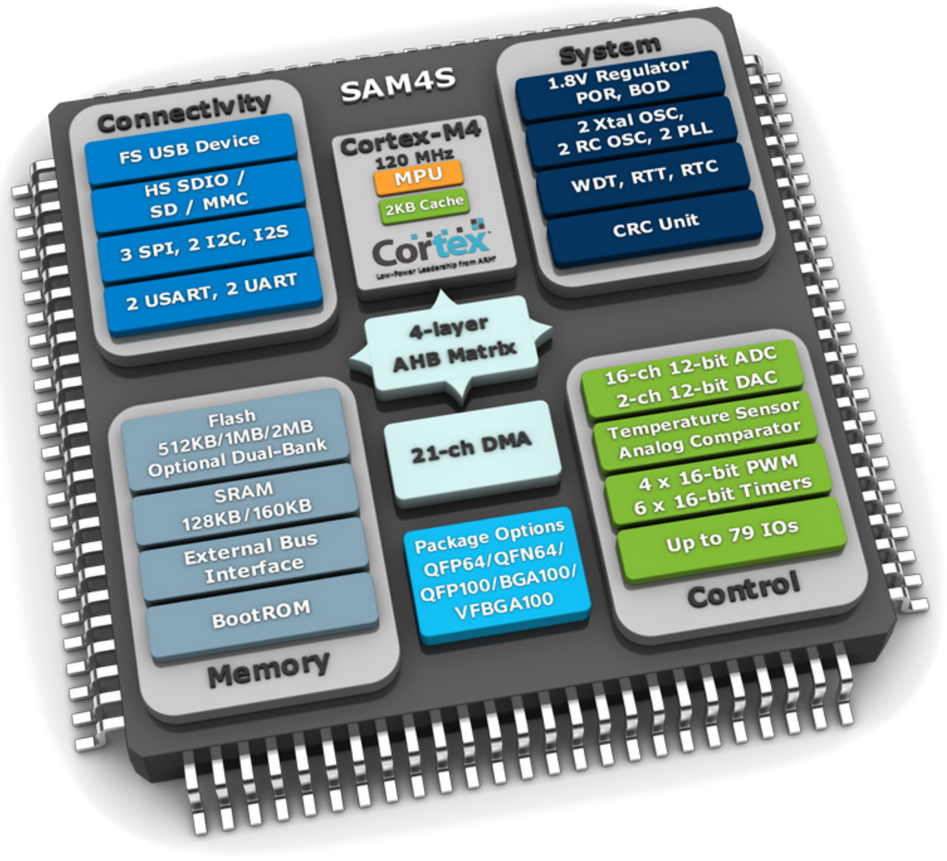
Cache-as-RAM to bring up non-booting boards



eBook

by Larry Osborn

ScanWorks®
Platform for Embedded Instruments

By Larry Osborn

Larry Osborn, Product Manager Processor-Controlled Test, at ASSET InterTech, has over 25 years of experience in product management, hardware/software product design and development, product delivery to the marketplace and user support. Over the years, Larry has a proven track record for identifying user needs and opportunities in the marketplace, providing innovative solutions and exceeding the expectations of users. At ASSET, Larry is responsible for the profit and loss for a product group. Prior to ASSET, he has held positions with Lockheed Martin, OCD Systems, Windriver, Hewlett-Packard, Ford Aerospace and Intel® Corporation. He holds a Bachelor's Degree in Computer Science from the University of Kansas and various technical and marketing training certifications.

Table of Contents

Table of Figures

Executive Summary

Design engineers often must diagnose and debug circuit boards that will not boot the bare metal firmware or have no BIOS at all. And, without some sort of software running on the board, comprehensively diagnosing structural and functional deficiencies is practically impossible. One effective way to get around this catch-22 is to use the on-chip cache memory in the board's processor instead of discrete on-board RAM memory space to execute diagnostic and test routines which will identify faults that could be causing boot failures, such as failures in the memory controller or the links to it. Instead of simply powering up the board and discovering it won't boot, a relatively short, yet deliberate step-by-step methodology will actually save time by identifying hardware faults that could prevent the boot process. In addition, the circuitry on a prototype board might be damaged if power were applied to it without an initial test phase. Cache-as-RAM techniques could be incorporated into this process because they are an effective way to bring up prototypes and verify hardware functionality without relying on fully functional and tested operating firmware.

Although executing boot code out of cache is not a straightforward process, implementing cache-as-RAM debug can be accomplished effectively and quickly with the proper run-time control tools that allow designers to quickly deploy script-based and code-based debug routines to verify a board's hardware functionality.

Please note that this white paper does not discuss cache architectures, the merits of Harvard vs. von Neumann architectures or cache behaviors. For information on some of these issues, see the papers listed here: "An Overview of Cache" and "Harvard Architecture".

Stop assuming. Reduce risk.

When assumptions are made, risk increases. All too often, engineers assume a circuit board has an acceptable level of structural integrity; that all the solder connections are good; that the devices on the board are correct and not counterfeit; and many other issues as well. By making these sorts of assumptions instead of quickly verifying the structural integrity and functionality of the hardware, the design engineer runs the risk of actually extending the board bring-up phase and possibly damaging the board when power is applied. By adopting a methodical, consistent and repeatable approach for each and every prototype that must be brought up, cycle times can be reduced over multiple boards and accidental damage avoided.

During prototype board bring up, the engineer may be tempted to simply apply power to the board, observe what happens and hope for the best. Too often, though, this can leave the design with so many unanswered questions that the board bring-up or debug process is unnecessarily lengthened to the point where it may jeopardize the new product introduction schedule. If the board fails to boot, engineers face a myriad of questions, such as: Is the memory controller accessible from the CPU? Is the output from RAM memory functioning properly? Are the clock and selection control signals to RAM operational? The list of possible failures, faults and malfunctions could go on and on. A methodical, inside out strategy based on run-control test processes, structural testing as well as functional verification can avoid this long list of questions and the ensuing confusion that could threaten the product's time-to-market. Part of this methodical approach should be to enable cache-as-RAM debug procedures for non-booting boards. (Figure 1.)

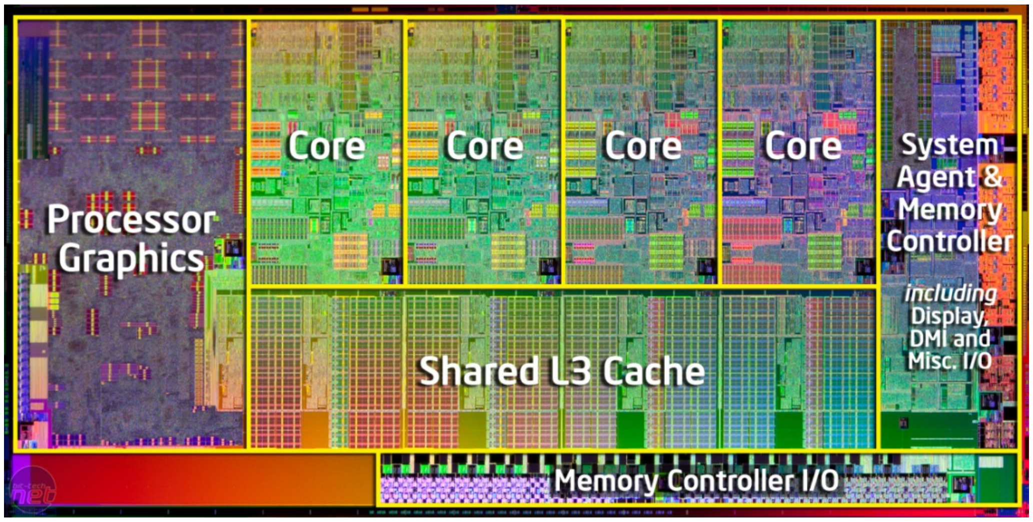


Figure 1: Microprocessor architecture with cache – Intel®'s Sandy Bridge microarchitecture features core-specific L1 and L2 cache, as well as L3 cache that is shared among four cores. L2 cache is used for cache-as-RAM implementations. (Source: Intel Corporation)

Inside out run-control debug

Most microprocessors today have run control, which means design engineers can start and stop the processor, insert breakpoints into code where the processor will stop so results can be analyzed, and other control mechanisms which facilitate the debug process via an external run-control tool, sometimes referred to as an emulator or probe. Some hardware assisted software-driven tools are able to take advantage of a processor's run control by placing the processor in debug mode where the device's internal cache can be a resource for debug and test. Unfortunately, using the processor's cache as the RAM space from which the board is typically booted is not as simple as just loading the boot firmware into cache without taking other factors into consideration. In some cases, development of the boot firmware has not yet been completed. But even when boot code is available, it cannot be loaded directly into cache, unless the boot code was specifically designed to use the cache. (Figure 2)

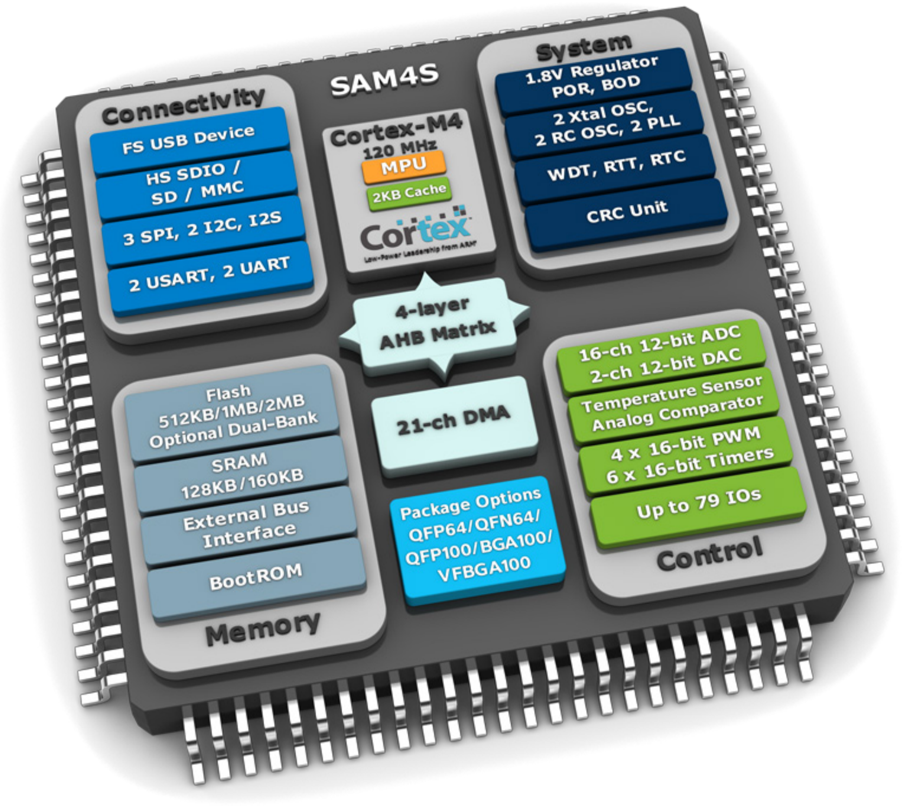


Figure 2: ARM Cortex M4 architecture – Atmel's SAM4S device based on the ARM Cortex M4 core includes two megabytes of embedded flash memory and optional on-chip cache memory. (Source: Atmel Corporation)

When boot code is available

When a prototype circuit board will not boot, the design engineer can turn to cache-as-RAM to determine whether the boot code failed because of faults in the memory controller, the RAM or flash memory, or the interconnects between the CPU, controller and memory. Better yet, these structural and functional aspects of the board can be tested with run-control tools before boot code is even loaded into RAM. Run-control tools are sometimes used to deploy cache-as-RAM debug processes, but some tools are capable of performing a much wider spectrum of test and debug routines, which can be invaluable during prototype board bring-up.

The first impulse of the engineer may be to simply recompile the boot code, load it into cache and execute. Because of the many differences between cache and RAM, this will not work. For example, the structure of the code may prevent a direct recompilation. Data structures could be fixed in size, but dynamic in nature. If so, these structures must be placed somewhere in cache where they will not affect code or stack space. Also, data structures could be hardcoded for read-only memory (ROM). Maybe there are sections of the boot code that point to areas on the board that are not functional. With many bare metal designs and BIOS code blocks, the tool chain used to create the code is expecting memory for a stack to accommodate function/procedure calls, which could target either RAM or cache. If targeted at RAM, the code has made the assumption that RAM is functional. If the target is in cache, how can the code possibly initialize the cache while it is executing out of cache? Is it instruction or data memory? There may be mode restrictions that affect boot code execution out of cache. The cache architecture may limit code execution to instruction memory and not support executing code from data memory. Placing a RAM-based stack in cache might cause the stack to extend into code execution memory space. Constants should also be examined to determine whether they are address-relative hardcoded data. Object addresses could also be a problem, depending on where they point.

Device initialization is usually an integral part of boot code. Placing this sort of code into cache could affect the cache itself. For example, segments of boot code might have been written with the assumption that the cache can be accessed and utilized during the boot-up process. If this were the case, boot code executing out of cache could conceivably write over itself. At the very least, the risk is great that the boot process will not go smoothly.

If the engineer is undertaking cache-as-RAM debug, these and many other issues must be considered before booting firmware can be executed out of cache. Of course, certain run-control tools are able to perform many of these functions automatically, saving the engineer significant time and shortening the debug process considerably.

When boot code is not available

When circuit board prototypes are assembled, boot code firmware is often still being developed. As a result, the design engineer does not know whether the prototypes will boot or not. This is the proverbial catch-22 again. What he can do with run-control tools is eliminate hardware faults from the list of reasons why the board might not boot even when the firmware eventually does become available. If the board will not boot when the firmware is eventually loaded, the problem must be in the firmware itself because hardware faults have already been ruled out. The finger-pointing between hardware and software development groups is reduced, if not eliminated.

In order to accomplish this, the run-control tools must be able to interface with the prototype boards. Providing rudimentary peek-and-poke routines for registers, input/output (I/O) channels and memory is a good starting point. The tool should also support the ability to verify that on-board devices will initialize; to accomplish this, most components will require that some data is written into registers. This data initialization may require a sequence of writes to address the device properly as specified in the data sheet. A run-control tool that supports scripting or some method of sequencing will be required to initialize the board's memory controller. Even then, some memory controllers are so complex that some code from the controller's supplier will be needed.

Diagnostics

Another area where cache-as-RAM can be used by in conjunction with hardware assisted run-control tools is in the area of diagnostic. Diagnostics are generally designed to provide detailed information about failures. These messages are often verbose and targeted at an I/O device (printer or monitor). In addition, diagnostics routines are typically executed from RAM. When a prototype is being debugged, often the I/O and RAM are operationally questionable at best. If diagnostics are available, they can be retargeted to execute from cache and the output retargeted to cache. The run-control tool can control the execution of the diagnostics and then interrogate the cache for the diagnostic output. Using cache-as-RAM in conjunction with hardware assisted run control can bring additional value in diagnosing faults and failures on circuit boards. Here again, the run-control tool should be analyzed to determine whether it supports the diagnostic functionality typically required by today's high-speed board designs. For example, debug engineers should consider the type of diagnostic information provided by the run-control tool as well as the output device utilized by the tool. Alternative means of outputting diagnostic data such as a non-I/O device may be required if the default output means fails. How diagnostic data is collected is also important.

The user should also consider how the tool reacts and what diagnostic information is provided when it encounters a device failure. In some tools, the failing device is simply disabled and logged, which does not provide significant debug information to the engineer. For example, it is not very helpful to an engineer if the tool were to exit at the first failure it found while trying to initialize a large memory architecture. The tool should have some means of retaining executing code in cache even when a device or larger failure is encountered.

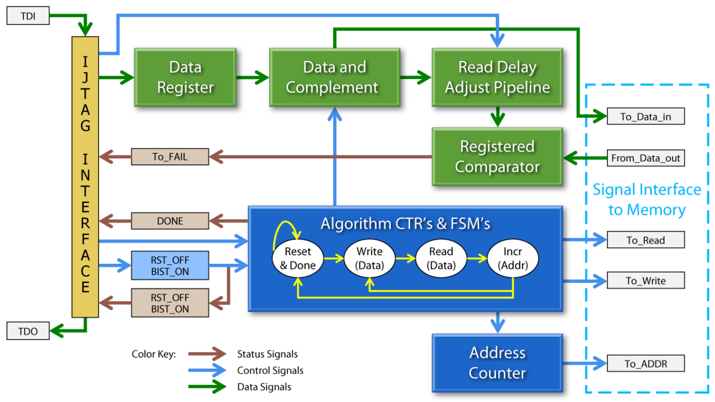
Conclusions

Although cache-as-RAM can be a useful method for design engineers who are bringing up and/or debugging prototype circuit boards, it is but one of the many aspects of using CPU resources in conjunction with powerful run-control tools. Because of the number of issues that must be considered to employ cache-as-RAM methods, capable software-driven run-control tools can not only help accelerate cache-as-RAM techniques but other debug methods as well. Some run-control tool platforms are able to debug both structural and functional aspects of circuit boards, saving the design team considerable time and effort. Choosing the right run-control tool should involve evaluating the tool supplier's understanding of the complex interactions between hardware and software.

For more information on run-control tools and how processor resources can be applied to board bring-up, validation and debug, check out ScanWorks® Processor-Controlled Test tools.

Learn More

Learn about memory test methods and their tradeoffs. In particular some of the complexities in testing high-speed DDR memory buses.



Register Today!

ScanWorks®
Platform for Embedded Instruments