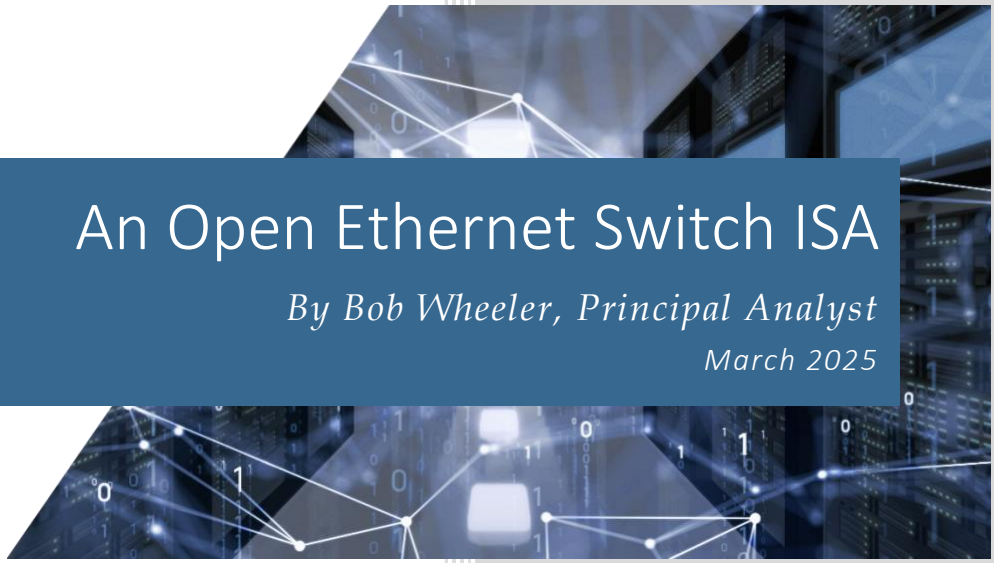

An Open Ethernet Switch ISA

By Bob Wheeler, Principal Analyst

March 2025

Wheeler's NETWORK

www.wheelersnetwork.com

In the AI era, silicon-design cycles have become a limiting factor in feature velocity. Network programmability has never been more critical, yet flexibility cannot come at the cost of performance and power efficiency. With its X-Switch architecture, Xsight Labs targeted the sweet spot for data-center switching, balancing performance and power while maximizing programmability. Now, to encourage innovation and adoption, the company opened the instruction sets for its X-Switch programmable Ethernet switch architecture. Xsight Labs sponsored the creation of this white paper, but the opinions and analysis are those of the author.

Assembling a Pipeline Alternative

When it comes to programmable data planes for high-speed networking, it has proven difficult to decouple data-path code from the underlying hardware architecture. Maximum programmability can lead to low performance and poor power efficiency. At the same time, programmability is important for feature velocity, futureproofing, and customization. These factors have only become more critical in the AI age, as the infrastructure-deployment cycle becomes compressed. The challenge, then, is in creating programming models that are performant while maximizing flexibility and abstraction.

Long before the generative-AI boom, startup Xsight Labs developed its programmable X-Switch architecture aimed at data-center Ethernet switching. It has since sampled two generations of switch silicon, culminating in the X2, a 5nm 12.8Tbps chip (see Xsight Recharges the Cloud ToR). The company's founders applied learnings from others' programmable-switch architectures, most of which failed to achieve broad adoption in the largest (hyperscale) data centers. The X-Switch design maximizes flexibility through run-to-completion execution and shared memory resources. It also scales up and down in bandwidth using a modular top-level design, as Figure 1 shows.

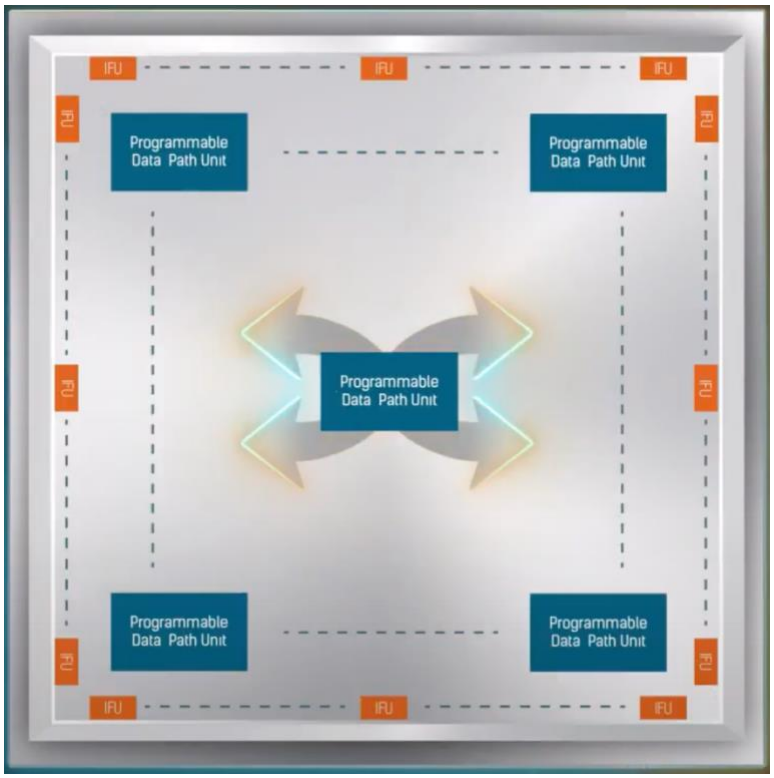

FIGURE 1. X-SWITCH SCALABLE ARCHITECTURE
(Source: Xsight Labs)

Xsight's approach specifically avoids the constraints created by the typical packet-processing pipelines found in most Ethernet switch silicon. These pipelines instantiate a fixed number of stages, which can guarantee minimum packet rates and maximum latency. As soon as a new feature exceeds what fits into a single pipeline pass, however, packets must be recirculated for additional processing, usually halving packet rate and doubling latency. By contrast, X-Switch maintains line rate up to the available instruction-cycle budget for a given packet size, after which additional instructions result in gradual performance degradation. Pipelines also often partition table and memory resources such that they are accessible only at specific stages. Sequential execution limits opportunities for parallelism, which X-Switch maximizes through parallel threads and asynchronous accelerators.

As we detailed previously, Xsight supplies drivers, multiple APIs, and an instruction-accurate simulator for the X2. The X-Switch assembler was, until recently, a company-internal tool. A customer needed a P4 compiler for X-Switch, however, to transition from another vendor's switch chips. Rather than develop the compiler, Xsight opened up its instruction set architecture (ISA) and assembler so the customer could develop it instead. Following the success of that customer program, the company decided to open-source its ISA to start building a community around its architecture. Because P4 has well-known constraints, opening assembly-language programming enables a wider range of X-Switch use cases. Xsight's goal is to proliferate programmable switching, drawing a parallel to the Arm instruction set in the CPU space.

X-Switch Delivers Granular Programmability

As Figure 1 shows, X-Switch instantiations integrate a matrix of programmable data-path units (DPUs) surrounded by interface units (IFUs), the latter containing Ethernet MAC and SerDes blocks. (The X-Switch DPU is not to be confused with a data-processing unit, such as Xsight's own E1 chip.) Each X-Switch DPU contains all the logic and memories for packet processing, including a programmable packet-forwarding engine (PFE).

As Figure 2 shows, the PFE comprises two distinct programmable stages, which have different instruction sets and program flows. In the first stage, the programmable parser receives up to 256 bytes of the packet header along with metadata from the ingress IFU. The parser walks through header fields and generates keys for associated lookups. Its program flow uses a protocol graph as well as a program counter. The parser's primary function is to preprocess the packet in a deterministic timeframe, avoiding accesses to memory or other external resources.

The second PFE stage, the Match-Action Processor (MAP) complex, handles the majority of packet processing. This complex integrates tens of MAP cores along with instruction and scratchpad memories as well as hardware accelerators (coprocessors). Each MAP core is a Harvard-architecture CPU with registers, a program counter (PC), instruction memory, and separate data memory. Each packet is assigned to a MAP core (or thread), which processes it to completion. Certain instructions initiate asynchronous operations handled by the coprocessors, such as lookups, memory load/store, checksum, and a few others. These nonblocking instructions enable the MAP core to continue executing until it encounters an explicit barrier.

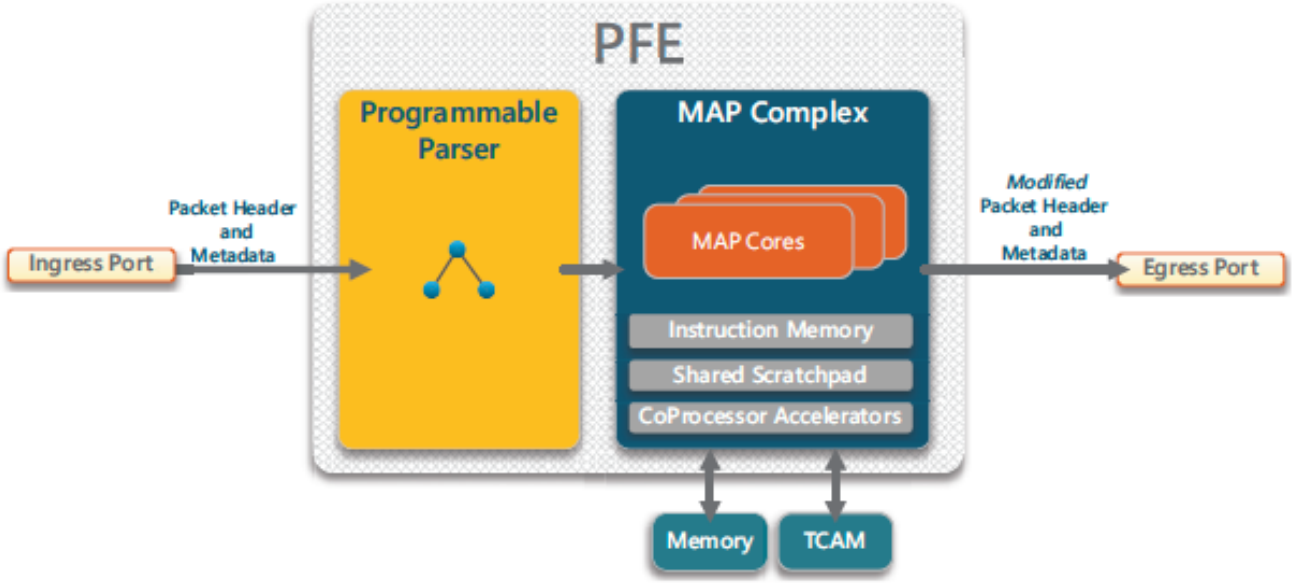

FIGURE 2. PROGRAMMABLE FORWARDING ENGINE
(Source: Xsight Labs)

A high-bandwidth X-Switch design such as the X2 contains tens of DPUs, each of which contains tens of MAP cores. All told, the chip includes thousands of MAP cores, with each one processing a packet from ingress to egress. This architecture allows a far greater level of parallelism in packet processing compared with pipeline-based designs, which typically have a few pipelines, each of which executes serially.

Simple Bespoke Instruction Sets

Because the parser and MAP cores have different functions and program flows, they also have different instruction sets. Xsight includes both instruction sets in its publicly available X-Switch Instruction Set Architecture document. The parser instruction set is more specialized, as it is designed to walk a protocol graph and pass user-defined metadata to the MAP thread that will process the packet. In addition to a program counter, the parser uses a transition table that codifies the protocol graph. It also uses a pointer into the packet-data buffer, called the "cursor," which is used to load/store data to/from a set of four 128-bit general-purpose registers. The parser instruction set (Parser ISA) has only 20 instructions, which include basic arithmetic operations, load/store, and a conditional branch. The programmer must understand the protocol-graph structure and traversal before writing parser microcode.

The MAP thread receives the parser's output in the form of Standard Meta Data (SMD), which is stored in a pre-allocated structure in per-MAP processing memory (PMEM). The parser also preloads two special-purpose MAP registers with header flags and offsets. The MAP core contains eleven 128-bit general-purpose registers. When the parser executes a HALT instruction, the MAP thread automatically starts executing at an entry point associated with the port type, such as an ingress port or the CPU interface.

An important feature of MAP programming is support for asynchronous operations, eight of which can be outstanding. The Lookup/Asynchronous Operation Flag (LFLAG) is really two flags, which indicate completion status and result status, respectively. A special SYNC instruction waits for operation completion based on an LFLAG bitmap, providing a barrier when the result is required. A dependency checker, implemented in hardware, maintains a list of registers awaiting a write-back operation, which

can take many cycles for coprocessor operations. This hardware performs an implicit SYNC when an instruction accesses a register in the dependency list.

The MAP instruction set (MAP ISA) includes many specialized instructions associated with coprocessors and external memories. These include atomic operations for counters, meters, load balancing, checksum, random-number generation, queue management, and to send a frame. They also include operations that manage structured memory and buffers, including copying a modified header back to frame memory. The MAP thread can also issue an asynchronous REPARSE operation to return a packet to the parser for an additional pass.

Aside from the intricacies of coprocessor functions, the rest of the MAP ISA is typical of a simple 32-bit RISC design. It includes basic arithmetic operations, load/store, bit/byte manipulation, jump/branch, and call/return. Altogether, the MAP ISA has just over 50 instructions. Note that there is a single return-address register, meaning the MAP core does not have a call stack. Xsight withheld the instruction-memory size, but typical microcode should be compact.

Table 1 shows a brief section of sample MAP assembly code, which initiates a longest-prefix-match lookup and then updates a counter. It then uses a SYNC instruction to wait for the lookup result, which is used in a conditional branch. Xsight's assembler supports macros for register assignments, easing readability, but we omitted the macro ASM here owing to space constraints.

ASM	Comments
CONCAT.CD R5.2, 0, R1.3, 2, 12	Build LPM lookup key: ipv4_fib_lkp_key[44:32] = port VRF configured in the port metadata
MOV R5.3, R4.3	ipv4_fib_lkp_key[31:0] = packet DIP extracted by parser
LKPLPM.LF5.R R0.3, R0.3, -1, -1, R5, R5, 0, 4	Start asynchronous LPM lookup, result will be written to R0.3 (ip_fib_lkp_result)
CONCAT.CD R0.1, 0, R3.1, 0, 16	Calculate packet length: IP length is extracted by parser
SUB R0.0, R12.0, 0, 8, R12.0, 16, 8	IP header offset calculated as L3 offset - ETH offset
ADD R0.0, R0.1, 0, 16, R0.0, 0, 8	Packet length = IP header offset + IP length
CONCAT.CD R0.1, 0, R1.0, 17, 15	RX counter ID is configured in the port metadata
MOVI.CD R6.1, 1	Counter delta format is [num of packets, num of bytes] = [1, packet len]
CONCAT R6.3, 0, R0.0, 0, 16	
COUNTER RN, R6, R0.1, 0, 8, 2	Update RX counter
SYNC.N 32, ipv4_fib_lookup_no_match	Sync LPM lookup, if miss branch to 'no LPM match' processing
BRBTSTSET R0.3, 16, ecmp_lookup	If LPM lookup result type == ECMP, continue to ECMP lookup

TABLE 1. MAP-CODE SNIPPET

(Source: Xsight Labs)

Programmability for Forward and Backward Compatibility

As mentioned above, the first external customer for Xsight's assembler used it to develop a P4 compiler. That customer is Oxide Computer, which develops rack-scale systems combining compute, storage, and networking for on-premises (private) clouds. The company's first-generation systems include a 12.8Tbps Ethernet switch based on Intel's Tofino 2 switch chip, which is programmable using

the P4 language. Oxide sought an alternative in part because Intel discontinued development of its Tofino line, but also because it wanted an open ISA rather than a closed ecosystem. By working with Xsight, it was also able to create a hardware-upgrade path that provided backward compatibility for its P4 code base.

The P4 community is now a project hosted by the Linux Foundation. Although the community has extended the language over the past decade, packet processing is still handled serially in a match-action pipeline. By contrast, the assembler enables Xsight customers to take full advantage of the X-Switch architecture's parallelism, breaking free from P4's limitations. The tradeoff is that performance may be less deterministic, potentially requiring analysis and tuning using the X-Switch simulator. Xsight also supplies an X2 view for GNS3 (Graphic Network Simulator), enabling simulation of network systems that include X2-based switches.

The X-Switch architecture enables advanced application use cases, such as the fast link recovery shown in Figure 3. In the X2, hardware detects a link failure and switches the associated port to loopback mode to avoid packet drops. When the spine switch (S1) receives a packet destined for the leaf switch (T3) connected through the failed link, the microcode can encapsulate (tunnel) the packet and reroute it to another spine switch (S0) with a working link to the leaf. The recovery-path spine decapsulates the packet and sends it to the original destination leaf. This link-recovery process is all handled in microcode, enabling recovery in less than 100 microseconds, far faster than BGP convergence in the control plane, which can take up to several minutes.

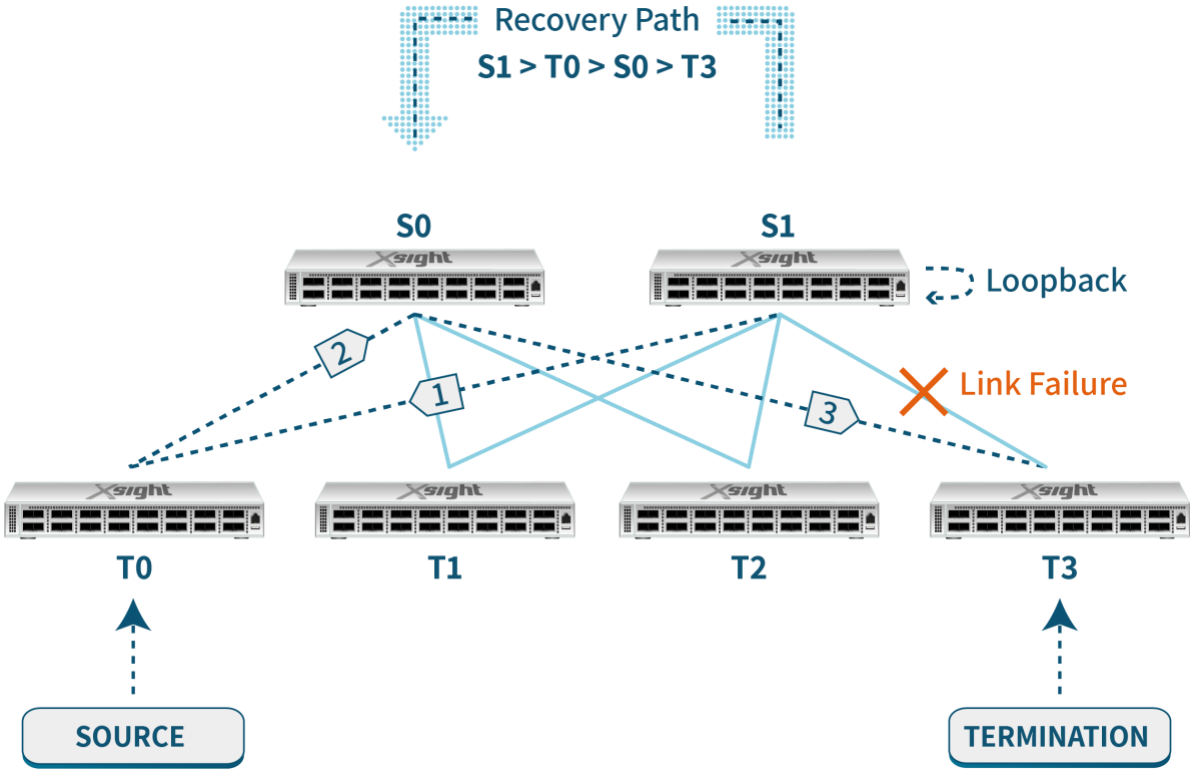

FIGURE 3. FAST LINK RECOVERY
(Source: Xsight Labs)

Congestion-aware routing is another advanced use case, which comprises measuring and signaling congestion as well as load balancing across the network fabric. For a given fabric link, X-Switch can measure flow rate, latency, and queue level. When congestion is detected, the microcode can signal

using Explicit Congestion Notification (ECN), Congestion Signaling (CSIG), or other customer-defined mechanisms. In conjunction with the hardware queue manager, the microcode can implement flowlet-based load balancing and packet spraying across fabric links.

In addition to handling custom protocols, X-Switch programmability enables forward compatibility with new standards such as the Ultra Ethernet Transport (UET) protocol. Figure 4 shows two advanced features expected in the first release from the Ultra Ethernet Consortium (UEC). Packet trimming (PT) is an alternative to dropping packets on congested links so that the destination port still receives the packet header containing congestion information. In X-Switch, the MAP microcode updates the header, truncates the payload, and enqueues the packet in a high-priority queue for faster congestion signaling.

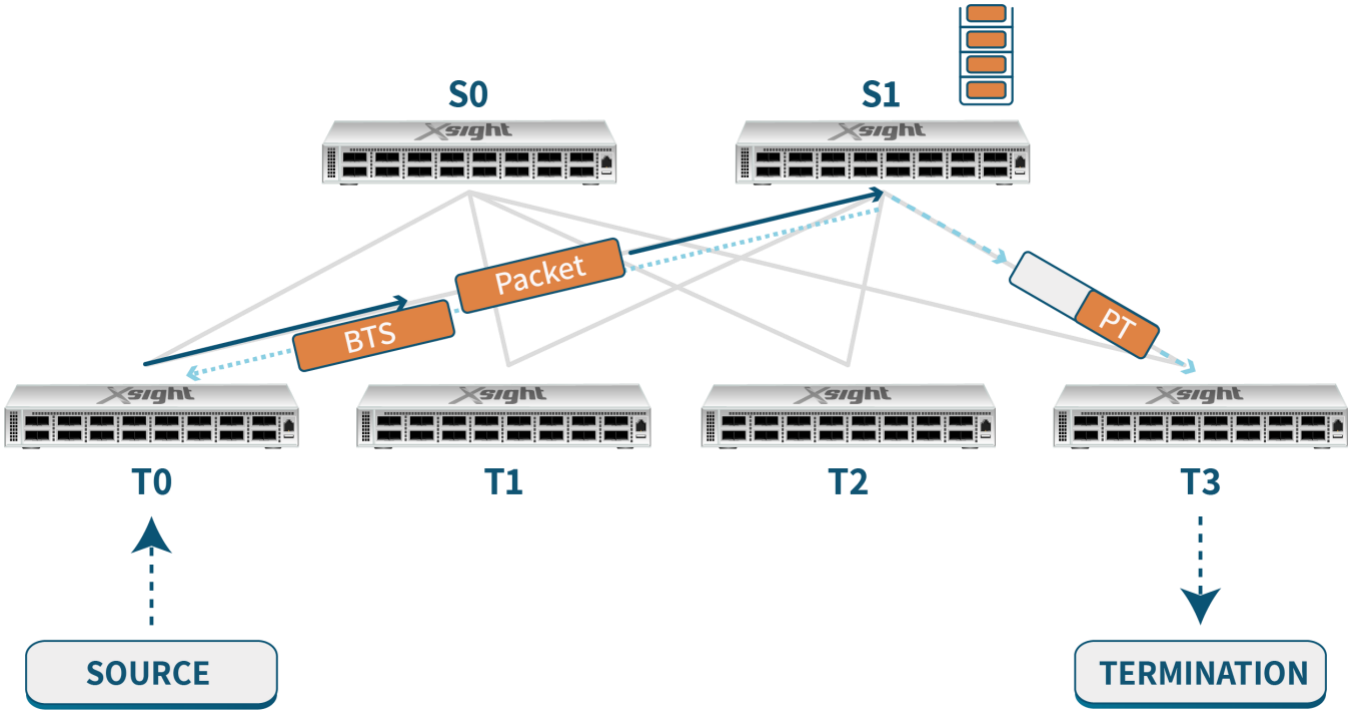

FIGURE 4. UET ADVANCED FEATURES
(Source: Xsight Labs)

Another expected UET feature is back-to-sender (BTS) signaling, which is similar to packet trimming except that the original packet is returned to the sender. Again, this feature is easily implemented in MAP microcode. The UEC's 1.0 specification is only the beginning of efforts to extend Ethernet for massive-scale data-center networks. Data-plane programmability will be essential to keeping pace with the evolving specifications.

Switch Hardware for the SDN Toolchest

Programmable network data planes have a long and checkered past, dating back to the network-processor explosion of the early 2000s. From an architecture perspective, there is no right answer to programmable data planes, as different applications emphasize various requirements including packet rate, power efficiency, and table scale. SmartNIC requirements, for example, are very different from those of Ethernet switches. With X-Switch, Xsight targeted the sweet spot for data-center switching, balancing performance and power while maximizing programmability. It traded the determinism of a pipeline for the graceful degradation of a run-to-completion model.

Reviewing the X-Switch programming model, our assessment is that any programmer that understands packet processing and can write RISC assembly code should easily master the MAP ISA. The Parser ISA is more esoteric, but Xsight supplies a Python wrapper that customers can use instead of writing assembly code. Still, the pool of network-savvy assembly-language programmers is small, so X-Switch programming remains highly specialized.

Democratizing the X-Switch architecture is a tall order, but the P4 language promises a nice entry point for customers and researchers alike. Both switch and SmartNIC chips supporting P4 have shipped in volume, proving the language can bridge multiple silicon architectures. X-Switch adds a modern and flexible design to this mix, opening opportunities for new P4–language extensions. As programmers gain an understanding of the X-Switch architecture, they can transition to Python code to take full advantage of the design's parallelism. Opening its ISA to the community will help incubate the architecture as Xsight delivers the X2 and develops future switch chips.

Bob Wheeler is an independent industry analyst covering semiconductors and networking for more than two decades. He is currently principal analyst at Wheeler's Network, established in 2022. Previously, Wheeler was a principal analyst at The Linley Group and a senior editor for Microprocessor Report. Joining the company in 2001, he authored articles, reports, and white papers covering a range of chips including Ethernet switches, DPUs, server processors, and embedded processors, as well as emerging technologies. Wheeler's Network offers white papers, strategic consulting, roadmap reviews, and custom reports. Our free blog is available at www.wheelersnetwork.com.

