TECHNICAL NOTE

TN217

Binary and Source Compatibility Issues for 4K Flash Sector Sizes

This technical note summarizes compatibility issues that could arise from a change in flash sector size to 4K from the smaller sector sizes currently used on single-board computers based on the Rabbit 2000 microprocessor. This technical note supplements an advisory notice that was sent out to existing customers of Rabbit Semiconductor.

The information in this technical note is primarily of interest to customers who are programming new units with precompiled programs that were compiled with versions of Dynamic C older than 7.20 using EPROM burners or "cloning" if their applications write to flash at run-time. This also applies to any customers who are using the Rabbit Field Utility (RFU) or Dynamic C to load such programs to production units.

Details on Compatibility Issues

Definitions

binary code—the compiled application stored in a file or in the flash of a target board.

binary-code-compatible—the same binary code will run on different devices when transferred there via EPROM burner, "cloning," or other means.

Recompilation Scenarios

1. Your application does not write to flash.

   No problems. Binary compatibility will be retained.

2. Your application writes to flash and was created with a version of Dynamic C earlier than 6.5x.

   The application is not binary-code-compatible with the 4K sector flash and should be recompiled with the latest version of Dynamic C. Some source code changes may be required where APIs have changed. In particular, several TCP/IP API changes were introduced in version 7.05. These differences are documented in the *Dynamic C TCP/IP User's Manual*.

3. Your application writes to flash and was created with a version of Dynamic C earlier than 7.20 but later than 6.5x.

   With the exception of Dynamic C version 6.57, the application is binary-code-compatible with the 4K sector flash, but the flash writing performance will be slower unless the code is recompiled using the latest version of Dynamic C. Applications written with version 6.57 that write to flash should be recompiled for 4K sector flash.

Writing to flash at run time, downloading, and debugging (especially downloading) will all be slow unless you upgrade to Dynamic C 7.20 or later. Some source code changes may be required where APIs have changed. In particular, several TCP/IP API changes were introduced in version 7.05. These differences are documented in the *Dynamic C TCP/IP User's Manual*.

Applications that depend on interrupts may have problems if not recompiled with version 7.20 or later because interrupts are turned off while flash is being written, and they will be turned off for a longer period of time with a 4K sector flash. ***If you recompile your application be sure to test it thoroughly.***

Tip—Store Persistent Data to the User Block

The new Dynamic C and flash drivers will always erase the primary flash on a board before loading a program. This erasure does not include the System ID block and the User block areas. Do not expect information in the primary flash to be retained between compiles except in those areas. The `writeUserBlock()` function is provided to store persistent data in the User block area. Secondary flash chips will not be erased unless the program extends into them. The secondary flash will also be erased if the program extends into the second flash. If the flash file system is used on the second flash, program code cannot be put there, so the files will remain intact.

It is best to use the function `writeUserBlock()` to write to arbitrary addresses in the primary flash in your application. Writes to arbitrary flash addresses on a 4K sector size flash using the `WriteFlash()` library function are supported with any Dynamic C version that has the drivers to support 4K flash sectors. However, should a flash sector size larger than 4K be incorporated in the future, the `WriteFlash()` function will not be backwards-compatible, but the `writeUserBlock()` function will be. `writeUserBlock()` uses a relative offset address into a reserved region of flash that is doubled-buffered in order to avoid excess RAM usage when writing very large sectors. Technical Note 216, *Is Your Application Ready for Large Sector Flash?*, contains more details about flash-writing issues for flash-sector sizes larger than 4K.