APPENDIX B
PORTING FROM M68K ARCHITECTURE

This section is an overview of the issues encountered when porting embedded development tools to work with the ColdFire processor when starting with the M68K architecture.

B.1 C COMPILERS AND HOST SOFTWARE

For the purpose of this discussion, it is assumed that an embedded software development tool chain consists of a “host” portion and a “target” portion. The host portion consists of tool chain parts that execute on a desktop computer or workstation. The target portion of the tool chain runs ColdFire executables on a physical ColdFire target board.

Compilers, assemblers, linkers, loaders, instruction set simulators, and the host portion of debuggers are examples of host tools. Many host tools such as linkers and loaders that work with the M68K architecture can also be used without modification with ColdFire.

Although you can use an existing M68K assembler and disassembler with ColdFire, Motorola recommends modifying the assembler so that nonColdFire assembly code cannot be put together in the executable. This is especially true if the assembler assembles handwritten code. Porting the disassembler is for convenience and can be performed later.

Debuggers usually are comprised of two parts. A host portion of the debugger typically issues higher level commands for the target portion of debugger target. The target portion of the debugger typically handles the exact details of the implementation of tracing, breakpoints, and other lower level details. The debugger host portion requires little modification. Most likely, the only architectural items of concern are the following:

- Differences in the designed supervisor registers and stack pointers (for displaying registers)
- Interpretation of exception stack frames (if not already performed by the target portion)

B.2 TARGET SOFTWARE PORT

Porting ROM monitors and operating systems can begin after the compiler and assembler have been ported. For example, consider the steps involved in porting a ROM debugger. Similar issues are encountered when porting an RTOS and target applications.

It is assumed that target software consists of C and assembly source code. The first step is to create executables that will run on existing M68K hardware to test the conversion from M68K code to the proper ColdFire subset. This step verifies that the process of code conversion does not introduce new errors.

To generate a ColdFire executable of the target debugger, you should use a ColdFire-compliant port of the same C compiler originally used to create the M68K debugger target. This procedure prevents differences in calling convention and parameter passing from C to handwritten assembly. Another advantage to this approach is that special C flags are retained. Many C compilers have special extensions as well.

Whichever approach is used, the assembly language lines that are outside the ColdFire instruction set must be identified. Any ColdFire assembler that properly flags nonColdFire instructions can be used. During the process of conversion, you can ignore architectural issues temporarily because the target is still an M68K.

Once the instruction set differences have been resolved, the architectural differences between the ColdFire and M68K need to be addressed.

B.3 INITIALIZATION CODE

The target software and firmware often execute code that identifies the type of processor. Such a process provides one port that works with various M68K Family members and implementations. The easiest way to identify the ColdFire architecture from other M68K processors is to execute an ILLEGAL opcode ($4AFC). This execution generates an exception stack frame while ensuring that the tracing is disabled. The first two bits of the exception stack frame would immediately determine whether the processor is a ColdFire processor.

Motorola suggests that ColdFire architecture testing be performed immediately to avoid executing potentially undefined opcodes in the ColdFire architecture. Unused opcodes in the ColdFire architecture are not guaranteed to result in an illegal instruction exception.

Another item to consider is that the ColdFire architecture will have integrated versions with modules yet to be defined. It may be a good idea to ensure that there are enough hooks to allow for initialization of routines.

B.4 EXCEPTION HANDLERS

When dealing with exceptions in debug-oriented software, it is often necessary to extract exception stack information to obtain the SR and PC. The format word (MC68010 and higher) is typically used by generalized exception routines. Their sole purpose is to catch unexpected exceptions and to easily use vector information to identify the cause of the exception. The MC68000 exception frame is different from that of other members of the M68K processors in that there is no notion of a format word. This difference would have forced target software to deal with exception stack frame differences already. The approach now in use provides guidance on handling ColdFire exception stack frame differences. In many low-level exception handlers, the extraction of the stacked SR, PC, or format word is performed in a common source file or the offsets are handled in some type of header file.

Interrupt handlers probably require no modification because in most cases, an interrupt occurs asynchronously with respect to normal program flow. Therefore, interrupt handlers cannot rely on items on the stack as it is often unnecessary to know exactly what was happening at the time of the interrupt.

System calls are often implemented by using the TRAP instruction. For trap exceptions, parameter passing is performed through data and address registers—rarely, if ever, directly through the stack. In addition, a system call typically does not need to know the stacked SR or PC information.

Breakpoints are usually implemented with the TRAP instruction or an illegal instruction such as an $A-line exception. If so, the stacked SR and PC are typically used. Other items in the stack may also need to be queried, especially if the breakpoint displays a stack trace. If so, you should examine the format closely for stack misalignments at the time of the breakpoint. This stack misalignment check would be useful in applications where stack alignment is a software design goal. These same concerns for the breakpoint implementation are applicable to trace exceptions as well.

A generalized exception handler can be implemented to catch unexpected exceptions. In addition to the SR and PC information, it is often necessary to obtain the vector information in the stack. Otherwise, the issues are similar to those found on breakpoints and tracing.

To port the ColdFire access error exception, it is best to start with an MC68000 bus error handler. The ColdFire access error recovery sequence has many similarities to the procedure recommended for the MC68000. However, you should be aware that a read bus error on the ColdFire will not advance the program counter to the next instruction. In addition, a write bus error may be taken long after an instruction has been executed and the stacked program counter may not point to the offending instruction.
The main cause of an address error exception in the M68K architecture is that program flow is forced to continue at an odd address boundary. In addition, an MC68000 reports an address error if a data byte access is initiated on an odd address. The ColdFire uses the address error for implementations that do not have the misalignment module. A misaligned data access is then initiated. Modification of the address error exception handler to reflect a ColdFire misalignment exception is optional. The MCF5202 contains hardware support for data misalignment and therefore this is not an issue for family members.

On a ROM monitor, it is often necessary to provide a means by which a user program is executed given a certain starting address. This is often implemented by placing an exception stack frame and then performing an RTE. If this is the case, the header files that define what a stack frame looks like would require modification to reflect the ColdFire stack frame format.

B.5 SUPERVISOR REGISTERS

The target software would eventually need to communicate the contents of registers to the host software. Both the host portions and target portions of a debugger must be modified to reflect the single stack pointer architecture of ColdFire. In addition, the target debugger must keep a copy of the MOVEC register images in memory so that it can provide the host software register contents when asked to do so. A UNIX grep utility can find all instances of the MOVEC instruction and perform the appropriate modifications to accommodate the unidirectional MOVEC instruction.

The ColdFire architecture does not distinguish between a supervisor stack and a user stack. There is only a single stack pointer, A7. One way of dealing with this issue is to emulate the

dual stacks by placing some code at the beginning and end portions of exception handlers to change the stack pointer contents, if necessary, during exceptions. This approach has the disadvantage that interrupt latency would be degraded because interrupts would have to be disabled during the stack-swapping process, but enable full flexibility of the 68K stack model.