Freescale Semiconductor, Inc.

C-PORT™
A Motorola Company

Network Processor Programming Models: The Key to Achieving Faster Time-to-Market and Extending Product Life

By: David Husak
C-Port Co-founder and
Chief Technical Officer

and

Robert Gohn
C-Port Vice President
Marketing

White Paper

The design of the network equipment powering the Internet revolution has undergone profound changes over the last decade. Today, with network equipment vendors racing to provide the new converged voice/video/data communications infrastructure, designers require both speed and flexibility to deliver within the highest time-to-market pressures the industry has ever seen.

Powerful new network processors are challenging traditional network device design methodologies by enabling software implementations of virtually all key communications functions at hardware speeds. Key to this revolution is the programming models that enable designers to implement the communications processing tasks on these processors. This paper explores these models, and their effects on delivering on the promise of a new and better network device design process.

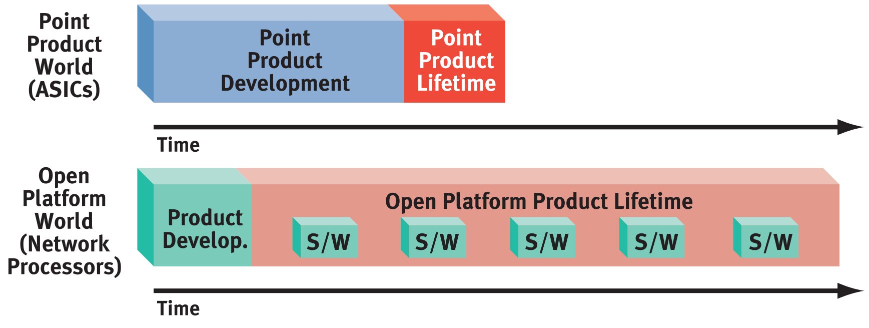
Industry Imperatives: Time-to-Market And Time-in-Market

Just as the Internet revolution is forever changing the face of public communication networks, the way products that make up these networks are designed is also changing. Network equipment developers have consistently faced a difficult trade-off: performance requirements demand hardware implementations of data forwarding functions, while new features, such as advanced Quality of Service (QoS), require flexibility that only software can deliver. Designers have been forced to revisit the fundamental hardware/ software trade-off with each new product (or even line card) they develop, sacrificing software reuse between product lines and product generations along the way. The result has been longer time-to-market, higher development costs, and shorter product lifetimes. Companies trying to compete in "internet time" can no longer afford this type of product development.

The network processor, a new type of semiconductor device, is changing the dynamics of the speed versus flexibility trade-off by enabling virtually all communications functions to be software programmable without sacrificing "hardware" speeds. These processors eliminate the high-risk, long development cycles of custom hardware by enabling advanced product features to be delivered completely in software, even long after initial product introduction. This allows network equipment vendors to concentrate precious development resources on delivering advanced services to their customers, rather than just the latest "feeds and speeds".

The best network processors form the foundation of a "communications platform" that contains the key elements required to radically transform the network device design process. For example, Motorola's Smart Networks Platform combines advanced network processor technology, "standard" programming interfaces, communications software components (from C-Port and Motorola alliances) and a comprehensive development environment. This enables network equipment vendors to quickly bring to market a wide array of different products based on the same hardware and software architecture. The result is significantly faster time-to-market for new products, and dramatically longer time-in-market (through the use of software upgrades to deliver new, advanced services that extend the product life cycle). See Figure 1.

Figure 1 ASICs versus Network Processors Product Life Cycles


Point Product World (ASICs)
Point Product Development
Point Product Lifetime
Time
Open Platform World (Network Processors)
Product Develop.
Open Platform Product Lifetime
S/W
S/W
S/W
S/W
S/W
Time

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

But not all network processor architectures can support the platform model. The communications platform requires more than a reasonable "merchant silicon" point-product alternative to ASIC design. With so much of the platform value riding on the programmability of the devices, the network processor programming model is a key metric by which these solutions must be evaluated.

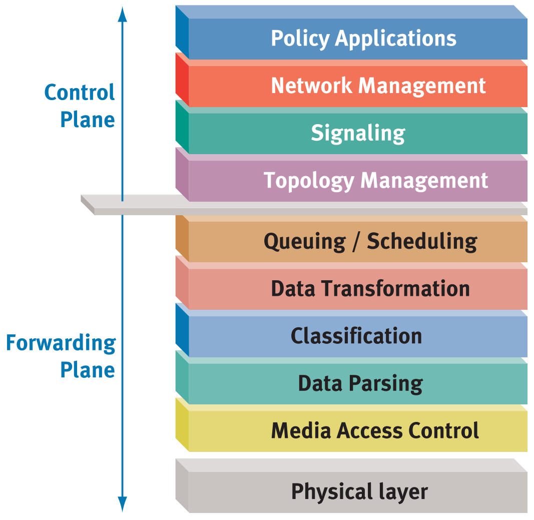
The Nature of Communications Processing Tasks

To evaluate network processor programming models, the nature of the tasks to be programmed must be understood. There are two broad categories of communications tasks (see Figure 2):

- **Forwarding Plane tasks** — Consisting of operations on forwarding path communications data that occur in real-time. These constitute the core device operations, and hence are performance critical. In a switch or router, these are the functions that receive, process, and transmit packets into and out of the device.
- **Control Plane tasks** — Consisting of less time-critical control and management functions that determine general device operation. In a switch or router, these functions control routing table maintenance, port states, and higher-level management.

In traditional designs, the forwarding plane functions are divided between fixed-function hardware (usually custom ASICs) and software running on a general-purpose CPU. Control plane functions are implemented in software either on the same CPU or another, dedicated "host" CPU.

Figure 2 Communications Processing Tasks



Network processors are specifically designed to bring programmability to the forwarding plane functions (layer 2 and higher of the ISO model) required by the LAN and WAN devices that make up today's networks. These forwarding functions include:

- **Media access control** — Implementation of low-layer protocols, such as Ethernet, SONET framing, ATM cell processing, and so on. These protocols define how the data is represented on the communications channel, and the rules governing how that channel is accessed. Paradoxically, this is the area of the greatest standardization among network devices (due to standards-based protocol definitions), and also the area of greatest diversity (due to the wide and ever growing variety of protocols). These include: Ethernet (with three different flavors at 10Mbps, 100Mbps and 1000Mbps), SONET supporting both data packets and ATM cells at a wide range of standard rates (OC-3, OC-12 OC-48, and so on), legacy T/E-carrier interfaces from the existing public voice infrastructure, and a variety of emerging optical interfaces all must coexist and interact.
- **Data parsing** — Parsing cell or packet headers containing addresses, protocol information, and so on. In the past, parsing functions were fixed based on the type of device being constructed (for example, LAN bridges, by definition, only needed to look at the layer 2 Ethernet header). Today, switching devices need the flexibility to gain access to and examine a wide variety of information at all layers of the ISO model — in real time and on a conditional packet-by-packet basis.
- **Classification** — Identifying a packet or cell against a set of criteria defined at layers 2, 3, 4, or higher of the ISO model. Once data is parsed, it must be classified in order to determine the required action. Actions might include such basic functions as a filtering/forwarding decision, as well as advanced QoS and accounting functions based on a specific end-to-end traffic flow. This is an area of rapidly changing requirements.
- **Data transformation** — Modification or translation of data within or between protocols. The variety of low-layer transport protocols is matched only by the diversity of protocol combinations and services. Transformation requirements can range from address translation within a given protocol (such as IP) to full protocol encapsulation or conversion (such as between IP and ATM).
- **Traffic management** — Including the queuing, policing, and scheduling of data traffic through the device according to defined QoS parameters, based on the results of classification and established policies. These functions are key to supporting convergence of voice, video, and data in next-generation networks.

Today, each of these functions presents the challenge of a wide diversity of possible implementations, rapid evolution based on continuing innovation, strong interdependencies between functions, and a need for interworking between the diverse protocols. Delivering programmability and integration of these functions represents a major evolution in network device design.

Network Processor Programming Model Choices

The computing world has always debated about what is the best processor hardware architectures: CISC versus RISC, single CPU versus multi-CPU, coprocessors versus faster clocks, and so on. However, it is the *software* that determines the success of computing platforms, both in terms of performance and programming ease. The limited success of symmetric, parallel computing architectures proved that raw computing power was not the decisive factor, but rather how that power could be harnessed by *software*. The same is true for network processors — the decisive factor is how the programming model serves the platform requirements of fast, simple, and flexible programmability.

There are two primary metrics for evaluating network processor programming models. The first is the level of programmability offered, both in terms of which functions can be programmed (see Figure 2), as well as the extent that these various functions can be programmed. While the physical space, cost, and power benefits of high functional integration into a single processor is well understood, there is a forgotten benefit to the programming model. Processor architectures that assume a "bag of parts" approach provide programmability for a subset of the forwarding plane functions, limiting the ability of programmers to effectively deal with the diversity within each level and the often complex interactions between them. Likewise, providing appropriate programmability within each level is crucial to accommodating these interactions. Hence a fully integrated, fully programmable network processor architecture is a major prerequisite for an effective programming model.

The second, and most important metric, is the actual programming method for the processor. Perhaps the largest struggle in traditional network system design with ASICs has been a "hardware first" architectural mentality, with software engineers designing around a less-than-ideal hardware/software partitioning. Network processor programming models need to turn this around, providing a hardware processor platform that serves the requirements of the software functions and, in the end, the software designers themselves. The key criteria is a simple programming paradigm, using well known methods, without sacrificing product performance.

The network processors available today fall into three broad categories of programming methods, with a spectrum of capabilities within each. These categories are discussed below.

Microcode Engine Programming

These devices implement virtually all the forwarding plane functions in custom designed, low-level microcode machines. All tasks including data parsing, search algorithms, data transformation, queuing, and scheduling algorithms must be specifically programmed by the designer. These machines maximize performance through multi-threading, which generally requires the microcode writer to consider everything from memory access times to thread interactions when optimizing each function.

Also, these architectures implement multiple instances (typically 6 to12) of these machines in parallel, with fixed hardware schedulers assigning incoming data to a given machine based on availability. This is similar to traditional symmetric multiprocessing computing models, which places additional constraints on the microcode writer to assure proper interactions and ordering of forwarded data.

Microcode's strength lies in the efficiency of the code once it is written. The code can be compact and fast. However, the downside of programming in low level microcode, comprehending everything from memory access latencies to multiprocessing dependencies, is the lack of portability of these code designs to other products based on same processor and to new, faster versions or new generations of processors. This code tends to be "one time use", which when combined with the inherent difficulty of writing microcode, might make these processors suitable for point product designs, but seriously compromises the value of "software programmability" for an overall communications platform.

4GL Programming

Another programming model focuses on leveraging proprietary search and pattern-matching algorithms to the communications processing task, specifically for parsing and classification. A number of these algorithms use custom "fourth generation languages" (4GLs) to describe the parsing and pattern-matching requirements for a number of applications. These 4GLs provide a concise method of "programming" the classification function, and processors that implement these algorithms provide a partial solution to this piece of the communications processing task.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The algorithms implemented by these processors typically trade-off memory size for search speed, which may or may not be an issue for the system design. There are, however, larger impacts on the programming model against the two main criteria outlined above. First, these processors focus almost exclusively on the parsing and classification tasks, providing only one piece of a "bag of parts" solution. The designer must either build the required external hardware (and associated software) around this part, or, if available, use other piece parts provided by the processor vendor (sometimes configurable with microcode as described above). In either case, the programming domain is disjoint, compromising what functions are actually programmable and the depth of that programmability.

Even if the other functions are ignored, using a proprietary description language for the classification requires new skills and tools, not just for the coding tasks but for debug, analysis, and maintenance. Good tools can mitigate some of this cost, but the inconsistency between the other forwarding plane functions and the control plane functions will remain.

Standard Language Programming

The "standard language" programming model leverages existing languages (such as C and C++ with their inherent benefits such as readily available skilled programmers and industry standard programming tools), usually combined with special coprocessors, to implement the various communications processing tasks. These use multiple embedded RISC cores as a key processing element to support the execution of standard C/C++ programs implementing the desired behavior.

Note that the use of RISC cores in a network processor does not automatically mean that the processor was designed to support a higher-level programming language paradigm. Many "RISC-based" network processors implement proprietary instruction sets (or proprietary extensions), which, while expedient from a hardware design perspective, force programmers to write all or significant portions of their code in RISC assembly language. Similarly, the processing capacity may not be adequate to support reasonable implementations in a higher-level language. Thus, programming these processors can be just as complex as writing in low-level microcode.

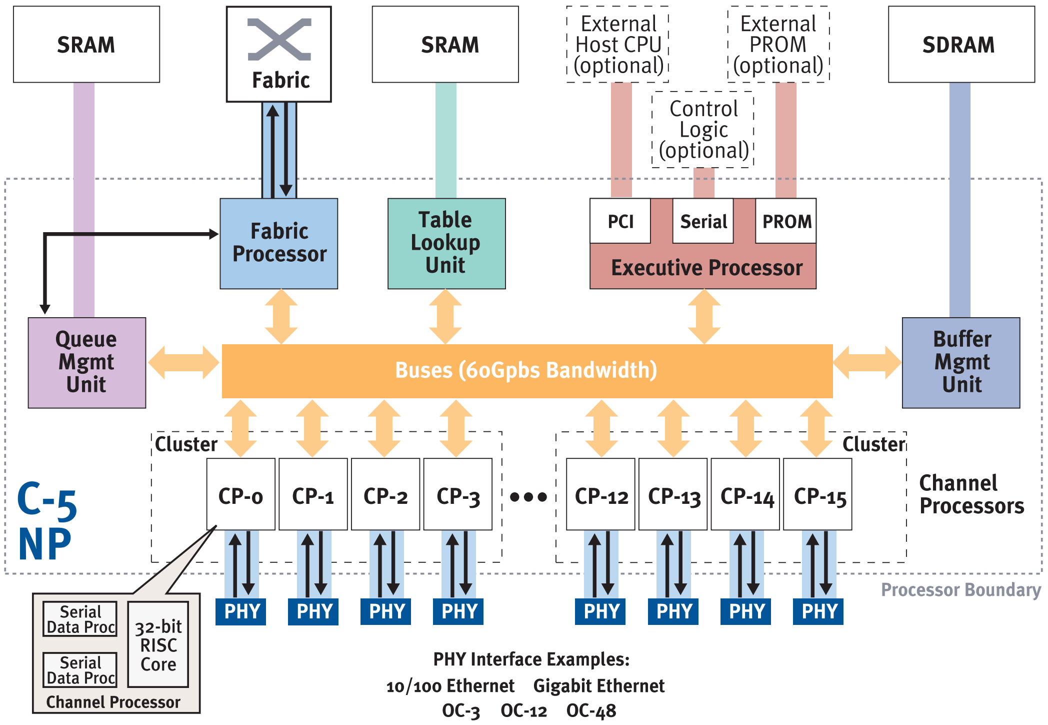
To effectively support the requirements of a communications platform, the programming model must support the ability to write effective programs in a higher-level standard language. The means the RISC cores need to have enough horsepower within a rich coprocessing architecture to support an API abstraction layer that insulates the operating code from low level chip implementation details without sacrificing performance. This is the key to providing a simple programming model environment and extending the life of the software.

The implementation of the coprocessing architecture is critical, as the coprocessors must off-load the RISC processor from the communications tasks that are notoriously poor in standard CPUs (such as the bit manipulation typically required in parsing and data transformation tasks).

The Foundation: A Network Processor Designed for Communications Tasks

C-Port's C-5 Network Processor (NP) is an example of a network processor designed from the ground up to provide a simple and robust programming model. The C-5 NP provides complete programmability for each of the forwarding plane tasks using standard C/C++ programming, enabling universal applications in a wide variety of network devices. The C-5 NP combines multiple RISC cores, specialized coprocessors, and microcode engines within a single integrated circuit to offer a full range of programmability at high performance. Figure 3 shows a block diagram of the C-5 NP.

Figure 3 C-5 NP Software-optimized Architecture



Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Channel Processors: Flexible Building Blocks

The fundamental building blocks of the C-5 NP are the 16 embedded Channel Processors (CPs). Each CP consists of a dedicated RISC CPU and dual Serial Data Processors (SDPs). The CP structure combines the best attributes of specialized configurable state-machine architectures with a fully programmable RISC core. CPs can be assigned to physical interfaces, aggregated together to support higher-bandwidth I/O streams, or assigned internally as a dedicated internal coprocessor.

The SDPs handle data encoding/decoding, framing, formatting, parsing, error checking (CRCs), and data movement. The SDPs also control programmable external pin logic, allowing them to implement virtually any layer 1 interface including connection to T/E-Carrier framers, 10/100 Ethernet PHY (RMII), Gigabit Ethernet PHY (GMII or TBI), OC-3 PHY, OC-12 PHY, and OC-48 framers/PHY. At layer 2, the SDPs can be independently configured to support Ethernet, PoS, HDLC streams, ATM, Frame Relay, FibreChannel, or virtually any format including various encapsulations such as MPLS. The programmability of the SDPs support the diversity of media access control interfaces, as well as first-order parsing requirements, and can support the "mix-and-match" requirements of different implementations on a port-by-port basis. This efficiently supports the needs of various interworking applications.

The SDPs are programmed in microcode, which is provided by C-Port for the vast majority of applications (all flavors of Ethernet, IP and ATM over SONET, T/E carrier serial data streams, and so on). All the tools necessary for equipment vendors to program the SDPs (including assembler and simulator support) are available. Support for MAC level diversity is available without any user coding.

The CP's RISC core, programmed in C or C++, is available to focus on higher-level tasks such as final switching / forwarding decision making, scheduling, statistics gathering, or other tasks required for higher-level services. The RISC core in each CP operates at the core clock rate of the C-5 NP, has dedicated internal instruction and data memory, and implements an industry standard instruction subset, avoiding the issues associated with proprietary instructions. With the SDPs off-loading the "bit level" tasks from the RISC core, the capacity of the RISC machine can be dedicated to the tasks that benefit the most from high-level language implementations.

Dedicated Coprocessors

The C-5 NP also provides five coprocessors optimized for common tasks and used by the CPs. These coprocessors handle shared tasks including table lookup, queue management, buffer management, fabric interfacing, and supervisory processing. Each unit is highly configurable and offers performance and capabilities that, if packaged as stand-alone devices, would be considered best-in-class communications components. For example, the Table Lookup Unit (TLU) enables a wide range of traffic classification functions and supports multiple, different search algorithms.

Designed for High-level Programming

The CPs, supported by the coprocessors, provide the fundamental building blocks from which multiple applications can be supported through high-level programming. For example, the CPs can take on different personalities to support ATM, Ethernet/IP, PPP/IP, Frame Relay, Channelized HDLC, or even proprietary protocols through a combination of microcode in the SDPs and C/C++ code running on the RISC core. The data paths through the CPs can be configured for external connection (to PHYs) or looped back internally, for use as an applications "coprocessor".

Although there are 16 CPs per C-5 NP, each CP is independently programmable, avoiding the limitations typical of traditional symmetric multiprocessor designs. With the flexibility provided by the CP architectures, it is a straight forward task to write software for the CP to perform a given function.

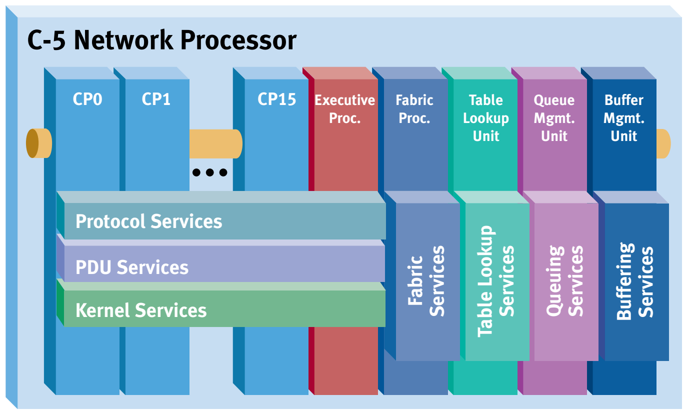
Making Programming More Simple Through a "Communications" API

Hardware flexibility is usually accompanied with complexity driven by the number of possible functional permutations. By adapting the concept of standard Application Programming Interfaces (APIs) to communications processing, this complexity can be put at the service of the programmer. The C-5 NP supports C-Ware Application Programming Interfaces (APIs), a set of open, efficient interfaces that abstract common functions from the underlying hardware. See Figure 4.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Figure 4 C-Ware APIs



For programmers accustomed to tweaking hundreds of lines of assembly code to squeeze out the last bit of performance from a CPU, the concept of using an API in forwarding plane code would appear odd. However, the C-5 NP computing power (over 3,000 MIPS total) was sized from the beginning to accommodate any overhead imposed by an API. This, combined with standard C/C++ programming, is the key to delivering on a simple programming model.

In an effort to leverage the power of this concept throughout the industry, a group of network processor, software, and equipment vendors (with C-Port, IBM, and Lucent as charter members) initiated the Common Programming Interface (CPIX) Forum (www.cpixforum.org). By defining a common framework and API, network processor vendors and communications software vendors can offer more portable and flexible solutions for network equipment designers.

Programming Environment Requirements

The use of a true communications platform in network device design changes the typical design process. A much larger percentage of the intellectual property of a product is delivered in software, hence the network processor development tools environment is critical to project success. In addition to the basic programming model, other factors influence the speed at which products can be brought to market.

These factors include:

- **Software reference design availability** — Most network processor vendors provide examples of forwarding plane software for some number of functions. The extent, quality, and breadth of these applications (as well as available implementations from software partners) can help make or break a project schedule.
- **Robust simulation environment** — Most network processor vendors provide extensive simulation environments that allow completion of forwarding plane code development and performance characterization before hardware integration. A key differentiator is the speed and accuracy of the network processor simulation. Those based on a full software implementation can be as accurate as a hardware model (for example, based on Verilog/VHDL models), but orders of magnitude faster, allowing more simulation bandwidth.
- **Development system availability** — A hardware development system, offering the ability to execute software on the “real” network processor, is also generally available from most vendors. While not a replacement for a good simulator (a simulator can always be better instrumented than real hardware), it is invaluable for starting final integration in advance of prototypes. A system that can be assembled to closely match the target system configuration (types of physical interfaces, and so on) is a great asset.
- **Other software tools** — Software tools, such as compilers, debuggers, performance analyzers, and so on are also key elements of the software development environment. Seamless integration of these tools across both the simulation and hardware development platforms is an often overlooked, but important, aspect of accelerating time-to-market.
- **Host processor integration** — As described earlier, the control plane functions are supported in a traditional embedded CPU. The hardware integration of this processor with the network processor is straight forward, but the software integration requires some considerable thought. Hence a software and hardware development environment that comprehends the host processor, including drivers for the leading real-time operating systems, host-level APIs, and some number of fully integrated applications, should be a key consideration.

For example, C-Port provides a complete communications development environment, consisting of a full software toolset (including simulator), and a development system. The development system consists of network processor modules, physical interface modules (for Ethernet, Gigabit Ethernet, OC-3, OC-12, and so on), and a host processor module based on a PowerPC CPU running the VxWorks RTOS. The vast majority of an application can be integrated and tested prior to integration with the target product hardware design, significantly reducing the time and risks of the product integration phase.

Freescale Semiconductor, Inc.

Conclusion

Network processors offer a significant opportunity to improve the architecture, design, and maintenance of today's networking devices. The opportunity, however, extends beyond the standard benefits of off-the-shelf merchant silicon. Processors that form the foundation of complete communications platforms, based on a simple programming model, promise to radically improve the way networking technology is brought to market. This adds up to better product features, faster time-to-market, and better reliability for network equipment vendors and their customers.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

C-5, C-Port, C-Ware, and the C-Port logo are all trademarks of C-Port Corporation.

© 1999, 2000, 2001 C-Port Corporation
PM00WP100

C-Port Corporation
120 Water Street
North Andover, MA 01845
978-773-2300 TEL
978-773-2301 FAX

www.cportcorp.com
www.mot-sps.com

C-PORT™
A Motorola Company

For More Information On This Product,
Go to: www.freescale.com