# Things to Consider when Porting to the PowerQUICC

As the next member in the family of integrated data communications controllers, the PowerQUICC retains a great amount of similarity between it and previous members of the Data communications family such as the 68302 IMP and the 68360 QUICC. The PowerQUICC is a natural migration from the 68360, providing greater serial power and CPU performance in a highly integrated package. It is important to remember that there are still many differences between the original QUICC and the PowerQUICC. It is not always safe to assume that a function in the PowerQUICC will work similarly to the function in the original QUICC.

This application note describes some of the things that a user may want to keep in mind when designing with the PowerQUICC. For more information, refer to the relevant sections in the User's Manual, application notes, and to the other related manuals such as the PowerPC Programming Environment manual.

## 0.1 PowerPC Core

The CPU of the PowerQUICC is now based upon the PowerPC architecture rather than the 68000. The processor architecture of the PowerPC is very different from the 68000. It is recommended that a designer enrolls in a course about PowerPC architechture before starting a PowerQUICC design.

## 0.2 Bit Labeling

It is critical to note that the bit labeling is reversed in the PowerPC programming environment from the 68000 programming environment. The actual arrangement of the bits is the same, but the high bit is now referred to as bit 0 and the low bit is referred to as bit 31(or 15, 7, etc..)

### 0.2.1 Code Portability

The PowerPC core is not binary compatible with the CPU32+ core on the 68360. Therefore, any code written will have to be ported to the new processor. If code was written in a high level language such as ‘C’, a cross complier may provide the simplest code migration path. Any 68000 assembly code will have to be re-written in C or in PowerPC assembly language.

## 0.3 Cache

The PowerQUICC has both an address and data cache. This is new relative to the CPU32+ core in the 68360, which had no cache at all. Cache allows the processor to run code much faster but may create pitfalls for both overall system performance and for debugging capability.

### 0.3.1 Cache Performance Impact

A cache can improve CPU performance dramatically if it is used effectively. However, turning on the cache does not guarantee higher performance. The degree to which a cache will help is directly related to the memory access pattern of the program running on the CPU. It is possible for the performance of a CPU to actually decrease in some situations.

A document of this size and breadth cannot effectively address the ways that a program can be optimized to achieve better cache performance. Some documented methods that can yield a significant improvement in access speed include locking cache blocks, making areas of memory uncacheable, optimizing program flow, and optimizing data access patterns. Note that these techniques are complex and their effectiveness can be altered by a simple program recompilation.

Caches may also complicate a benchmarking process. A benchmark, such as Dhrystone, will normally fit entirely in a cache and may give misleading indications of processor and application performance. Because of this, it is recommended that segments of code are run on an evaluation board in order to determine the performance that can be expected from a specific application.

### 0.3.2 Data Coherency

If the cache is in “write back” mode, changes made in the cache are not also made to memory. This can create data coherency problems if another bus master (such as an SDMA, IDMA, or external processor) accesses a memory location which has a more recently updated value in the cache. This can result in data loss or program execution problems. Thus, “write through” mode should be used for any memory area which might be accessed by another bus-master.

### 0.3.3 Debugging

The CPU32+ of the 68360 fetches its instructions from the system bus each time a particular instruction is executed. Thus, it is very easy to determine which instructions are executing at any given time by monitoring the external bus. It is also possible to monitor areas of memory to determine when reads and writes are taking place.

When a cache is placed between the CPU and the system bus, many of the instructions that are being executed will hopefully be out of the sight of the system, resulting in higher performance. Ideally, the CPU fetches instructions from cache rather than external memory. This results in these fetches no longer being visible, which may complicate a debug process. The same restriction applies to data reads and (in some cases) writes. This restriction is familiar to those who have worked with a 68040 or 68060.

The PowerQUICC has some additional functions in its background debug mode which allowing more debugging visibility than existed in the 68040. These functions are described in Section XXX of the PowerQUICC User’s Manual.

## 0.4 Memory Management Unit

Another function of the PowerPC core is provided in the MMU (memory management unit). The MMU allows the use of virtual memory and special caching options. The MMU on the PowerQUICC cannot be disabled and its’ use can be complex. Please see the application note “XXXX” to learn more about using the MMU.

## 0.5 Real Time Operating Systems

A Real-Time Operating System (RTOS) may simplify use of the MMU on the PowerQUICC. Most commercial RTOS implementations are designed to handle MMU and interrupt schemes . This allows the software designer to concentrate more on application specific code than the subtleties of the MMU and interrupts. If porting code from the MC68360 QUICC, do so cautiously keeping MMU, cache and interrupt handling as special cases.

## 0.6 The Communications Processor

The Communications Processor Module (CPM) in the PowerQUICC is almost identical to the CPM in the 68360. Therefore, most device drivers should be easily portable after the processor-specific details have been taken care of. If a driver is interrupt driven, the differences in the interrupt structure may cause some device drivers to behave differently from those operating on the 68360. In the case of problems with ported code,it may help to place a drivers in a polled mode of operation to isolate the source of the problem. Also note that some register locations have changed in the 860’s CPM, requiring a check of existing header files to assure accuracy.