Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

ARM Dons Armor

TrustZone Security Extensions Strengthen ARMv6 Architecture

By Tom R. Halfhill {8/25/03-01}

"Trusted computing" is such a hot topic that a dictionary editor recently asked *MPR* if she should include the term in the next edition she's compiling. Nothing validates a trend like the migration of technobabble to everyday language. To make designing secure embedded systems easier, ARM is adding new security extensions to the ARMv6 architecture.

The new TrustZone extensions are relatively simple, consisting primarily of one new instruction, a new configuration bit, and an additional permission level that supplements the existing user and privileged modes. TrustZone is simple because ARM's objectives are limited. Instead of trying to make the entire system an impregnable fortress—a goal ARM considers unrealistic—TrustZone allows small amounts of security-critical code to run as a monitored process alongside the real-time operating system (RTOS) and application software. The secure context might be entrusted to handle encryption, decryption, authentication, certificate management, and other small but vital tasks.

ARM's target applications are smartphones, handheld computers, and any embedded system likely to suffer from the malicious hackers who are expanding their attention beyond PCs. Wireless networking and field-upgradable software are opening new doors for intrusions, yet the typical embedded system has fewer system resources to spare for security measures than a PC has.

For now, at least, ARM isn't promoting TrustZone for smartcards, nor has ARM perceived much interest in TrustZone from smartcard vendors. TrustZone will initially appear in ARM cores too large for the cards, so at this point it's not directly competitive with the SmartMIPS core from MIPS Technologies. (See *MPR 10/1/01-02*, "SmartMIPS for Smart Cards.")

Although ARM says TrustZone's architectural definition is final, the company continues to work with key customers, universities, and consultants to search for vulnerabilities and nail down the last details. ARM plans to implement the extensions in a new version of the ARM11 core, to be announced later this year. The first silicon implementations could appear by late 2004, followed by products in 2005. Going forward, TrustZone will be a standard feature of the ARMv6 architecture and future ARM cores.

Secure to the Core

A key feature of TrustZone is that the extensions are hard wired into the microprocessor core, not implemented in firmware or software. That should provide an extra degree of security over a software-only approach. Operating systems are easily compromised or corrupted—even when they are designed to be secure, which most are not. If nothing else, the sheer volume of code in a modern operating system makes bulletproof verification and security almost impossible. In addition, any security solution requiring third-party vendors to rewrite their operating systems would almost certainly be doomed to failure. A better approach is to chisel the basic security features into the processor's logic and minimize the amount of software that needs to be secure. The fewer doors and windows there are, the fewer the opportunities for break-ins.

However, implementing security in the microprocessor also has disadvantages: it's less flexible than a pure software

solution and it inflates the size of the core. For both reasons, ARM is trying to keep TrustZone as bare bones as possible. ARM estimates TrustZone will require about 15K–20K gates and will increase the area of a basic ARM11 core by about 5%. That's acceptable for applications that need the security features. For applications that don't need security, ARM will probably continue to offer ARM11 cores without TrustZone for the near future.

In essence, TrustZone works by adding a special permission domain that supplements the existing user and privileged modes. Although it's more privileged than the privileged mode in which the RTOS runs (which itself is more privileged than the user mode in which applications run), it's not an exclusive operating mode. Both privileged code and user code can run in TrustZone's secure mode. In this sense, the permission levels of ARMv6 differ from the successive "rings" of privileges familiar to x86 programmers. ARM describes the TrustZone security level as a parallel permission domain, not as new layer in a stack of domains.

To enter TrustZone, the RTOS must invoke the single new TrustZone instruction: secure monitor interrupt (SMI). Only an RTOS running in privileged mode can invoke SMI. Right away, this restriction eliminates many opportunities for mischief, because the processor generates an undefined opcode exception if an application running in user mode tries to execute SMI directly. Instead, the application must request permission to enter TrustZone by calling an API routine in the RTOS. The API call gives the RTOS a chance to perform any checks it deems necessary to ensure the application isn't up to something dodgy. However, the degree of security provided at this juncture is entirely voluntary on the part of the RTOS vendor—minimally, all the RTOS must do is provide a vector to the SMI instruction.

A Bit of Security

When the SMI instruction executes, it switches the processor into the TrustZone domain, also known as secure monitor mode. To make the switch, SMI sets a new security bit, called the S-bit, in a coprocessor 15 (CP15) secure status register, which is part of the core's memory system. The CP15 registers are familiar to ARM programmers; they contain configuration bits for several core functions.

The secure monitor is a small, self-contained, nonreentrant program that is completely independent from the regular RTOS, the secure operating-system kernel, and the secure device drivers that also are required. It is responsible for switching contexts between secure and nonsecure states. ARM will provide reference code for the secure monitor, but it is still determining how to deliver the secure kernel and drivers.

The responsibilities of the secure kernel will depend on the application. At minimum, it must be capable of managing memory securely, but it should duplicate as few functions of the RTOS as possible. Keeping the monitor and secure kernel small will conserve system resources and make it easier for developers to formally prove the system's integrity. ARM envisions a monitor and secure kernel requiring tens of kilobytes—not hundreds.

As long as the TrustZone S-bit remains set, the secure monitor will oversee all operations on the processor. Its first job is to preserve the state of the currently running nonsecure process before switching contexts to the new secure process. The monitor will save the contents of the nonsecure process's registers, most likely in tightly coupled memory (TCM), the on-chip scratchpad RAM that's an optional feature of the ARM11. (See *MPR 6/3/02-01*, "ARM Family Expands at EPF.") The monitor will also save the nonsecure process's configuration settings in an extra bank of CP15 registers. In all, TrustZone adds 350 bits of state information to the processor and requires about 200 clock cycles to switch contexts, although context switching can be faster if the secure state uses only a subset of the processor's registers.

Note that because it is the monitor's responsibility to save and restore states when switching between secure and nonsecure contexts, there is no need to add any context-switching code to the RTOS. The RTOS continues to handle context switches for nonsecure processes only.

The secure monitor doesn't necessarily have to flush the instruction cache, data cache, TCM, or any other memory when switching contexts, because TrustZone divides those structures into secure and nonsecure partitions. The caches, memory-management unit (MMU), and translation lookaside buffer (TLB) have additional tag bits—S-bit tags—to keep track of those partitions. Only a secure process can access a secure memory partition or cache line. Any attempt by a nonsecure process to access the restricted cache lines or memory will generate a cache miss and/or external abort, as if the program had tried to access nonexistent memory.

Although secure partitioning makes parts of the caches off-limits to nonsecure processes, the secure cache lines aren't actually locked (unless the programmer explicitly locks them). New data from a secure or nonsecure process can evict old data from the secure cache lines as part of the cache's normal behavior.

Scratchpad memory isn't organized into lines like a cache, so a system that uses TrustZone must dedicate separate TCMs for secure and nonsecure data. The ARMv6 architecture allows up to four TCMs, and developers can partition a single physical block of on-chip SRAM into multiple logical TCMs. Partitions can vary in size at run time, though it may be undesirable for one process to dynamically allocate resources that another process may need. ARM expects the secure monitor and its scratchpad memory to reside in a secure TCM—partly for its own protection, and partly so the monitor's response latencies are predictable.

By maintaining separate partitions in caches and memory, and by strictly enforcing access privileges for those partitions, TrustZone can eliminate a back door commonly exploited by malicious programs. An untrusted program shouldn't be able to substitute its own instructions or data to seize control of the system or corrupt the output.

Don't Interrupt Me

Another challenge for TrustZone is handling interrupts that may originate from a hostile program. The interrupt could be a ruse for diverting execution to a counterfeit interrupt handler that subverts the secure process it preempted. One solution is to simply block all interrupts while the processor runs in secure mode. Although that might work in a system that doesn't perform time-consuming secure tasks and needn't respond to real-time events, it's unacceptable in a hard real-time system.

ARM thinks most systems will need separate interrupt vector tables for their secure and nonsecure modes, with different interrupt policies for each mode. The secure vector table would reside in secure memory and might point to interrupt handlers that are likewise secure. The nonsecure vector table and handlers would reside in regular memory. Although this solution might duplicate a few handlers and impose some redundancy on the system, it would prevent a malicious program from overwriting the secure vector tables and handlers. An alternative would be to extend security to all the interrupt handlers, but that would require the system to switch into secure mode to handle any interrupt—good for security, but perhaps not so good for interrupt-response latencies.

Even some interrupts that don't seem to need security could benefit from TrustZone. For example, many embedded systems use interrupts to regularly reset a watchdog timer on chip. Without a reset, the timer eventually counts down to zero and triggers a system reboot, which allows the system to automatically recover from a problem that freezes the software. A hostile program could mask the timer interrupt and force the system to reboot over and over again—either as a denial-of-service attack or to open a door for installing a corrupt device driver, application, or even a whole new RTOS. Conversely, a hostile program could paralyze the system without disabling the timer interrupt, thereby preventing a reboot and recovery.

To deflect those kinds of attacks, TrustZone can stop the system from masking the highest-priority interrupts while running nonsecure code. The ARMv6 architecture supports two kinds of interrupts: the regular IRQs and the higher-priority fast interrupts (FIQ). IRQs always jump to a branch instruction in the interrupt-vector table, causing a branch delay when they call the interrupt handler. The FIQ vector is always the last entry in the vector table, which is actually the entry point for the handler, eliminating the need for a branch. Because FIQs enjoy a higher priority than IRQs, it's more important to make them unmaskable by nonsecure processes. Secure processes can use the FIQs. With TrustZone, an FIQ can force the system to switch into secure mode, from which the monitor can jump into a protected vector table and interrupt handler.

When the processor finishes a secure task—whether it's an interrupt-service routine or part of an application—the TrustZone monitor executes the SMI instruction again to clear the S-bit in the CP15 status register. The monitor also restores all other registers to the state of their previous context and switches the processor back into nonsecure mode. All instructions and data stored in secure partitions of the caches, TCMs, and memory remain undisturbed.

TrustZone can also ward off attacks that exploit a spontaneous reboot or infect the system with "sleeper cell" code that waits until the next regular startup. Developers can configure the system to initialize in a secure privileged mode and check for intrusions before loading the RTOS. By authenticating each code module and device driver during the bootstrap process, the system can ensure that it awakens in a secure state.

Armor for One and Armor for All

TrustZone wouldn't be nearly as effective if other parts of the system were left vulnerable, so it can extend protection to off-chip memory and peripherals. The same S-bit tags in the MMU and TLB that restrict access to protected partitions of the TCMs can also fence off regions of off-chip RAM, ROM, and flash memory. Likewise, developers can configure peripherals to check the S-bit before performing security-critical operations. The S-bit appears on the AMBA bus that connects on-chip peripherals, DMA controllers, I/O interfaces, coprocessors, and logic blocks to the processor core. Minor modifications to those devices will ensure that the state of the S-bit determines permission for all transactions. Figure 1 shows how TrustZone can provide systemwide security.

Virtually any component of an embedded system—hardware or software—can take advantage of TrustZone in some fashion, because the extensions are an inherent part of the ARMv6 architecture. No other licensable general-purpose

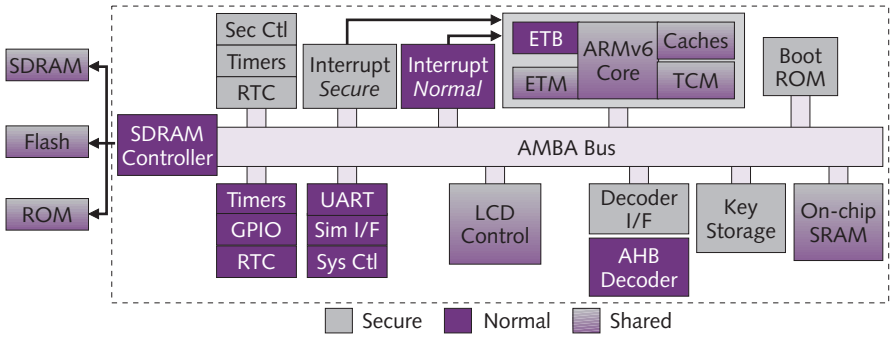

Figure 1. An embedded system can use TrustZone to extend security throughout an ARM-based system-on-chip (SoC) and to portions of off-chip memory. Note that timers, real-time clocks, codecs, and other devices and logic blocks can be modified to check the state of the S-bit before performing security-critical functions. To close another potential back door, chip designers can disable the debug features for secure parts of the chip after development.

Price & Availability

ARM is currently adding TrustZone to a new version of the ARM11 core to be announced later this year. ARM expects the first silicon implementations to appear in late 2004 and the first products in 2005. The company has not announced the way TrustZone will affect its licensing fees and royalties. For more information, see *www.arm.com/armtech/TrustZone*.

microprocessor core offers similar protection at the architectural level.

However, it is possible to add similar extensions to the configurable microprocessor cores licensed by ARC International, MIPS Technologies, and Tensilica, or even to add custom extensions that go well beyond TrustZone. For instance, ARC has instruction extensions for the ARCtangent-A5 processor that accelerate any protocols using DES or 3DES encryption, such as IPSec (Internet Protocol Security), SSL (Secure Sockets Layer), TLS (Transport Layer Security), and encrypted PPP (Point-to-Point Protocol). In addition, ARC recently announced two new security packages, IPShield and ARCprotect. IPShield adds IPSec to ARC's licensable TCP/IP stack for ARCtangent-A5, ARM, MIPS, ColdFire, and PowerPC processors. ARCprotect uses the DES extension and about half a dozen other custom instructions to accelerate IPShield on the ARCtangent-A5.

Usually, custom extensions are unique to the licensee that designs them, not broadly available to all licensees as part of the core architecture. For security extensions, that's both an advantage and a disadvantage. On the plus side, custom extensions can be even more secure than ARM's TrustZone if the designer is secretive. Hacking a custom instruction is almost impossible, because its functions are hidden in hardwired logic. Disassembling the software reveals nothing but an opcode—and an officially undefined opcode at that. The disadvantage of implementing security with custom extensions is that it shifts more design work to licensees and makes supporting the extensions with third-party tools and software more difficult. One way around this disadvantage is for the configurable-microprocessor vendor to develop and license the extensions to customers. For example, ARC's DES extensions are not part of the ARCtangent architecture, but ARC makes them available to all ARCtangent-A5 licensees.

Security extensions like TrustZone that don't accelerate specific algorithms or protocols may cost some performance. Every subroutine, interrupt handler, memory access, and device I/O transaction that needs to remain secure must either trigger a switch into secure monitor mode or check the S-bit to verify that the processor is still running in secure mode. If security is carried to the extreme, the overhead will also be extreme. Developers should carefully evaluate which parts of their system truly need this level of security.

Another reason not to go overboard with TrustZone is the challenge of verifying the system's security, which scales at least as rapidly as the size and complexity of the code. Theoretically, the entire RTOS, and even the application software, could execute in TrustZone's secure mode. However, finding and closing all the potential loopholes in such a design would be almost impossible. ARM is encouraging designers to shield only their most important code.

A system that uses TrustZone may need more on-chip memory for the secure monitor, the secure kernel, and the state information saved during a security switch. In addition to the 15K–20K gates of logic required for the TrustZone extensions, the extra memory will make the chip a little larger and consume more power. It's too early to measure the total effects on die area and power, but we estimate the inflation will be less than 10%—a relatively small price to pay for a secure system. Any other security technology implemented at a higher level than the processor architecture would probably require even more memory, consume more energy (for the additional processing), and be less effective.

Of course, no system can be completely impervious to attack, and ARM is careful to avoid making any such guarantees. TrustZone isn't really a security system; it's an architecture extension that makes it easier for developers to build their own security system. By itself, TrustZone does nothing. Many ARM developers will ignore it altogether. For those who need to design secure embedded systems—a growing percentage of developers—TrustZone provides the basic foundation that higher-level software solutions cannot duplicate. ◇