Design Note: Target Considerations for Coldfire[†]

Jonathan S. Shapiro, Ph.D.
The EROS Group, LLC

Dec 1, 2007

Abstract

Documentation of gross issues and considerations in porting to the ColdFire V4e platform.

1 Introduction

This document addresses design considerations in porting the Coyotos kernel [1] to the ColdFire V4e processor family. This note specifically considers issues in porting to the MCF5485, but most of the issues raised and their solutions should be common to other members of the family.

The ColdFire V4e family generically meets the functional requirements to run Coyotos. It implements a paged memory management unit and suitable user vs. supervisor protection isolation. Privileged registers can be configured for supervisor-only access. Handling of sensitive registers is less careful, but these registers primarily disclose boot-time configuration information; there do not appear to be any substantive disclosures of interest to applications through these registers.

With these issues addressed, the following issues remain to be considered:

1. Feasibility of family compatibility with a single kernel.
2. Selection of Coyotos kernel-defined page size for this processor family.
3. Memory management subsystem and possible implementations of the Coyotos shadow translation mechanism for this architecture.
4. Design of processor cache(s), interactions with the memory management mechanism, and any required special handling for these.
5. Interrupt and trap frame formation, system call entry/exit mechanism(s).

[†] Copyright © 2007, Jonathan S. Shapiro. Verbatim copies of this document may be duplicated or distributed in print or electronic form for non-commercial purposes.

THIS SPECIFICATION IS PROVIDED "AS IS" WITHOUT ANY WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

2 Family Compatibility

Different members of the V4e family place certain key configuration registers at incompatible memory addresses. This has been an abiding problem for the GNU binutils maintainers, the problem has been explored fairly carefully. The current assemblers can be used to generate implementation-specific code where required, but there is no reason to believe that the Freescale team views forward supervisor-mode compatibility as a design objective.

Open Issue There is no published algorithm for performing CPU model identification within the V4e family. It is likely that something can be worked out, but since it isn't a pressing issue for any current users of this family, there are no immediate plans to investigate this further.

3 Page Size Selection

The ColdFire architecture supports a range of page sizes. The Coyotos kernel architecture is designed in terms of a single least page size that forms the atomic unit of address space construction. On the ColdFire, the hardware-supported page size choices are 1K, 4K, or 8K.

Our original plan was to push hard for a 4K page size in order to minimize deviation from the i386 implementation and to improve page utilization. The problem with this choice is that (1) the cache set size is 8K, (2) the cache is virtually indexed by the least 13 bits of the issued address, and therefore (3) if the page size chosen is smaller than the set size cache inconsistency results. The original plan called for a coloring strategy. Discussion of the coloring issues is retained in this document for the sake of later

review, but that discussion does not apply to the current implementation.

Because of the cache consistency issues, an 8K page size was adopted for this implementation. Linux has also adopted an 8K page size for this architecture.

Notwithstanding the general page size decision, it may be advantageous to use 1M mappings to describe the kernel region in order to reduce kernel TLB overhead.

4 Memory Management Subsystem

The V4e family implements separate soft-loaded TLBs for code and data references. Each TLB has 32 entries and is fully associative. Entries can be individually pinned in the TLB, but there are no plans to exploit this in the initial port except as may be required to support the software translation architecture. Future implementations may exploit this feature more aggressively.

The use of a split I/D TLB architecture lets us enforce the Coyotos NX permission bit in software.

The soft-loaded architecture implies that most kernel virtual addresses can be reloaded without any lookup. Aside from the transient mapping region, the kernel portion of map can be configured as Virtual=Physical+constant. This permits very fast reload of supervisor-mode addresses following a simple bounds check.

The soft-loaded architecture also gives us an efficient way to *unmap* temporary mappings. The main challenge we will face with temporary mappings is the need to honor map colorings within the fast path.

Initial Translation Strategy As an initial implementation strategy, we are going to try to operate with no software-managed second-layer TLB. We will instead implement a TLB software miss handler that knows how to reload kernel virtual addresses through an internally maintained table, and which handles user-mode address faults by traversing the GPT structures. No DependTable is required; the TLB is simply flushed whenever a GPT store is performed, which wipes out all mappings. Similarly, no RevMap is required.

This strategy will get us running, but it obviously won't be suitable in production. Note that this design puts enormous pressure on the TLB. Our target has 128M of memory, the kernel heap is approximagely 1/120th of this, and assuming that 1M mappings can be used for the kernel heap they may well occupy 13 of the 32 available TLB entries.

Improved Approach for production use, the current plan is to back the hardware TLB by a second-layer TLB implemented in software. This lowers GPT walk pressure directly, which in turn lowers kernel-induced TLB pressure. The challenge will be to index the second-level TLB so that it can be searched using either a virtual or a physical address. A multi-set approach may work here.

In order to support this, the RevMap must be reintroduced, and a DependTable is once again necessary. DependTable pressure in this design will be high, because we cannot exploit run-length encoding in the depend structure. One way to reduce this is to simulate bottom-level page tables, but it is not clear that this would reduce dynamic DependTable pressure significantly.

Another alternative is to come up with a way to cache partial GPT traversals. We might build a cache of translations from translation roots up to but not including the leafmost GPT. At page fault time, we can consult this cache to quickly find the leaf GPT and then search that GPT to find the desired target of the mapping.

The idea of a traversal cache may be less pressing in Coyotos than in EROS, and the first thing to do is certainly to implement a fast soft traversal of GPTs directly.

5 Cache Consistency Issues

> **Note**
>
> This section is now obsolete. It is retained for historical reference.

The V4e family cache is virtually indexed and physically tagged. On the MCF5485 the set size is 8K. Given a 4K page size, this creates some messy coherency issues if virtual and physical addresses do not match in the least 13 bits.

The current proposal is to keep track of the active mapping colors for each physical frame. As long as all mappings are read-only, simply ignore the incoherency. The instant that a writable mapping exists, invalidate all mappings of non-matching color and hand-flush the relevant parts of the cache.

The implementation is a nuisance, but the practical performance implications of this are expected to be small:

- Shared code pages are never observed to be written in the wild.
- Purely read-only pages are not impacted.
- Private writable pages are impacted only if there are simultaneous non-congruent mappings. This is not observed in the wild.
- Shared pages are almost always mapped with compatibly congruent virtual addresses in order to gain

the benefit of mapping metadata sharing. The practical consequence of this is that shared *writable* pages almost always turn out to have a single color in practice, and in this case the cache flush can be avoided.

If we observe that the coloring logic is actually getting triggered in practice, we may want to configure the cache in write-through mode.

When you are done wretching over this, consider that it could be worse: on the ARM it is necessary to guarantee that a writable page has exactly one mapping at a time, and further necessary to flush the entire cache when that mapping must be changed. On ColdFire we only need to flush *half* the cache. Oh frabjous day.[1]

Open Issue Section 5.2.3.5 of the MCF5485 reference manual [2] states that the cache uses the virtual address for indexing but the physical address for allocation, and notes that multiple mappings may lead to incoherency in the cache. However it appears that the incoherency can only arise in non-interacting cache sets, so this may be acceptable if no writes are possible.

However, it is possible that this statement is entirely literal. In that case, we will really need to guarantee that bit 12 must really match in the virtual and physical addresses at all times. If so, then attempts to map at a miscolored virtual address may require that we "bounce" the physical page to a compatible physical frame. This would be inconvenient, but it's doable. For the reasons given above it is unlikely to arise in practice, but it would be a shame if it actually proves to be necessary.

6 Interrupt and Trap Frames, System Calls

The V4e family implements a common exception frame for all interrupts, traps, and system calls. In contrast to IA-32, Interrupt and trap vectors do not overlap in the vector table. There is no specialized SYSCALL instruction. System calls are accomplished using the volitional TRAP instruction, which can dispatch to any of 16 trap vectors. All of these cause transition to supervisor mode and switch to a supervisor stack pointer. The regularity of the trap interface means that no system call entry or exit trampoline is required for this architecture.

One misfortune is that the TRAP instruction does *not* disable interrupts. This may preclude a fast register save design within the kernel as was done in the IA-32 implementation. The intention is that software should do this by immediately executing something like:

```
mov  %sr,$disable
... proceed with save ...
```

Interrupts are suppressed until this first instruction is run, so nested interrupts can be controlled.

Earlier versions of this document expressed concern that interrupts in the register save path would preclude the fast-save design. They do not, but another consideration does: the TLB miss fault.

Because this is a soft-translated architecture, a TLB miss fault generates an exception. The exception frame is pushed to the stack. If no kernel stack is available to receive this exception state, the processor becomes very unhappy. Unfortunately for us, it is not practical to pin the mappings for the Process vector. This means that we cannot save directly to the Process structure, and we will therefore be unable to use an IA32-style fast save strategy.

The good news is that the hardware has separate user and kernel stack pointers, so we can probably get away with testing the origin of the fault (user vs. kernel) and in the user case save only one or two registers, load the current process pointer, save the remainder to the process state and then migrate the registers and the trap information.

On this architecture, the only privileged state in the integer save area is the SR register state, which needs to be handled specially during reload in any case.

7 Platform Memory Map

This section addresses the configuration of the device memory map for the target.

7.1 Memory Organization

Virtual Addresses In the *virtual* map, the Coyotos implementation will reserve the [0x0,0xBFFFFFFF] range for user-mode addresses. The range [0xC0000000,0xFFFFFFFF] is reserved for kernel use. This is also the Linux convention. Unfortunately, the distinction between virtual and physical addresses on this processor is not entirely clean.

Physical Map The Coldfire is an integrated single-chip controller, and it implements chip select for memory devices on the processor. Internally, post-translation addresses are delivered to the XL bus. The XL bus in turn has an SDRAM controller, a "flexbus" controller, two small (4k) SRAM devices, and various other items. External RAM memories are connected to the SDRAM controller. External ROM/FLASH memories are connected

[1] Read *Through the Looking Glass*. Do not pass go. Do not pause to admire yourself in the mirror. If you happen to run into an odd, heavyset, self-absorbed hare, say hello for me.

to the flexbus. The processor implements chip select pins that can be more or less directly wired to these outboard devices. For each device, including those implemented internally, there is a base address register and an address mask register that determine when the corresponding chip select goes active. This means that the locations of devices can be revised in software.

It is possible, but not advisable, to contrive for the chip select addresses of devices to overlap. Don't do this! If the memory chip designs aren't right, a short can be induced on the data bus as one chip drives D1 high and the next drives it low. I have not investigated whether the processor may implement a priority scheme in the chip select logic to preclude this problem.

MMU Placement The MMU on the Coldfire was a bolt-on, and the resulting bussing structure is peculiar. In particular, there are two on-board SRAM devices (controlled by RAMBAR[0:1]), two on-board ROM devices (controlled by ROMBAR[0:1]) and two access control registers ACR[0:3] that define address regions. Each of these registers describes an arbitrary region of the address space that is some power of two in size. Any address matched by these registers is passed to the post-translate bus without modification. There is a translation priority scheme, and the TLB sits *lower than* any of these in that scheme.

There is also an MBAR register that specifies the register base address of the "System Integration Unit". While no translation priority is specified for MBAR, the discussion of the ACR registers indicates that one of these should be used to make the SIU registers non-cacheable.

The net effect of this is that regions defined by RAMBAR, ROMBAR, or ACR are "in front of" the mapping structure. In the context of our implementation this means that all such addresses must be *above* 0xC0000000.

SRAM Considerations The two small SRAM devices controlled by RAMBAR[0:1] are on a local bus. The MC5485 manual is explicit that a hit in RAMBAR[0:1] causes chip select to be asserted to these small SRAMs. It is not clear whether these devices can also be addressed via the MMU. There is a diagram of address pipeline flow in the MMU chapter which suggests that the post-TLB addresses and the post-RAMBAR addresses both end up on the XL bus. It is not clear if SRAM chip select is then done from the XL bus or if it is done from the RAMBAR address matching logic. I suspect the latter, in which cases the devices controlled by RAMBAR[0:1], ROMBAR[0:1], and ACR[0:3] cannot be mapped via the MMU. This is particularly unfortunate for ACR[0:3], because it means that device registers cannot be mapped for application access, and we will need to implement a "load/store I/O register" sysem call.

7.2 Power-on Conditions

At power-on, the external device attached to FlexBus chip select zero (CS0) is the only device enabled. It is configured at a base address of zero with an address mask of zero. It therefore covers the range [0,0xFFFFFFFF]. The bus width and latency of the corresponding device is determined by pin-strap. For the ASD board, the boot flash latency is 70ns and the code flash latency is 85ns.

All other device are initially disabled. It is the responsibility of the bootstrap software to establish base addresses for all other memory devices and appropriately enable their supporting chip select logic.

The documentation of the reset exception indicates that initial supervisor PC and SP are loaded from addresses 0 and 4 respectively. The documentation of the exception vector reserves positions for these. This seems to imply that the exception vector used by the bootstrap code wants to be the first thing in the boot flash. After reset, there is no obvious relationship between the exception vector and the PC/SP, but the exception vector is required to appear at a 1M boundary in memory.

7.3 Hardware Constraints on Memory Map

When the TLB is disabled, there is no particularly unusual complexity inconfiguring this processor.

TLB Interactions When the processor is running with translation enabled, matters are more complicated. A TLB miss is handled as an exception. The processor handles exceptions, including TLB exceptions, by pushing an exception frame to the supervisor stack and proceeding through the exception table (which must be 1Mbyte aligned). This means that the kernel stack, exception vector region, miss handler code, and any data structures used by the soft-load fast path must either be pinned in the TLB or must be stored in non-mapped memory. It is not clear from the documentation whether the interrupt vector fetch is performed as an I-reference or a D-reference. We will need to experiment.

One approach would be to place the kernel stack in RAMBAR0, put the kernel exception vector and the fast path TLB data structures in RAMBAR1, and then pin an entry for the TLB miss handler code in the I-TLB. Alternatively, if the exception vector references are I-references the vector table can be placed at the front of code space.

The TLB miss processing issue, and its associated potential for double faults suggests that the bootstrap code should run without the TLB enabled. There is no obvious reason to enable the TLB for that code, and avoiding the need for any software miss handler seems worthwhile.

7.4 Software Constraints on Memory Map

Because they must not appear within the user portion of the address space, the MBAR and RAMBAR[0:1] registers must be configured at addresses above 0xC0000000. While the system integration unit (base address controlled by MBAR) is not prioritized relative to the TLB, the manual indicates that one of the ACR registers should be programmed to preclude caching to those addresses. It is not clear if that statement in the manual is a legacy statement predating the introduction of the MMU.

Coyotos kernel memory starts at 0xC000000. It is hypothetically possible to configure the SDRAM base address at 0xC0000000 and then program the ACR[0] (data) and ACR[2] (code) to treat the range starting at 0xC0000000 as a pass-through supervisor-only range. Doing so ought to have the effect of significantly reducing kernel TLB overhead – especially so if the kernel heap directly follows kernel data.

Note that doing this does not preclude pointing portions of the user range into the SDRAM via the TLB. What it *does* do is allow us to offload any power of two kernel mapping that can be described by a `va=pa+`*`const`* relationship. Since we *ought* to be able to cover most of kernel data and heap this way, this seems likely to significantly reduce kernel TLB pressures.

8 Boot Flash and Program

Our current board design calls for two flash devices. Provided the flash can be programmed pagewise, a single flash device might suffice in future variants. Note that the boot flash uses SRAM0 and SRAM1 as working storage. It cannot use the lower parts of SDRAM because it will be overwriting them.

Responsibilities of the bootstrap program are:

1. Quiesce the hardware, as needed.

2. Configure the SDRAM, code flash, a hole for boot flash, the CPLD device, and LCD device in sequence starting at 0xC0000000. Use ACR registers to set up a large region that spans the SDRAM. We *may* re-set the boot flash base address early, or it may be more convenient to let the kernel do it later when we are no longer running from it.

3. Initialize the CPLD device, notably including the out-board logic that disables CS0 if the boot image fails signature check.

4. Configure SRAM0 for working data storage.

5. Perform a cryptographic signature check on the kernel image. Separately perform a cryptographic signature check on the initial system image. If either check fails, make note of this for later use.

6. Copy the code flash kernel image to SDRAM. Note that at this point we can no longer use lower SDRAM for working storage.

7. Configure SRAM1 for data references. Copy trampoline code to SRAM1. Trampoline code may later alter the CPLD state depending on the preceding cryptographic check result.

8. Re-configure SRAM1 for instruction references. Disable interrupts and branch to trampoline.

9. Conditionally disable chip select for boot flash using CPLD and branch to Coyotos kernel start address.

Issue The boot flash will refuse to boot an unsigned kernel. If the code flash becomes corrupted, should we fall back to PXE boot for re-install?

Issue When ASD's signature key becomes stale it will necessary to do an update dance on the boot flash. There is an interim state where two signature validation keys must be present and either is okay. What happens if a customer does not stay up to date?

Issue The upgrade agent on the code flash should probably refuse to install any upgrade that the boot flash will not pass.

Issue We probably need a two-key protocol, where the second key is used *only* to sign key updates.

9 Code Flash and Kernel

The kernel will begin executing having already been loaded into SDRAM. At control transfer interrupts will be disabled. The first few instructions of the kernel will make any required changes to the device memory address configuration, notably including setting up appropriate mappings for kernel use of SRAM0 and SRAM1.

In fact, it is probably a good idea for the kernel to assume that any configuration provided by the boot flash is completely broken. The main reason for this is that the environment bequeathed by the boot flash will not match the environment bequeathed by the prototype board, and it would be good to have a kernel than can be run for testing after directly loading it to memory.

References

[1] J. S. Shapiro, Eric Northup, M. Scott Doerrie, and Swaroop Sridhar. *Coyotos Microkernel Specification*, 2006, available online at www.coyotos.org.

[2] Freescale, Inc. *MCF548x Reference Manual*, Jan 2006, Document Number MCF5485RM, Revision 3.