

A Single Memory for Code & Data

Ruminations on design & applications



Overview

Applications for memory technologies generally are divided between executable code and data tasks. ROM-based technologies are nonvolatile and are oriented toward code applications. These include mask-ROM, OTP-EPROM, and Flash. EEPROM, another derivative of ROM technology, serves primarily as a nonvolatile data memory. EEPROM is clearly a compromise since it performs both code and data storage with low-performance relative to the alternatives. Flash can be used for certain data storage applications. In these applications, it retains its superiority as a code memory and provides limited, low performance data storage. Flash's main virtue in such applications is low cost rather than ease of use or performance for data storage.

RAM-based technologies serve as working-space for code execution and as data memory. Mainstream alternatives include DRAM and SRAM. These technologies provide an excellent blend of code and data functionality. Unfortunately, the traditional choices provide only temporary storage. Unless battery backed, they must be combined with some form of nonvolatile memory for permanent storage.

Space-constrained applications need maximum functionality in a single device. Ideally, combined code and data storage would reside within a single memory device. Since it must be nonvolatile, it eliminates ordinary RAM from consideration. This leaves ROM technologies, which have poor performance as data storage mediums, battery-backed RAM, and Ferroelectric RAM (FRAM).

Few nonvolatile memory technologies lend themselves to a combination of executable code and data better than FRAM. A single FRAM device fulfills the requirements of both applications. The practical limit of this approach is the available memory density. This issue is beginning to fade as higher density FRAM devices become available.

The emergence of high density FRAM provides an ideal single-chip code and data memory solution. A few design considerations should be reviewed in implementing a single-chip memory environment. This application note reviews the memory requirements for both code and data, and then the design considerations for combining them in a single FRAM device.

Code Storage

Prior to reviewing the suitability of a device for a single chip code & data environment, it is helpful to review the ideal characteristics for each separately. In many systems, code storage is ROM or OTP-EPROM and requires little thought. The memory is nonvolatile and cannot be altered in-system. Therefore, the basic parameters of memory size, access time, and operating voltage are the design issues. Above all, the memory must be nonvolatile and should retain its state under all conditions. Code storage is a read-oriented environment. Working parameters and temporary variables that must be altered are stored in RAM.

Introduction of field programmable storage raises issues about the methods and circumstances of writing the code memory. Flash memory is widely viewed as the choice for field programmable memory. The earliest flash memory devices did not truly provide in-system programming since they needed elevated external voltages and required complete erasure prior to programming. In a single chip environment, the program that controls the upgrade process may be required to reside-in and be executed-from the storage device that is being programmed. This makes bulk erasure impractical.

Later flash devices could be programmed at system VCC, and were sectored so that only a portion of the memory could be erased prior to programming. This improved their field upgrade capability. Today's flash memory devices are still difficult to upgrade in a single-chip memory environment, but overall they are considered relatively ideal for code storage. Flash has substantial shortcomings as a data memory, discussed in the next section. A summary of the attributes needed and desired for code storage is listed below. The later two points are particularly desirable for systems with just one or a few memory devices.

Code Memory Considerations

- ✓ *Nonvolatile*
- ✓ *Appropriate density*
- ✓ *Read access time*
- ✓ *Ability to prevent inadvertent writes*
- ✓ *Field programmable in parts or sections*
- ✓ *Programmable w/concurrent read access*

Data Storage

In many respects, the requirements for data are the opposite for code. Data storage is a far more diverse task, requiring flexibility and easy write-access. It may be either nonvolatile or volatile depending on the type of data. For many applications, the write functions of a data memory are at least as important as the read functions. An extreme example, the flight data recorder (black box), may never be read (we hope). In most cases, a write operation should not require complex protocols, and definitely should not take extended time. In addition, a high number of writes should be accommodated for virtually unlimited data collection.

In strictly volatile applications, RAM technologies are the typical data storage selection. SRAM is used for relatively small storage, fast data access, and low power; while DRAM is used for relatively large data buffers. They perform these functions well, as long as volatile storage is sufficient. Batteries can be added to make the RAM contents nonvolatile, but this creates more problems.

Nonvolatile data storage is a challenge for the mainstream memory technologies. ROM-based technologies like Flash make poor data memories for all but the most static configuration settings. The very term 'programming' implies that adding new content to Flash is significantly more difficult than adding it to RAM-based memories. Complex algorithms for writing, delays for erasing and relatively long write cycles are normal for Flash. Sectors program at comparatively slow speed, but must first be erased which is extremely slow. EEPROM is a traditional choice for nonvolatile data memory since it is simpler to use. However, it offers very slow write times and both technologies permit few write-cycles before wearing out.

FRAM is the first memory optimized for nonvolatile data storage. FRAM writes occur at the same speed as the reads. It is nonvolatile, and offers a very high (>1E12) number of access cycles. For 3V FRAM products, the number of endurance cycles is 1E16. No algorithms or protocols are needed to write the memory, and it is byte addressable. FRAM provides the most flexible solution with the fewest drawbacks for data storage.

A general summary of data storage attributes is more difficult than code storage. The applications are far from homogenous. In addition, the lack of solutions in earlier generations has restricted the amount and type of data that can be gathered in many systems. A list of potential data-memory attributes to be considered follows.

Data Memory Considerations

- ✓ *Fast write access*
- ✓ *Many writes allowed*
- ✓ *Simple write protocol (none is best)*
- ✓ *Byte addressable writes*
- ✓ *Nonvolatile (some applications)*
- ✓ *Ability to serve volatile & nonvolatile needs*

Single Chip Code & Data Requirements

Combining code and data applications in a single device can mean getting one memory to provide almost mutually exclusive services. Code requires nonvolatile storage and the need for occasional writes. Upgrading code will never call for a large number of write operations (cycles). The write time usually is not important. In some respects, the more difficult it is to write a code memory the better since an accidental write could be catastrophic. Data memory often needs a mix of volatile and nonvolatile storage, or at least a nonvolatile storage that can be written without restriction. It requires relatively large numbers of writes, and write speed can be an important factor.

As mentioned above, FRAM technology currently is optimized for data memory applications. Its superior write-performance makes it preferable to Flash or EEPROM, and its nonvolatile aspect makes it preferable to SRAM. Since it is nonvolatile, it can serve as code memory as well. The main limitation of using FRAM in a single-chip memory application is available density. Today the FM18L08 32Kx8 is the highest density FRAM device sold by Ramtron. If the density is sufficient, an important system design issue is to prevent inadvertent writes to code storage areas while still permitting them in an upgrade situation and of course to the data storage area.

Writing FRAM is very similar to writing an SRAM. When serving as code storage, it is critical to make sure that the system does not accidentally write to an area of memory being used for executable code.

FRAM is easily adapted for code storage by creating a simple write-protection circuit. This logic can be used to prevent inadvertent writes to code areas (or even data areas). An example circuit on the next page provides programmable block write protection for FRAM or any other RAM solution. Hardwiring the Sector Write Access inputs creates a fixed block write protection scheme, while connecting them to logic or a microcontroller provides a method for dynamically changing the write protection. The circuit is described below.

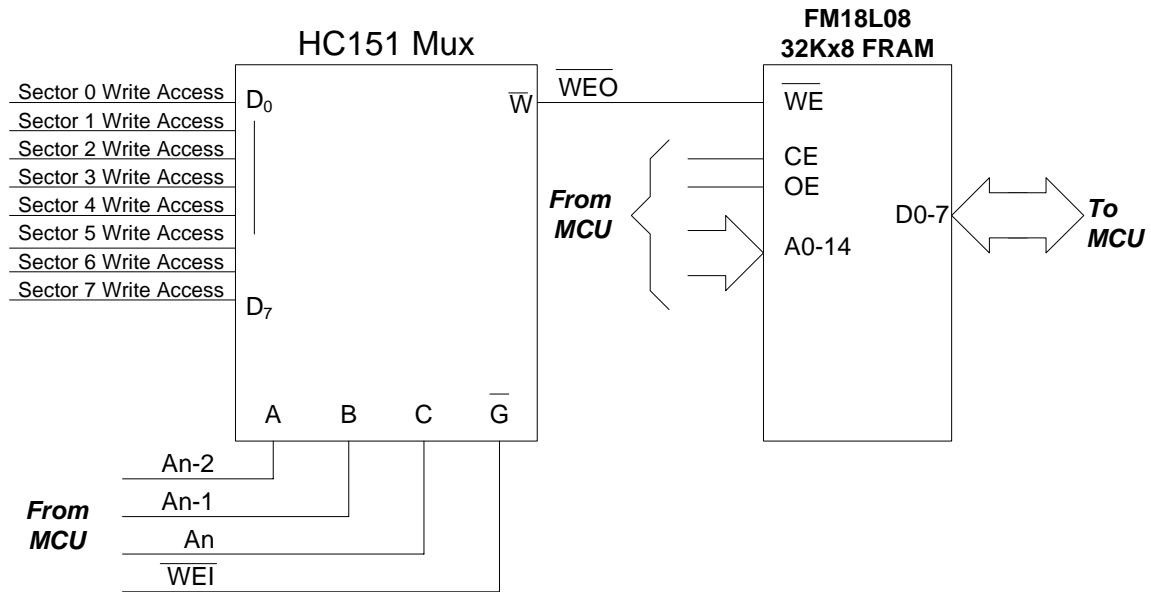


Figure 1. Sector Write Protection Circuit

Operating the Write Protection Scheme

This circuit uses an ordinary CMOS multiplexer such as the HC151 to create an address dependent write-enable that can be connected directly to the FRAM device. For reference, the truth table of the HC151 is provided.

In this circuit, the Write Enable (/WE) signal input is active low as the FRAM would require. The /WE input signal from the microcontroller or decoder is labeled /WEI. The /WE output from the circuit to the FRAM is labeled /WEO. Any time that the /WEI signal is high, the /WEO output of the HC151 remains high. When /WEI goes low, the current address and corresponding write-access input determine if the /WEO to the FRAM is permitted to go low.

Assuming the three, most-significant memory address lines are connected to the HC151; the memory space is divided into 8 equal sectors. For each sector, the D_n input of the HC151 determines if the sector is write-protected. The incoming address will select one of eight sectors. If the corresponding D_n input is high, then the HC151 /WEO will go low when /WEI is low. Thus, these sectors can be written. If the selected D_n is low, then the /WEO output will remain high for all addresses in that sector, regardless of the state of the /WEI input.

The simple circuit above allows a system to add programmable block write protection to any RAM memory including FRAM. One option is simply to hard-wire the settings for write-access to fixed values based on the locations of code and data. This

Table 1. HC151 Truth Table

Select Inputs			Strobe: G	W
C	B	A	/WEI	/WEO
X	X	X	H	H
L	L	L	L	/D0
L	L	H	L	/D1
L	H	L	L	/D2
L	H	H	L	/D3
H	L	L	L	/D4
H	L	H	L	/D5
H	H	L	L	/D6
H	H	H	L	/D7

Sector write-access values set to 1 will allow writes to that sector. Sector write-access values set to 0 will block writes to that sector.

prevents inadvertent writes but eliminates the possibility of making a field upgrade of the code without making a circuit board change. A minor variation of hard wiring is to use jumpers. This would allow for changes but still require human intervention.

A more flexible alternative involves connecting the write access settings to a microcontroller (possibly via other logic). If the power-on-reset state of these inputs is set to low, then the default condition is to write protect the entire device. Under software control, the system can then alter these settings to either provide limited full-time write access to certain sectors or to dynamically open and close access to all sectors. This scheme allows write protection of critical data areas as well as code.