Layerscape Architecture

A Look Inside the Next-Generation QorIQ LS Series SoCs

freescale™

The 18th-century inventor Charles Babbage had it right. Although his five-ton "difference engine" had more than 20,000 moving parts, it was remarkable because it was reprogrammable. For all that hardware, it was the software that made it extraordinary.

In modern times, surveys of engineering teams across the world show that they typically spend more time writing software than they do creating hardware. Engineering firms also employ more programmers than engineers, and developers overwhelmingly choose their software environment before selecting the MPU chip(s) it will run on. Their colleagues in marketing know that software provides the most direct path to product differentiation. Original code provides the "value add": the differentiator that sets one product apart in a crowded and fast-moving market.

This is not to say that hardware is not crucial: quite the opposite. The holy grail of hardware design has become to actually make the hardware itself transparent. The hardware needs to be quick to bring up, performance optimized out-of-the-box and software aware.

The Next Steps

If an MPU is just a means to an end—a gateway or a key to unlock an established base of software—then it's important to select the right key. A popular processor architecture, such as the Intel® x86, brings with it an enormous backlog of x86 software, most of it developed for the personal computer market. Likewise, Power Architecture®, ARM® or MIPS architectures come with an implied membership to their respective software "clubs." If the intention is to run PC software, an x86 processor is the mandatory choice. For the embedded or industrial markets, a developer's options are much more flexible.

A software-aware architecture is a platform that enables customers to fully and easily exploit architectural capabilities and features through performance-optimized libraries and (easy to implement) software.

Back when most programs were written in assembly language, the choice of processor also determined the choice of programmers. Developers specialized in different CPU instruction sets and diligently stuck to their preferred ISAs. The code itself was no more transportable than the talent; software written for one processor family was useless on another, except perhaps as an example of how to structure the code the next time. Coding aficionados would defend assembly-language software as efficient and fast, while their business managers usually derided it as slow to develop, overly buggy, specific to one hardware platform, difficult to port and inscrutable to anyone not involved in the original development effort.

Ideally software would be both fast and efficient *and* quick and easy to develop and maintain. No technology has ever quite achieved that ideal, although a shift away from assembly-level programming and toward higher level languages such as C has helped considerably. Nowadays, fewer than 10 percent of embedded developers use assembly language extensively; more than 70 percent use C or C++ almost exclusively. Portable operating systems have helped, too. Linux® in particular is available for almost any processor architecture and any hardware configuration, making it a near-universal platform for embedded developers.

Between portable programming languages and ubiquitous platforms, the industry is closer than ever to achieving the ideal balance of code efficiency (in terms of runtime performance) and development efficiency (in terms of cost, time to market and maintainability). The balance of performance with flexibility and ease of use has become the space race that embedded processor suppliers are working towards. This is where the idea of software-aware architecture becomes crucial. A software-aware architecture is a platform that enables customers to fully and easily exploit architectural capabilities and features through performance-optimized libraries and (easy to implement) software.

Introducing Layerscape Architecture

Layerscape architecture is the underlying system architecture of the next-generation QorIQ LS series SoCs. Designed from the outset to take advantage of the new realities of development, abstraction and efficiency (in both senses of the word), Layerscape architecture was created to "expose" each chip's performance in a way that programmers find very accessible. The architecture extends the current trend toward multicore chip design (both homogenous and heterogeneous) to achieve maximum performance, while also abstracting away enough of the complex hardware to make software development efficient, maintainable, neat, fast and relatively simple. In short, Layerscape architecture balances the performance and efficiency of handwritten assembly-language code with the ease of use of high-level languages and modern code maintainability.

Layerscape architecture can be viewed as an evolution of the Data Path Acceleration Architecture (DPAA) found in the QorIQ P and T series—an infrastructure supporting the sharing of networking interfaces and accelerators by multiple CPU cores, and the accelerators themselves.

Layerscape Architecture Block Diagram

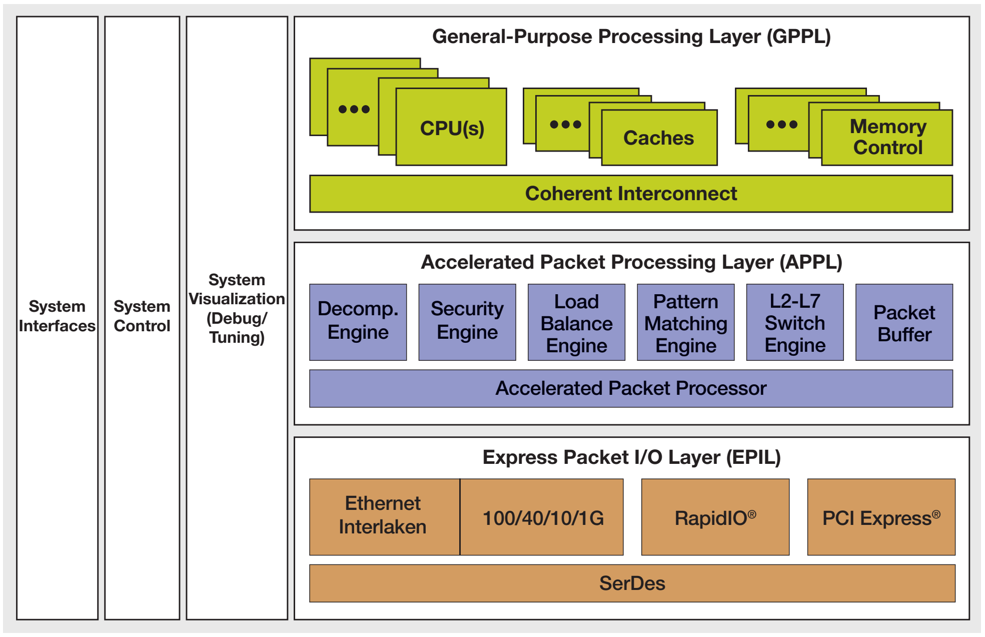

Figure 1. All LS series chips are logically, though not always physically, partitioned into three layers. The topmost layer can include any type of processor, such as those based on Power Architecture or ARM technologies. Lower layers are accessed through buffers, queues and APIs that abstract away the details of their implementation.

Layerscape architecture extends the DPAA, in that separate layers of the ISO networking model are accelerated separately and appropriately, depending on the specific chip. Some chips may handle compression (for example) in software, while others have dedicated hardware accelerators. Either way, the function is transparent to programmers, making it straightforward to switch from one chip implementation to another without altering any code. Structured programming interfaces encapsulate the compression (in this example), so that neither the code that calls it, nor is called by it, need know how the compression is actually implemented. Here again, abstraction preserves efficiency and performance, as well as the sanity of the developers.

In the QorIQ LS series, each communications processor is logically organized into three layers, as shown in figure 1. The general-purpose processing layer (GPPL), accelerated packet processing layer (APPL) and express packet input/output layer (EPIL) represent roughly the high, middle and low layers respectively of the standard ISO model. Whether or not the chip is physically partitioned in this way is irrelevant; the programmer perceives it this way, regardless of how an individual chip might be provisioned.

At the lowest level, the express packet I/O layer (figure 1, in brown) provides true deterministic wire-rate performance between all network interfaces supporting L2+ switching capabilities and contains the chip's interfaces for network datagrams, such as Ethernet, Interlaken, Serial RapidIO®, HiGig and PCI Express®. Important but unrelated interfaces, such as USB or SATA, would not be part of this interface layer but would instead be part of the chip's "system interface" block, as seen on the left of the diagram (figure 1). Although, strictly speaking, PCI Express isn't a network interface, it's often used as such between blades in a rack, hence its inclusion here.

The middle layer (figure 1, highlighted in blue) contains the chip's packet-processing elements, whether those be hardwired accelerators, programmable engines or some combination of the two. The APPL provides customer-defined, autonomous and value-add capabilities through a traditional sequential, synchronous, run to completion model and is fully programmable through embedded C-based structured programming. Again, these elements will communicate with the general-purpose processors over well-defined interfaces that abstract away the details of their (and the processors') implementation in such a way that preserves valuable developer code.

The general-purpose processors (figure 1, highlighted in green) are, obviously, general purpose in nature and are free for users/developers to use for their operating system(s), applications, high-level code and other value-added features. In keeping with Layerscape architecture's values of abstraction, efficiency and hardware independence, this layer can support both Power Architecture and ARM cores. Of note is the fact that Power Architecture technology generally uses big-endian byte ordering, while ARM technology is normally little-endian, yet Layerscape architecture happily supports both.

Clearly, the modular hardware architecture lends itself to many different chip configurations and is a single architecture with consistent software across the platform. The modular and flexible hardware framework includes independently scalable layers to maximize performance and power efficiency across the QorIQ portfolio. As mentioned above, those configurations can even include general-purpose processors from various instruction set families, thus allowing the developer to leverage different code bases. Layerscape architecture's modularity also allows performance up- and downgrades—sometimes within the same physical socket—while preserving customer code.

A very basic chip implementation, for example, might include only the low-level interfaces (Ethernet, for instance) and the top-level general-purpose processor (i.e., ARM or Power Architecture cores), with no intermediate accelerators in between. In that case, the EPIL layer would perform packet parsing, classification and distribution to frame queues (not shown). A general-purpose CPU (or perhaps multiple CPUs) would then consume those packets from the queues.

Layerscape architecture combines the extreme performance of today's most capable communications processors with the familiar, modular, high-level programming models used worldwide.

Expanding this concept across multiple Ethernet ports, the same chip could act as a layer 2 switch by taking advantage of Layerscape architecture's built-in "link aggregation" feature. A more generously provisioned chip might include hardware in the middle APPL for fine-grained packet classification, IPsec, SSL, LRO/TSO and other advanced inline offloads. Similarly, the low-level EPIL might identify certain packet types and reroute them directly to relevant accelerators in the middle APPL, bypassing the general-purpose processors entirely.

The foundation of the solution is the software that allows the programmer to quickly and easily harness the power of the architecture. The solution begins with optimized networking libraries for hardware accelerated functions such as IPSec, deep packet inspection, IP forwarding, NAT/FW, etc., allowing the embedded developer to focus on value-added software and not performance tuning. Well defined datapath and control APIs are standard for many networking applications and are easily extendable for custom applications using an imperative C programming model. Additionally, a software framework providing standard services such as debug and profiling, resource management, virtualization and initialization are provided to ensure ease of use. Finally, reference implementations for key applications such as software-defined networking, wireless transport/backhaul, TCP termination and routing will be provided to not only reduce your R&D investment but accelerate time to market.

Summary

Layerscape architecture combines the extreme performance of today's most capable communications processors with the familiar, modular, high-level programming models used worldwide. It makes advanced communications engines accessible without requiring an advanced degree in hardware engineering. More importantly, it doesn't require relearning the details of each chip implementation as one generation of QorIQ LS series devices gives way to its successor. The bounded and well-defined programming model survives from chip to chip, generation to generation, building upon the work of developers rather than discarding it as hardware implementations change. In short, Layerscape architecture preserves the most important and most valuable aspect of any development team: its differentiating software. Once again, the right hardware proves to be the key to unlocking the right software.

For more information, visit freescale.com/QorIQ

Document Number: LAYARCHTECHOVWP REV 0