technique extends the endurance of the EEPROM another 16 times from 7091 to 70,912 hours to 113,456 to 1,134,592 hours. This increase translates to an endurance limit of 56.7 to 567 years at 2000 hours per year.

If your design meets these requirements, operation from this point is as follows: At power-on of the unit, the software refers to the lower nibble of the 17th pointer byte to determine the active pointer among the 16 possibilities; the software reads the value of the active pointer to determine the page address of the last data write before power-off; and the software increments the pointer and begins to write to the indicated page.

**Tackling another problem**

Upon read-back, an additional problem can arise. The software recognizes the page location of the last data write before power-off, but it cannot determine which bytes on the page are valid because of the absence of a byte pointer.

The solution is simple: When each new page starts, the technique "erases" the bytes on the page to 0xFF, which is an illegal data value (legal data ranges from 0x00 to 0xFA or 0 to 250 decimal). You can operate many EEPROMs in page mode, which allows a page to fill quickly. Thus, minimal software is necessary to perform this erase operation. By scanning the last written page, the software identifies the last byte written into the EEPROM.

At power-on, the software starts a new page and does not attempt to continue writing to the last active page after power-off. This feature allows for date-and-time stamping of each byte to the nearest second.

At the start of each new page, the software first erases 256 bytes to 0xFF. Next, the software writes a header consisting of a few bytes at the top of the page. The header bytes can contain information pertinent to your design. Date-and-time stamping the data requires, as a minimum, the header structure (Figure 2).

You can easily store and retrieve the date of each data byte. For the approximately 252 data bytes on the page (depending on the size of your header), you can use 4 bits to indicate the last two digits of the current year, 4 bits to indicate the current month, 1 bit to indicate a.m. or p.m. (A/P), and 5 bits to indicate the date. SECONDS_HIGH (1 byte) together with SECONDS_LOW (1 byte) indicate the number of seconds (0 to 43,199) past 12 o’clock (a.m. or p.m.) that correspond to the creation of the page header.

You can create a time stamp that is accurate to the nearest second because a data write
automatically occurs every second. To retrieve the time stamp of a data byte, the software reads the number of seconds past 12 o'clock that corresponds to the time the software created the header and adds to that number the positional location (offset by the header bytes) of the data byte in the page. (Remember that each data byte records once per second.) For example, the first data byte's time stamp equals SECONDS_HIGH/SECONDS_LOW+0. The second data byte's time stamp equals SECONDS_HIGH/SECONDS_LOW11, and the pattern continues with subsequent data bytes.

**Helpful hints**

A few tips might ease your efforts to overcome EEPROM limitations. First, none of the 17 pointer bytes require initialization. The first time you use an EEPROM, a random but legal 4-bit address appears in the lower nibble of the 17th pointer. This nibble points to one of the 16 page pointers, which also has a random starting page address in its lower 7 bits. All values except 0x7F are legal. If the software detects an 0x7F, it automatically increments this value to 0x00. This location is where the software writes the first page. After random start-up, the pointers behave predictably.

Second, serial-access EEPROMs are common. To speed operation with these EEPROMs, you can use 2 or 3 bytes of RAM to keep mirror copies of the page-pointer contents and the address (one of 16) of the active page pointer. You can store updates of these variables in the EEPROM, which stores copies only in RAM.

**Author info**

Peter Buitenbant works as a consultant in analog/digital/microcontroller circuit design as well as C and assembly-language programming. Most recently, he has developed a battery-powered personal RF monitor, a colorimeter, a ratio turbidimeter, and the analog and digital circuitry for an explosive gas detector. He has a BA in physics from Queens College (New York), a BSEE from Columbia University (New York), and an MSEE from Brooklyn Polytechnical Institute (New York). You can reach him at peterbui@compuserve.com.

**REFERENCE**


2. Microchip Corporation data sheet, 32K 5.0V SPI Bus Serial EEPROM, 25C320, document DS21159B.