embeddedworld2017
Exhibition&Conference
... it's a smarter world

Hardware enforced separation in embedded multicore SoCs

Geoffrey Waters
Digital Networking Group
NXP Semiconductor
Austin, TX USA
Geoffrey.waters@nxp.com

***Abstract*—Separation and controlled information flow are the foundations of MILS. This paper describes the methods by which NXP QorIQ processors provide hardware enforced process separation, and root of trust to guarantee execution of safety and security software such as a MILS separation kernel and inter-process communications.**

***Keywords*—MILS; NXP, QorIQ™; Layerscape™, Trust Architecture™; separation kernel; partitioning**

I. Introduction

The cyber-physical world is upon us. By 2020, upwards of 30 billion 'things' will make up the internet of things [1], and many of those things will be able to cause significant harm in the physical world if not properly designed for safety and security. It behooves us to understand what systems engineers operating at the nexus of safety and security for the last couple decades have already learned, and benefit from what these 'early adopters' have helped make ready for mass deployment.

The intent of this paper is to briefly describe the Multiple Independent Levels of Security (MILS) approach to security (and safety), describe the separation kernel concept, then dive into details of hardware which enforces the safety and security policies intended by the separation kernel.

The previously mentioned early adopters of the MILS architecture were largely found in the military & aerospace domain [2]. Mil/aero systems are sophisticated cyber-physical systems, incorporating sensors, actuators, and processors. On the military side, security was a driving requirement, as the networked capabilities of western militaries were (and continue to be) a target for penetration by state level adversaries. Civilian avionics were less concerned with malicious attacks than other types of system faults, and as software grew to millions of lines of code, resilient systems required new methods. Both military and civilian aerospace are very concerned by space, weight, area, and power (SWAP), as well as component obsolescence.

Military requirements for an architecture capable of supporting the US Federal Aviation Administration's DO-178B Software Considerations in Airborne Systems and Equipment Certification [3] and the Common Criteria's EAL 7 security level [4] was the genesis of the MILS architecture. MILS lays out the principles of isolated functions communicating through unidirectional channels. Only the information intended to be shared by one isolated function with another is detectable by the receiving function. Two way communication requires two unidirectional channels, each with policy enforcement. The MILS allowance for isolation via physical or logical means promoted development of separation kernels; infrastructure software that could allow multiple functions to run on the same hardware, with logical isolation and secure communications provided by the separation kernel.

MILS may seem very similar to server virtualization used in cloud computing, and indeed MILS separation kernels leverage virtualization techniques. MILS separation kernels are distinguished from more typical commercial or open source hypervisors by the depth and rigor of their safety and security analysis, and the certifiable guarantees they offer. Whereas server virtualization is oriented toward efficiency, and delivering enough performance on average, MILS (along with proper system analysis) provides assured performance in the worst-case scenario. Server virtualization has historically expected very little communication amongst the virtual machines, and (prior to network function virtualization and service chaining) provided no explicit means of supporting intra-VM communication, while MILS provides explicit methods for secure communications.

While a commercial hypervisor should prevent partition A from accessing partition B's data, a MILS separation kernel guarantees it. MILS separation kernels also offer tighter control over the time domain; the MILS separation kernel guarantees that partition B will be allowed to execute within the required window of time, a commercial hypervisor's scheduling is more of a Quality of Service scheme, which may delay partition B's execution until partition A yields. Explicit secure communication mechanisms and channels, policy configuration, and detection and reaction to policy violations (whether faults or malicious behaviors) are additional distinguishing features of MILS separation kernels.

Just as commercial hypervisors have enabled highly efficient cloud computing, MILS separation kernels, combined with multicore processors, have enabled highly efficient (performance to SWAP) military and aerospace systems. Virtualization with safety and security guarantees also helps long time to market, long lived mil/aero systems, deal with component obsolescence. Partition C, previously executed on a dedicated, but now obsolete processor, can be migrated, legacy operating system and all, onto a suitable multicore processor, while maintaining overall system safety and security guarantees. The quantifiable nature of MILS safety and security allow for compositional certification, where in the example above, the processor + MILS separation kernel (the MILS platform) can be certified as a safe & secure system, and the migration of partition C onto the MILS platform only requires a delta certification of partition C, not a re-examination of all aspects of the hardware and software.

II. Hardware Enforced Partitioning

As previously stated, the MILS architecture supports both physical and logical isolation schemes. Logical separation of partitions on a processor (single core or multicore) requires at a minimum a processor whose instruction set architecture (ISA) and memory management unit (MMU) supports >1 execution level. This allows the separation kernel to execute at the lower level and the partitions to run at the higher level [5]. The typical MILS system runs on a processor with 3 execution levels; 0-hypervisor, 1-supervisor, and 2-user, where the separation kernel executes at level 0.

At boot time, the separation kernel allocates memory regions to each partition (instructions and data), and configures the MMU so that the partition can successfully access its own memory regions (virtual to physical address translation succeeds), but attempted accesses to memory addresses other than those assigned to the partition are blocked, and trigger an alert (trap) to the separation kernel. At the risk of generalizing, a MILS separation kernel would take a dimmer view of an illegal access attempt by a partition than a commercial hypervisor… As the separation kernel schedules processes on cores, it updates the process ID register in the core. This process ID accompanies the addresses generated by the process, and is non-by-passable. A process in partition A can't spoof the MMU into thinking it is a process belonging to partition B.

While a MILS separation kernel can work with as little as a CPU with MMU, if the MILS platform is going to host multiple partitions, and those partitions need access to IO or other specialty hardware, processor characteristics start to dictate suitability. The following sections look at some of the characteristics of processors (or SoCs) particularly suited for MILS platforms.

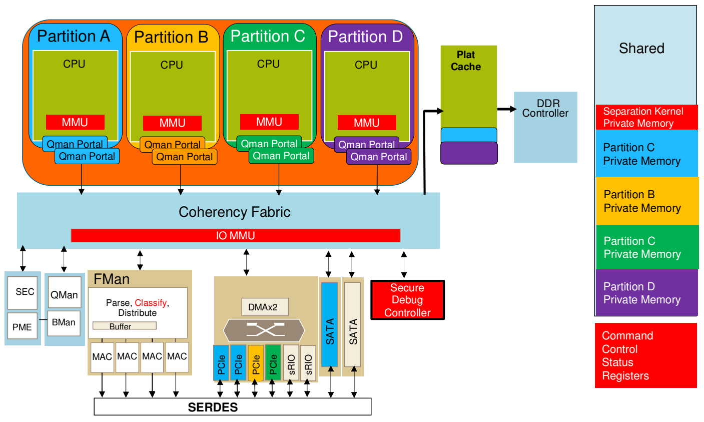

Figure 1: Multi-core QorIQ® Processor

A. *Multiple Cores*

In figure 1, a quad core SoC [6] with integrated peripherals and accelerators is shown. The separation kernel runs across all cores as if it were a symmetric multiprocessing OS, and is responsible for scheduling when the partitions execute. When a processor has multiple CPU cores, meeting temporal scheduling requirements for all partitions becomes easier. Using process affinity, the separation kernel can pin a partition to a specific CPU core or set of cores, so that the partition only competes with the separation kernel (and its integrated communications channels) for CPU cycles [7].

Compared to cloud servers which may host hundreds of partitions/virtual machines, MILS platforms are still embedded systems and will tend to have smaller numbers of partitions. It is often possible for all partitions to be pinned to a dedicated CPU core, or the partitions with the most time critical software to be pinned, while the remainder are scheduled as necessary on the remaining CPUs.

A partition may consist of a guest operating systems with multiple applications, where the OS and applications may further decompose into multiple processes or threads. Affinity techniques can be applied all the way down to the thread, and generally the performance requirements of the most CPU intensive single thread will determine the suitability of a given multicore. Having many cores is nice, but if the individual cores are too weak (from a single threaded processing perspective), the SoC isn't suitable.
.

B. *IO MMU*

An important principle in the MILS architecture is the non-by-passable nature of the logical separation. If partition A, being blocked from directly accessing partition B's private memory, can bypass MMU-based access controls to indirectly access partition B's private memory, logical isolation doesn't truly exist. Indirect access can occur if CPUs aren't the only bus masters in the system, and more often than not, this is the case. IO controllers and various off-load engines incorporate or work with system hardware DMAs (Direct Memory Access) to read and write to system memory. Partition A, knowing it can't directly access partition B's private memory due to the MMU settings of the processor it is running on, programs a DMA engine to read partition B's private memory and write the information to partition A's private memory.

This scenario is totally feasible in systems without an IO MMU. As the name implies, the IO MMU is a MMU which performs virtual to physical address translation and access control, but it does it on behalf of 'IO'; the most commonly occurring non-CPU bus masters. To make it clearer that the IO MMU is operating on more than just IOs, NXP's Power® Architecture based QorIQ devices call this block the Platform MMU (PAMU), and more recent ARM® based QorIQ Layerscape devices adopt ARM's name, the System MMU (SMMU). Like the CPU MMUs, the separation kernel is responsible for configuring the IO MMU (via IO MMU driver) [8] to align non-CPU bus master access permissions with the MILS platform's logical isolation scheme.

C. Non-CPU Bus Masters and Transitive Trust

Logical isolation of a non-CPU bus master (henceforth simply 'DMA') via an IO MMU requires the IO MMU to be able to recognize which DMA is attempting a given memory access. This is accomplished by architecting the SoC interconnect so that in addition to the target address, the DMA sends a unique ID. Depending on the SoC, the DMA's bus master ID may be hardware wired, configured by the separation kernel at boot time, or a combination of hardware default plus additional configuration. If configurable, the separation kernel is responsible for making sure the ID register addresses aren't accessible by the partitions. DMAs are generally fixed function hardware, and have no ability to spoof their bus master ID to the IO MMU.

We've identified the origin of the bus master ID provided by the DMA to the IO MMU, but what is the origin of the address the DMA provides? In most cases, the address comes from a descriptor, programmed by the partition requesting the DMA's services. As previously discussed, partition A may try to put an address belonging to partition B in the DMA descriptor, and the DMA will dutifully output that address toward the IO MMU. The IO MMU will look up the DMA's bus master ID in an access control list, and check if the address falls with the configured allowed address range. In a MILS system, most likely the address doesn't.

For the address to be allowable, it would mean that any partition with the ability to send a descriptor to the DMA could cause a write to the same address in partition B's memory. Even if this address was in a buffer intended for receiving communications from partition A (secure channel via shared memory buffer), the fact that partition C or D could write to the same address allows for denial of service of partition A's communications with partition B. Partition C could inadvertently or intentionally overwrite partition A's data before partition B had a chance to read it. Clearly a DMA with broad access permissions is a threat. This can be managed in software with IO virtualization, or in hardware with transitive trust.

In the IO virtualization scheme, partitions don't have direct access to DMAs. Using a virtual device driver, they create requests (which may match the DMA's descriptor format) and write them to an address which triggers a trap to the separation kernel. The separation kernel analyzes the request against the logical isolation scheme, and if compliant, the separation kernel (which has access to the memory of all partitions) either performs the request in software, or the separation kernel creates the descriptor which the DMA uses to perform the request. All DMAs are exclusively owned by the separation kernel, making it OK for the DMAs to have the same permissions (as enforced by the IO MMU) as the separation kernel has (as enforced by the CPU MMUs).

In a transitive trust scheme, the DMA temporarily takes on the permissions of the partition it is serving. NXP QorIQ devices support transitive trust via a SoC infrastructure called the Datapath Acceleration Architecture (DPAA). The major components of this infrastructure are the Queue Manager (QMan) and the Buffer Manager (BMan). The QMan maintains a list of virtual channels called frame queues, as well as a list in individual requests (frame descriptors) on each frame queue. Producers submit frame descriptors to consumers by enqueuing them onto specific frame queues; consumers dequeue. DMAs have hardwired QMan portals, software produces and consumes via memory mapped software portals into the QMan. Because the QMan software portals are memory mapped, they can be assigned by the separation kernel and access controlled by MMU configuration. The BMan has a similar role and architecture, but manages pools of buffers, rather than queues of frame descriptors. The QMan, BMan, and DMAs are considered to be owned by the separation kernel, but used by the partitions. Normal SW portal and DMA events, which can include successful and unsuccessful completions of requests, are reported back to the partition that sent the request. More serious hardware failures trigger interrupts which are routed to the separation kernel only.

When partition A submits work to a DMA, it enqueues frame descriptors (via the software portal) to a frame queue mapped as a one way communication channel between partition A and that DMA. Based on information received from its hardware portal during the dequeue, the DMA changes its master ID to a value which identifies the DMA as working on behalf of partition A. The DMA's memory access requests are still checked by the IO MMU, but the access control becomes finer grained. Some DMAs can execute frame descriptors from multiple partitions simultaneously, and on a transaction by transaction basis, the IO MMU will perform different address translations/access control operations, based on changing bus master IDs. In earlier versions of DPAA, these changing bus master IDs are called 'Logical IO Device Numbers' or LIODNs. In later versions, the term is 'Isolation Context ID' or ICID.

Not all memory accesses are initiated by partitions. Certain DMAs, such as Ethernet controllers, receive packetized data from the outside world and buffer it to system memory. 'Dumb' Ethernet controllers implement a single bus master ID; accordingly they either have to be directly assigned (para-virtualization) [9] to a single partition (all packets arriving on the interface are known to belong to the partition), or they have to be owned by the separation kernel (IO virtualization). In the latter case, the dumb Ethernet controller buffers all arriving packets into the separation kernel's private memory. The separation kernel is then responsible for classifying the data and copying it to a buffer in the private memory of the partition the data was intended for.

A 'smart' Ethernet controller is capable of classifying arriving packets. Based on determination of partition ownership via classification, the smart controller selects the correct configured bus master IDs and uses it while writing the packet to the partition's private memory.

By the definition above, other IO controllers in QorIQ processors, such as SATA, PCI Express, and UART are 'dumb'. They don't perform classification on arriving data, aren't connected to the QMan/BMan, and always use the same bus master ID. Like the dumb Ethernet controller, each must either be directly assigned to a single partition, or be owned by the separation kernel.

It may be harsh to label PCIe as a 'dumb' IO. PCIe requires special consideration from a partitioning perspective because external devices can read & write system memory via the controller interface. PCIe Address Translation and Mapping Units (ATMUs) perform an IO MMU like function [10], translating the external memory address into a local memory address, which can then be access controlled (and potentially translated again) via the IO MMU. When a QorIQ device is used as PCIe end-point (EP) with Single Root IO Virtualization (SRIOV), the PCIe controller exposes multiple virtual functions (VFs) to the Host. The purpose of this paper isn't to cover SRIOV, but suffice it to say that when a QorIQ processor is used as a PCIe EP with SRIOV, the ATMUs operating on in-bound transactions have different address translations for each SRIOV VF, and can use a different LIODN/ICID depending on the VF.

To summarize, hardware partitioning is achieved through unspoofable identification of the bus master and checking the request against the bus master's permissions in all SoC MMUs/IO MMU. The DPAA and PCIe with SRIOV allow transitive trust, whereby the DMA or IO acts as a proxy for a partition, and identifies the partition on whose behalf it is operating. Initial configuration complexity is higher when exploiting this hardware enforced partitioning, but the performance benefits are substantial (>10x) compared to IO virtualization in the separation kernel.

D. Separation vs Isolation

Separation and Isolation have been used somewhat interchangeably in this paper, but there are differences between these concepts worth clarifying.

Separation is a logical concept, produced by methods such as access control, temporal fair sharing, and dedicated secure unidirectional communication channels. Partition A can't access partition B's private resources (memory ranges), but they share other resources (the bandwidth of the DDR controller); with the result being Partition B may not get access to every resource it wants the instant it wants them. While it is possible to create temporal isolation schemes with enough margin around partition A's resource usage that partition B never perceives a wait, this is rare in practice.

Isolation is both a logical and physical concept. Isolation results in a complete inability of partition A communicate with partition B, to interfere with it, or to even create effects which are detectable by partition B. Partition B has resources which are dedicated to the degree that in all conditions, when partition B wants, it gets. Partition A can't create bus delays, heat, noise, etc that partition B could detect. The existence of detectable effects constitutes a covert channel.

Achieving full isolation on a multi-core SoC may be theoretically possible, but rather inefficient in practice. NXP's QorIQ processors don't claim to support full hardware isolation, and proprietary information regarding shared resource fairness mechanisms and potential sources of interference is shared with customers under non-disclosure agreements.

III. TAMPER-PROOFING

The soundness of the MILS architecture can be undermined by poor implementations. As should be evident from previous sections, the separation kernel, and its proper configuration of MMUs, IO MMU, and other hardware enforced partitioning mechanisms, is the most critical component in a MILS system. It is vital that the separation kernel and all configuration files are protected against accidental corruption or malicious modification throughout the system's life cycle.

Functional safety techniques such as checksums [11] and watchdog timers may be sufficient to detect the types of random faults which could accidentally corrupt the separation kernel and its configuration files, but these techniques are insufficient to deal with malicious modification. Attackers with sufficient access to the system could modify the separation kernel and update the checksum. For this reason, non-by-passable hardware rooted security features are required for system assurance.

NXP QorIQ processors provide non-bypassable hardware rooted security features in the form of the QorIQ Platform's Trust Architecture™. The trust architecture is a set of hardware and software techniques designed to support trusted boot and maintenance of the trusted environment during runtime.

The QorIQ Trust Architecture's secure boot mechanism is the main line of defense against malicious modification. The developer of the MILS system programs fuses in the QorIQ processor, creating a permanent binding between the system and a list of RSA public keys. Rather than a simple checksum, the developer digitally signs the separation kernel, its configuration files, and preferably, the partition images with RSA private keys paired with the RSA public keys. The signed images are programmed into system non-volatile memory and the system is deployed.

Once in the field, the system boots, and the QorIQ processor (executing from a non-modifiable internal bootROM), verifies the public keys provided with the image against the fuse based values. If the keys are valid, the processor checks the digital signature over the separation kernel, config files, and any other signed files, and those verifications pass, the separation kernel begins execution. Failure to verify the digital signature (meaning the contents of non-volatile memory don't match the expected content) is considered a tamper event, and the system will refuse to boot. External physical tamper sensors can also be added to the system and connected to the QorIQ processor, so that if any physical tampers are detected, the system also refuses to boot.

Trust Architecture is a tool kit. Beyond secure boot based tamper detection and inputs for external physical tamper detection, it also supports secure debug, runtime integrity checking, and anti-rollback.

System developers select the secure debug policy by irreversible fuse programming, with options to permanently disable the debug port, or program a challenge/response value.

Run time integrity checking relies on the QorIQ processor's integrated crypto acceleration engine (called the SEC). The SEC performs SHA-256 hashing at multi-Gbps speeds capability to repeatedly hash memory pages holding the separation kernel itself (and anything else the developer deems important). If the hash comparison against the known good value fails, the SEC triggers a security violation warning.

Rollback is an attack strategy in which the attacker attempts to undo security updates by tricking a system into executing an authentic (but old and vulnerable) version of software. NXP QorIQ processors offer monotonic counter and key revocation features to detect 'de-authorized' older revisions of software and block their execution.

Besides preventing malicious code from executing, the Trust Architecture also gives trusted code the ability to create 'cryptographic blobs' using a fuse based one-time programmable master key (OTPMK). While the system is in a Trusted state, software can command the SEC to use the OTPMK to encrypt code, data, certificates, or additional keys, such that if any of these items needs to be decrypted, the developer can be assured that the decryption will only occur if the system has passed secure boot and no tamper events have been detected.

For any type of tamper event, the reaction to the tamper is under the developer's control. Reactions can range from clearing & locking out the master key to instantaneous hard reset of the board, to full bricking of the system.

Newer NXP QorIQ devices using ARM processors also support TrustZone®. TrustZone is an additional CPU execution state designed to support separation of platform resources into 'secure world' and 'non-secure world'. In the context of a MILS system, the separation kernel could run in secure world and the partitions in non-secure world, or the separation kernel and all partitions could run in non-secure world, with highly specialized security functions running in secure world. NXP's Trust Architecture and ARM's TrustZone are complementary technologies, and a full analysis of TrustZone's potential in MILS systems is a topic for future investigation.

IV. Summary

The MILS architecture, originally developed for military avionics, is seeing increasing adoption in other industries dealing with cyber-physical devices. Support for safely and securely combining mixed criticality functions and providing a means for compositional security certification should make MILS the architecture of choice for a variety of industrial applications, including power generation & distribution, factory & service robotics, and intelligent transportation networks. Perhaps in a near return to MILS origins, automotive systems currently distinguished by function (telematics, navigation, infotainment, driver assist, chassis & powertrain ECUs) will consolidate into a new class of 'HADionics' (Highly Automated Drive-ionics) built on MILS separation kernels and processors supporting hardware enforced partitioning.

Acknowledgment

I would like to thank the design team responsible for the QorIQ Trust Architecture, and thank the Horizon2020 project,

certMILS (GA number 731456), for including NXP as third party. The tools and the talent to make a difference are in place.

References

[1] Sam Lucero Sr. Principal Analyst, M2M and IoT, IHS, "IoT platforms: enabling the Internet of Things," March 2016 https://cdn.ihs.com/www/pdf/enabling-IOT.pdf

[2] John Rushby, "A Trusted Computing Base for Embedded Systems," Proceedings 7th DoD/NBS Computer Security Conference, Gaithersburg, Maryland (September 24–26, 1984), 294–311, http://www.csl.sri.com/users/rushby/abstracts/ ncsc84-tcb.

[3] DO-178B was introduced in Jan, 1993 and superseded by DO-178C in July, 2013. These standards are published by the Radio Technical Commission for Aeronautics (RTCA) and are available to RTCA members at http://www.rtca.org/store_product.asp?prodid=803

[4] Common Criteria Evaluation and Validation Scheme (CCEVS), National Information Assurance Partnership (NIAP), http://www.niap-ccevs.org/cc-scheme

[5] Trust and Trustworthy Computing: Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010, Proceedings

Alessandro Acquisti, Sean W. Smith, Ahmad-Reza Sadeghi; Springer Science & Business Media, Jun 9, 2010

[6] Representative of the NXP QorIQ T1040. Reference Manual for the SoC and for the Power Architecture e5500 CPU cores can be accessed at http://www.nxp.com/products/microcontrollers-and-processors/power-architecture-processors/qoriq-platforms/t-series/qoriq-t1040-and-t1020-multicore-communications-processors:T1040?tab=Documentation_Tab

[7] John Fuscoll; The Linux Programmer's Toolbox; Pearson Education, March 2007

[8] Computer Architecture: ISCA 2010 International Workshops A4MMC, AMAS-BT, EAMA, WEED, WIOSCA, Saint-Malo, France, June 19-23, 2010, Revised Selected Papers; Ana Lucia Varbanescu, Anca Molnos, Rob van Nieuwpoort; Springer, Feb 15, 2012

[9] "Understanding Full Virtualization, Paravirtualization, and Hardware Assist"; VMware whitepaper, revision: 20070911; WP-028-PRD-0101; https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

[10] PCI-SIG Address Transalation Services Revision 1.1, January 26, 2009

[11] Robert Hanmer; Patterns for Fault Tolerant Software; John Wiley & Sons, Jul 12, 2013.