TECHNICAL NOTE

TN216

Is your Application Ready for Large Sector Flash?

And Other Caveats for Writing to Flash at Run-Time

The rapidly changing market for flash devices may make it necessary for Rabbit Semiconductor to install "large sector" devices on future products instead of the "small sector" devices currently used. At the time of this writing, there are no plans to do so, but market conditions could change that. It is wise to consider the possibility now if your application writes to flash at run-time. Switching to large sector flash devices will significantly affect such applications. For the purposes of this document, flash chips with 4K sector size are considered *small* sector flash chips.

Changes to Dynamic C have already been made. Customers designing their own Rabbit-based system who wish to use large sector flash may do so starting with Dynamic C 7.20.

Flash Device Types

Flash devices come in a variety of sizes and formats which make them more or less useful for a particular application. Two basic differences in flash device types are physical sector size and whether the device is byte- or sector-writable.

Small Sector vs. Large Sector

The terms "small sector" and "large sector" as used in industry are somewhat arbitrary. This document uses the following definitions:

A *small sector flash* must contain equally-sized sectors of 128, 256, 512, 1024, or 4096 bytes. It can be either sector-writable or byte-writable (see below).

A *large sector flash* is either a byte-writable device with at least two sectors of different sizes, or a device with sectors of uniformed size larger than 4K. Typical large sector flash devices have a variety of sector sizes from 4K bytes up to 128K bytes on a single device, while some others contain equally sized sectors of 16K bytes up to 64K bytes.

Sector-Writable vs. Byte-Writable

Nearly all flash devices are either sector-writable or byte-writable. To change a byte of a sector-writable device, the entire sector containing that data must be erased and rewritten. This is usually done within the device itself: the new sector's worth of data is transferred to the device, and it proceeds to erase that sector and rewrite it. For this reason, sector-writable devices typically have small sectors of 512 bytes or less.

Any byte of a byte-writable device can be rewritten at any time, but the device can only erase entire sectors at a time. In addition, the physical design of the flash device only allows bits to change from 1 to 0 unless an entire sector erase occurs, which resets all the bytes in the sector to 0xFF (all bits are 1). For example, you can change a single byte from 0x05 (00000101) to 0x04 (00000100), but not to 0x07 (00000111) since you would be changing the a bit from 0 to 1. To accomplish this change, the entire sector would need to be erased to 0xFF (11111111) and rewritten.

Rabbit uses and supports both byte- and sector-writable devices. At the time of this writing Rabbit ships products containing the SST ST29EE010 (sector-write), Atmel AT29C020 (sector-write), and Mosel Vitelic 29C51002 (byte-write). Many other small sector devices are supported by the flash driver as well. *The Rabbit 2000 Designer's Handbook* has a list of supported flash types.

Primary Flash vs. Secondary Flash

All current Rabbit products have either one or two flash devices on them. The primary design for a two-flash controller is for the first (primary) flash to store the program and the secondary flash to serve as a "flash file system."

As of Dynamic C 7.05, support exists for compiling code to both flash devices (typically producing 512K bytes of code space total) as well as for placing a small flash file system with the code in a single flash. keep in mind that it is still desirable to put the flash file system on a second device if possible since each write to the flash device containing the currently running code requires blocking all interrupts during the write cycle, which adversely affects real time performance. Putting both program code and a file system in the secondary flash is *not* supported.

All current Rabbit products that contain two flash devices have identical devices for both the primary and secondary flash. This may not be true in the future since large sector flash is inherently unsuitable for use in a flash file system. It is entirely possible that Rabbit may release a product in the future with a large sector primary flash and small sector secondary flash.

Note on 4K "Small" Sector Flash

Support for this type of flash existed in Dynamic C starting with version 6.5x, however download, debug, run-time writing performance is slow. Starting with Dynamic C 7.20, this performance has been optimized. See Technical Note 217 for compatibility issues.

Large Sector Flash Support

Typically, large sector flash are slightly faster at writing data when compared to small sector flash. However, the overhead involved in changing a byte if the sector needs to be erased first is formidable. Large sector devices such as the Atmel AT49F002 contain sectors up to 128KB in size which takes on the order of a second or two to erase, and up to 10 seconds to erase the entire chip.

Versions of Dynamic C prior to 7.20 rely on the ability to rewrite a sector in order to download a program, debug, and write data to the flash. It should also be obvious that saving the contents of a 128KB sector to RAM before an erase is not feasible on a board with only 128KB of RAM installed, and difficult even with more RAM available.

To solve this problem, Dynamic C 7.20 and later will compile the program differently if it detects a large-sector flash on board. First of all, the BIOS will be compiled to a bin file but not downloaded until the program is also compiled to a bin file. A loader containing flash drivers will be downloaded to RAM and will erase the flash and load the BIOS and user program to flash.

If flash is going to be written at run-time, only top boot block type parts should be used, but boot block locking should not be turned on.

Flash File System

The flash file system is significantly more difficult to implement if small sectors are not available. For this reason, Rabbit plans to keep a small sector flash installed as the second flash on boards that use a second flash.

The flash file system will work on a board that contains a large sector flash as long as the file system is contained in the small sector flash (the default setup). The flash file system will not work on a large sector flash, so if you have a single-flash device with a large sector flash installed, the flash file system will not be usable. Consider using battery-backed RAM for data storage.

Support currently exists for using the flash file system on the primary flash, but this support will not be maintained for large sector flash in the event they are used.

Ways of Writing Flash

Various methods of writing to the target flash exist, and any changes to these methods will be examined here:

Initial Programming

There are currently three ways to initially write to the flash before a user application has a chance to. After one of these has completed, there are a small number of API functions that can write to a flash device at run time.

1. **Dynamic C**

   As mentioned above, Dynamic C will now download a program differently when it detects a large sector flash on the target: the BIOS and program will be downloaded together, and the BIOS (while running in RAM) will erase all flash sectors that need to be updated.

2. **Cloning**

   Cloning will behave similarly to the BIOS; the loader first transferred to the clone will erase the necessary flash sectors before the BIOS and program are transferred. For more information about cloning, please see Technical Note 207, *Rabbit 2000 Cloning Board*.

3. **Rabbit Field Utility**

   The RFU, a utility that downloads bin files, follows the same procedure that Dynamic C does.

Writing at RunTime

The following functions write to a flash device at runtime.

WriteFlash

The WriteFlash function is used to write an arbitrary amount of data directly to an absolute physical address in flash. This relies on the ability to store a sector in RAM and rewrite the contents, which is not feasible for large sector flash devices.

IMPORTANT: Use of **`WriteFlash()`** is *highly* discouraged since it will work for some boards (with small sector flash) but not others (with large sector flash). If a large quantity of data needs to be stored, use battery-backed RAM (via `xalloc()`), `writeUserBlock()`, or the flash file system.

WriteFlash2

This function performs the same operation as `WriteFlash()` but writes to the secondary flash instead of the primary one. It is kept as a separate function since it is not necessary to block interrupts while writing to a flash device that does not contain running code (unless code is being run in the second flash as well).

Rabbit strongly recommends the use of the flash file system, but plans to keep the second flash as a small sector device, so `WriteFlash2()` will be a safe alternative for application use. A detailed function description can be found in the *Dynamic C Function Reference Manual* and in the Function Lookup feature from Dynamic C's Help menu.

writeUserBlock

Introduced in Dynamic C 7.04x3, the user block is 8KB (at least) of flash that is protected from normal writes. The original purpose of this area was to store calibration constants, but two functions, `readUserBlock()` and `writeUserBlock()`, were provided for the user to read and write to this area as well. Detailed function descriptions can be found in the *Dynamic C Function Reference Manual* and in the Function Lookup feature from Dynamic C's Help menu.

Support for the user block is retained for large sector devices. Starting with Dynamic C 7.20, the internal implementation is changed so that a two copy system is used. Two or three sectors (depending on the sector layout) are used to hold identical copies of the System ID and User block areas. This eliminates the need for storing sectors in RAM before erasing them.

A requirement for using the double copy scheme is that the System ID block be version 3. See the *Rabbit 2000 Designer's Handbook* or for a description of System ID blocks. If Rabbit ever ships a product with a large sector flash device, it will already have a version 3 type block, but board designers should install a version 3 type block if they plan to write to flash at run time. The only difference between a version 2 and version 3 block is the version number itself, but the special attention should be paid to the fields `userBlockSize` and `userBlockLoc`.

`userBlockSize` should be the size of the top sector of the flash minus `sizeof(SysIDBlock)`. The top sector must have either one contiguous sector of the same size below it, or two contiguous sectors of half the size of the top sector below it. (The latter is the usual layout of very large, non-uniform sector devices with top boot blocks.)

A program to write the system ID block is not released with Dynamic C, but is available at:

`http://www.rabbit.com/support/downloads/index.shtml`

Summary

Rabbit will continue to provide the best-fit solutions possible for embedded systems, changing market conditions notwithstanding.