NETRONOME

WHITE PAPER

Netronome NFP: Theory of Operation

TO ACHIEVE PERFORMANCE GOALS, A MULTI-CORE PROCESSOR NEEDS AN EFFICIENT DATA MOVEMENT ARCHITECTURE.

CONTENTS

1. INTRODUCTION

Netronome Agilio SmartNICs are based on Network Flow Processors (NFPs). The NFP architecture improves throughput and reduces power with specialized programmable cores, accelerators and a data movement architecture based on relaxed coherence. The programmable Flow Processing Cores (FPCs) enable a rapid rate of innovation in SmartNIC features and functionality. Network packet processing in network interface cards (NICs) is a data-intensive task that needs to meet stringent power and performance budgets. At a 50Gb/s line rate, a NIC has less than 13ns to process a packet, under a power budget of 25W. Vendors typically design and offer fixed-function devices to meet these requirements. In contrast, as workloads evolve, cloud/network operators can upgrade the functionality of the SmartNIC post-deployment by changing the firmware on the NFP.

To achieve performance goals, a multi-core processor also needs an efficient data movement architecture. An inefficient design can expend significant on-chip area and power to provide memory coherence at a low latency across all the programmable cores. The cost of consistency grows with the data rate, the target latency for consistency and the physical area over which consistency has to be achieved. In network processing and many other data-intensive applications, not all data processed by a system need be consistent at the same rate and with the same latency. Coherence for control data that specifies how packets are to be processed is more important than coherence for packet data. Data consistency or coherence requirements can be relaxed for a significant fraction of system data.

The NFP architecture can meet the functional requirements for a wide range of applications, such as virtual switching, routing, tunneling, BPF/XDP-based filtering, DDoS, etc. at exceptionally high-performance line rate needed in next-generation data centers. Section 2 reviews the architecture of the NFP. Section 3 discusses the programming model.

2. ARCHITECTURE OVERVIEW

Figure 1 shows the significant Logic Blocks in an NFP architecture. Data enters the NFP from the network ports or hosts through the Enhanced Network Interface or Enhanced Host Interface Logic Blocks respectively and is pre-processed at the interfaces so it can be directed to the right location and processed more efficiently. Packet processing in the Enhanced Network Interface occurs using Design Blocks such as the Packet Classifier and Packet Characterizer. The Enhanced Host Interface Logic Block includes PCIe, DMA and I/O virtualization-related Design Blocks. Most packet processing occurs at the Network Processing Logic Block instances composed of multiple FPCs and local memories. Packets and flow state are stored in internal and external memories, connected through Internal and External Memory Controller Logic Blocks. In the next subsections, the discussion will focus on some of the Logic Blocks used in the NFP.

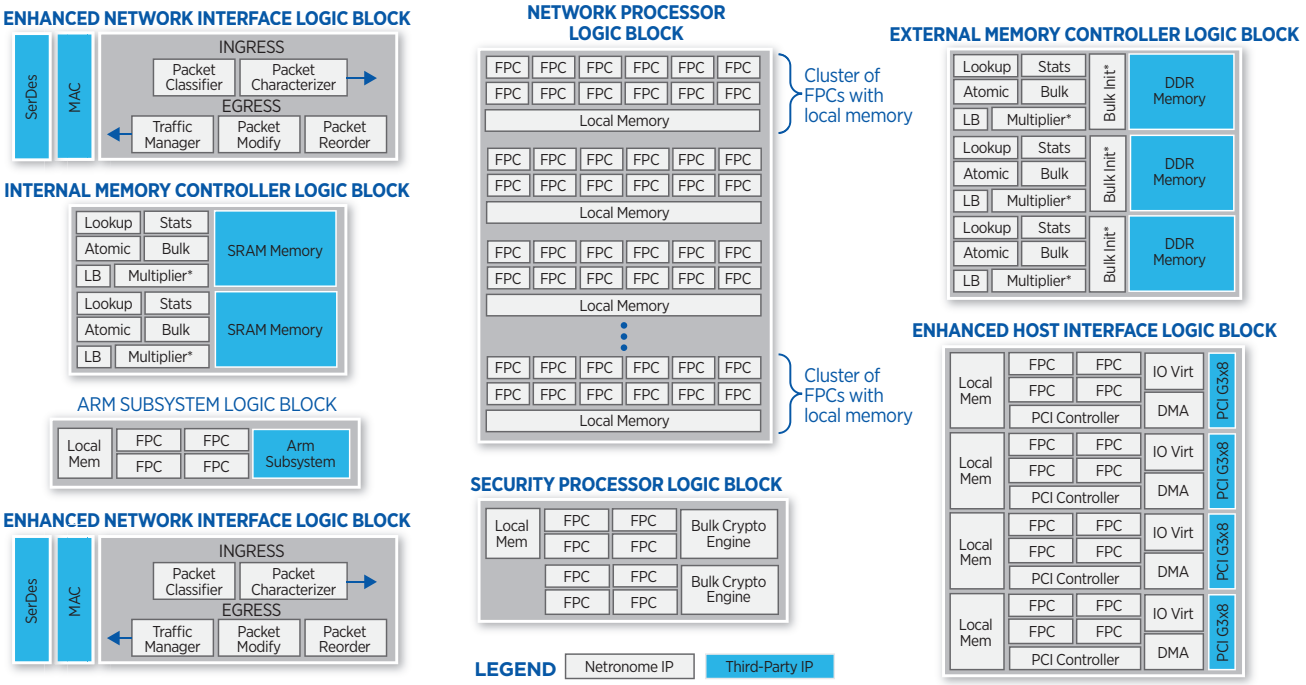

Figure 1. Logic Blocks in the NFP architecture

Network Processor

Network Processor Logic Block instances, as shown in Figure 1, consist of clusters of multiple FPCs and local memories. The FPCs in a Network Processor instance can share instruction memory and have low-latency access to other memory in the same cluster. The elements of the cluster within a Network Processor Block instance are tied together using the Island Master Bridge (IMB) bus. Figure 2 focuses on the details of this block to show how the bus interfaces enable each FPC to communicate with other FPCs within the Logic Block or to other Logic Blocks. The figure also includes (in the bottom-right corner) an abstracted view of the Network Processor Logic Block with six Distributed Switch Fabric (DSF) interfaces.

A FPC is the principal data processing unit in the NFP. Most Logic Blocks have one or more FPCs as shown in Figure 1. A single FPC provides programmers with storage of up to 32K

instructions and 40-bit address space for programmable packet processing. Devices in the NFP-6000 family contain up to 120 FPCs. Given the classical disparity between FPC clock rate and external memory accesses, a single thread of execution would very frequently stall a processing unit, simply waiting for memory operations to complete. An FPC provides support for software-controlled, multi-threaded operation. Having multiple threads (such as the 32 available in the FPC) allows for "thread interleaving" where there is, very often, at least one thread ready for execution. Developers use a run-to-completion model in which a "sea of worker" FPCs service a work queue to process packets. This is covered in greater detail in Section 3.

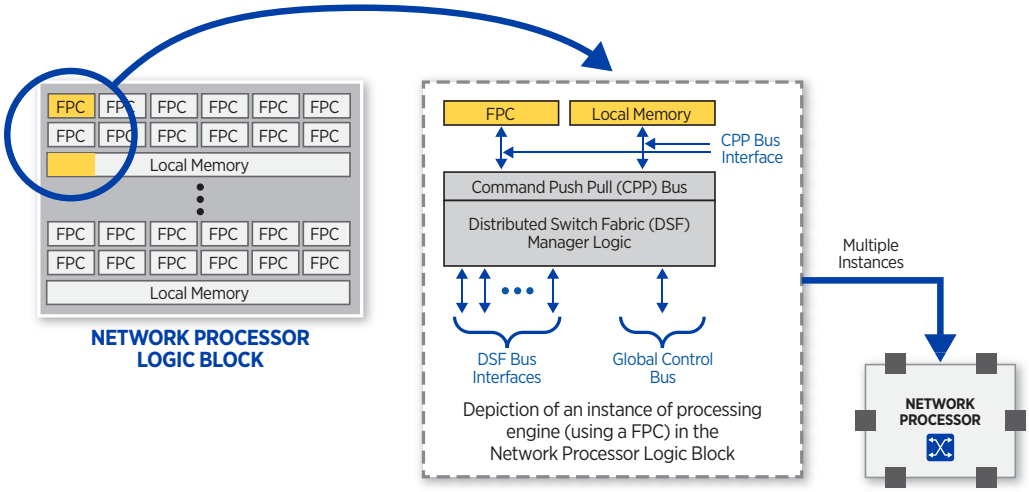

Figure 2. Details of the Network Processor Logic Block

Distributed Switch Fabric

The Distributed Switch Fabric (DSF) interconnect is the main global bus in an NFP. As shown in Figure 3, an NFP is physically implemented as a tiled array of Logic Blocks. In each Logic Block, bus agents provide the connectivity from the Logic Block to the rest of the NFP. The DSF is logically global but has no central logic. It is physically implemented entirely from near-neighbor connections between adjacent blocks (shown as connector blocks in Figure 3). Depending on the number of DSF interfaces implemented in each Logic Block, the result can be a K-dimensional fabric. In the specific example shown in Figure 3, each Logic Block has six DSF interfaces (K=6). Each 64-bit wide DSF interface link at 1GHz delivers 128Gb/s of bidirectional bandwidth into each Logic Block at the nodal point. So, a total of K*128Gb/s throughput is theoretically possible across each Logic Block. In the Figure 3 example, each Logic Block can support 768Gb/s of bidirectional bandwidth.

The distributed nature of the DSF enables the creation of an identical overlay mesh for all Logic Blocks as shown in Figure 3. Data transfers on the DSF can be orchestrated programmatically. The DSF command syntax is extensible and supports commands for data transfer and even processing at remote bus agents in a different island. For example, FPCs can create data transfers such that accelerator blocks can have data transferred to/from memory and/or external I/O directly, and issue commands to process data at a remote location where it resides, minimizing data movement and processing time.

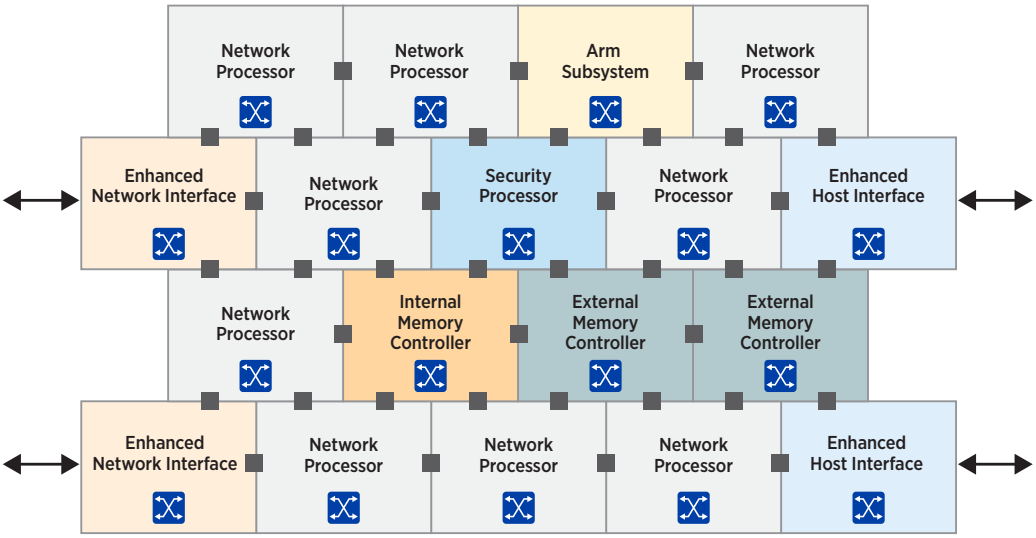

Figure 3. The Distributed Switch Fabric and tiled design

Processing Memory

The NFP has a second type of Logic Block to enhance data movement efficiency - unique Processing Memories that combine data storage, access and compute. Each Processing Memory comes with multiple engines to support an extensive array of functionality for data storage and manipulation right in the memory (see Figure 4). At every memory unit, in addition to a traditional controller, compute engines support a variety of operations, each processing data based on the commands received over the DSF. When combined with the ability to execute remote commands on the DSF, Processing Memory enables significant compute at the memory, including atomic operations, bulk transfer, distributed statistics collection and a wide range of table and trie data-structures. Many network functions can be executed on data where it resides in memory, reducing data movement costs and processing time.

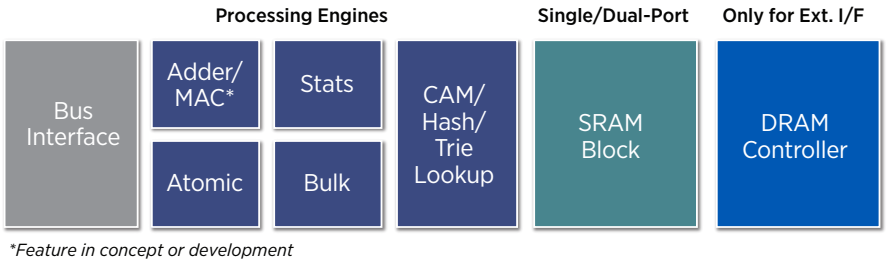

Figure 4. Processing memory architecture

Enhanced Network Interface

The NFP implements Enhanced Network Interfaces (ENIs) in which logic attached directly to L2 interfaces conditions packets as they are received by and when they leave the NFP. As shown in Figure 1, the ENI provides a wide array of dedicated hardware functionality to condition packets entering the NFP to minimize processing costs in the rest of the NFP.

On packet ingress, the ENI sequences incoming packets, classifies and prioritizes packets and flows according to preconfigured rules, and prior to delivery for further processing by other islands in the NFP. Pre-classification reduces the computational load on FPCs and enables distributed computing with memory engines. On the egress side, the ENI reorders packets, can modify packets, and using a sophisticated Traffic Manager, schedules and shapes outbound packets. Packet data, including the packet header and other L2 or L3 information, may be rewritten based on metadata scripts provided by the FPCs and prepended to the packet.

Enhanced Host Interface

This section describes the Enhanced Host Interface (EHI) Logic Block in an NFP. Current NFP devices use PCIe Gen3x8 Design Blocks (up to four of them). The EHI combines multiple PCIe, FPC and Processing Memory Design Blocks, along with a DSF interface for connectivity to other Logic Blocks. Each PCIe block in the EHI Logic Block enables transactions to be both received and initiated. The number of blocks in the design varies across implementations. When configured as a PCIe Endpoint, transactions received allow access to internal registers, engines and local memory, as well as other Logic Blocks, such as the Internal Memory Controller and the Arm Subsystem. When configured to be a Root Complex, the PCIe generates link transactions to discover, configure and manage other endpoints. Supporting both modes and processing these transactions, the PCIe component provides both a control and data plane interconnect for a wide range of platforms and applications.

The EHI includes an adequate number of FPCs and local memory along with the PCIe Design Blocks, enabling programmatic flexibility in how data is delivered to the host. For example, features like programmable receive side scaling (RSS) can be implemented to enable application-specific optimized delivery of data to host CPU cores.

THE NFP IS PROGRAMMED USING A RUN-TO-COMPLETION MODEL WITH TIGHT CONSISTENCY FOR CONTROL FLOW AND RELAXED COHERENCE FOR PACKET DATA PROCESSING.

3. PROGRAMMING THE NFP

The NFP is programmed using a run-to-completion model with tight consistency for control flow and relaxed coherence for packet data processing.

Run-to-Completion

In this model, FPCs are used in one or more groups of workers, with all the FPCs in one group servicing a common work queue. On packet ingress, packets leaving the ENI Logic Block are placed into a work queue. On egress, a reordering block in the ENI sequences packets to leave in the order they entered the NFP and passes the packets to a traffic manager. In between ingress and egress, packets are processed by one or more FPC groups in a pipeline. Each group services packets from a work queue and inserts its output into another work queue. FPCs in a work queue also update state for the flow to which a packet belongs. The assignment of work queues to FPC groups is managed by the developer. In general, FPCs in one cluster in the Logic Block are typically assigned to a common work queue since they can share their instruction store. Though packet processing order is not guaranteed, global packet ordering is preserved on packet egress.

Relaxed Coherence/Single Data Copy

Internal memory in an NFP is realized as multiple memories in multiple Logic Blocks. However, all memory in an NFP operates as part of a global address space. Though not all memories have the same access latency, the DSF enables high-bandwidth and low-latency data transfers between any data processing element and memory. Therefore, application logic on FPCs and other processing elements on the DSF can operate with relaxed coherence, with a single copy of all data. Any processing element operates on a single copy of the data even if the cost of accessing the data is not uniform across different processing elements. Relaxed coherence eliminates the costs associated with hardware or software to maintain multiple coherent copies of data. Multi-thread support in the FPCs ensures the FPC's datapath does not stall waiting for data on reads.

Programming Abstractions

The run-to-completion programming model can support both network-to-network and network-to-host packet flows. The model can be presented to the programmer in a variety of abstractions.

- The NFP can be programmed directly with low-level control in microcode. The programmer controls all aspects of the NFP's operation. Developers have implemented virtual switching functions using this programming method.
- With a toolchain from Netronome, the NFP can also be programmed at a higher level of abstraction using a variant C language optimized for this architecture. Developers have implemented network virtualization and overlay functions based on virtual routing and MPLS using this programming method.

The NFP can also be programmed in models in which the programmer has to code just the network datapath functionality in a high-level language. A software infrastructure from Netronome manages memory, compiles the high-level language and services network/host interfaces.

- eBPF is a mechanism by which custom functionality can be added dynamically to the Linux kernel. Netronome has upstreamed tools for transparent offload of eBPF/XDP applications. Developers can transparently offload eBPF/XDP applications to the NFP with an open source toolchain. Developers have demonstrated offload for many applications including load balancers, DDoS filters, and intrusion detection system filters.
- P4 is a relatively new high-level network datapath programming language in which developers can programmatically specify a match-action datapath. Netronome's SDK provides a toolchain for developers to compile P4 applications to NFPs.

Over 60 universities and companies are developing eBPF/XDP and P4 applications on the NFP through the open-nfp.org research portal.

4. CONCLUSION

The NFP SoCs are used in SmartNICs to provide in-field flexibility to cloud and data center operators. These chips use the advanced Netronome IP blocks and apply efficient data movement techniques with relaxed coherence to improve the power-performance of a programmable architecture for data-intensive applications. The key components in the NFP architecture include a scalable distributed switch fabric, processing memory, accelerators and flow-processing cores implemented as Design Blocks and grouped into functional Logic Blocks. NFPs use a run-to-completion programming model that supports a variety of programming methodologies. Several generations of NFPs, with throughput ranging from 10Gb/s to 200Gb/s, have been designed and field-hardened across multiple fabrication processes and are in production.

NETRONOME

Netronome Systems, Inc.
2903 Bunker Hill Lane, Suite 150 Santa Clara, CA 95054
Tel: 408.496.0022 | Fax: 408.586.0002
www.netronome.com