The Linley Group

MICROPROCESSOR report

Insightful Analysis of Processor Technology

ESPERANTO MAXES OUT RISC-V

High-End Maxion CPU Raises RISC-V Performance Bar

By Linley Gwennap (December 10, 2018)

Esperanto ne est-as nur-a lingvo jam ne. Esperanto is also a startup combining the two hottest processor technologies: AI acceleration and RISC-V. This business plan, along with an all-star engineering team led by CEO and RISC pioneer Dave Ditzel, helped the company close a recent Series B round of $58 million—enough to bring its first processor to market. Esperanto completed the RTL for this product in September, so we expect samples to arrive in 2H19 followed by production around mid-2020. Given the long lead time, the company is withholding details, but it began the unwrapping process by disclosing the microarchitecture of its ET-Maxion CPU.

The new processor chip actually includes two different RISC-V CPU designs. ET-Maxion is a high-performance out-of-order (OOO) microarchitecture for general-purpose code. ET-Minion is a smaller AI-focused core that includes a set of proprietary vector extensions. (The RISC-V vector committee has yet to approve an official version.) Esperanto has publicly described a design with 16 Maxion cores and 4,096 Minion cores, but we expect the initial product will be smaller, with no more than half as many cores. The processor will run Linux and other high-level software on the Maxion cores while delegating AI-intensive workloads to the Minions.

The company expects Maxion to deliver strong performance while maintaining excellent power efficiency. To achieve this performance, the CPU can decode four instructions per cycle and issue them to five execution units in the OOO core. It implements the base RV64GC instruction set, a common denominator among current 64-bit RISC-V designs. Using a 10-stage pipeline, Maxion targets a relatively modest 2.0GHz clock speed in 7nm technology, boosting power efficiency. It should considerably outperform any other announced RISC-V CPU.

Initially, Esperanto plans to sell its processor on accelerator cards for data centers and other customers with large AI workloads. This business model follows the lead of other AI-chip startups such as Graphcore and Habana as well as established vendors such as Nvidia and Xilinx (see *MPR 11/12/18,* "AI Competition Begins to Bloom"). But given that Maxion is the most powerful RISC-V CPU

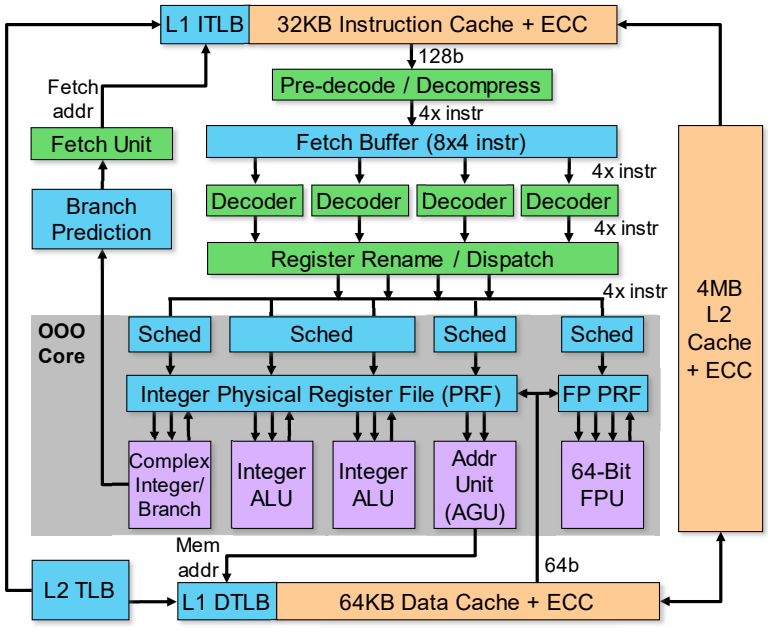

Figure 1. Maxion microarchitecture. The CPU can fetch, decode, dispatch, and retire four RISC-V instructions per cycle. The out-of-order (OOO) core has four independent schedulers for the five function units.

design yet revealed, several companies—including Western Digital, an Esperanto investor—have inquired about licensing it for other applications.

Starting With a Boom

With his professor David Patterson, Ditzel cowrote the seminal 1980 paper "The Case for RISC"; he went on to drive SPARC CPU development at Sun and later found x86 startup Transmeta. In 2014, he helped found Esperanto Technologies to design a new high-performance CPU. Since then, the team has grown to more than 100 engineers, including former AMD senior fellow Dan Bailey; Roger Espasa, who helped drive vector extensions for both Alpha and x86; Dave Glasco, who led SoC design teams at AMD and Nvidia; Duane Northcutt, who cofounded DriveScale, a cloud-infrastructure vendor; and former *Microprocessor Report* contributors Brian Case and Peter Glaskowsky.

The Silicon Valley startup initially planned to create its own instruction set, but it eventually determined the emerging RISC-V ISA could provide most of what it needed while simplifying the software development. RISC-V also provided a starting point for the Maxion microarchitecture: the Berkeley Out-of-Order Machine (Boom). (Chris Celio, who developed Boom at UC Berkeley, now works on Maxion.) The Boom tool can generate RISC-V CPUs with variable numbers of decoders and function units; Esperanto chose a design that supports four instructions per cycle, as Figure 1 shows.

Boom designs have several limitations, however. The simple six-stage pipeline limits clock speed to 1.0GHz even in 7nm. Branch prediction relies on the 20-year-old Gshare algorithm, and memory prefetching is nonexistent. Naturally, the university tool lacks provisions for debugging, performance monitoring, and reliability (such as ECC). Thus, Esperanto invested considerable effort to improve and upgrade the initial design.

Maxion uses a two-level branch predictor combining a smaller single-cycle branch target buffer (BTB) and a larger backing predictor (BPD). Esperanto designed a 2K-entry BTB with a state-of-the-art prediction algorithm to increase accuracy. It also added a path-based predictor for indirect branches. If the BTB indicates a taken branch, it creates a one-cycle bubble in the fetch sequence to fetch from the new target. If the slower BPD predicts a taken branch, the bubble is two cycles.

The predicted fetch address feeds into the 48-entry instruction TLB, which translates it to a physical address. The 32KB instruction cache then provides 16 instruction bytes. Although most designers employ parity protection for the I-cache (since the instructions are never modified), Maxion's I-cache includes options for ECC, parity, or no protection. A fetch buffer decouples the fetch process from the decoders. It has eight entries (32 instructions), enough to cover most branch-prediction bubbles. If the buffer is empty, instructions flow directly to the decoders. Not counting the buffer, the front end requires four pipeline stages.

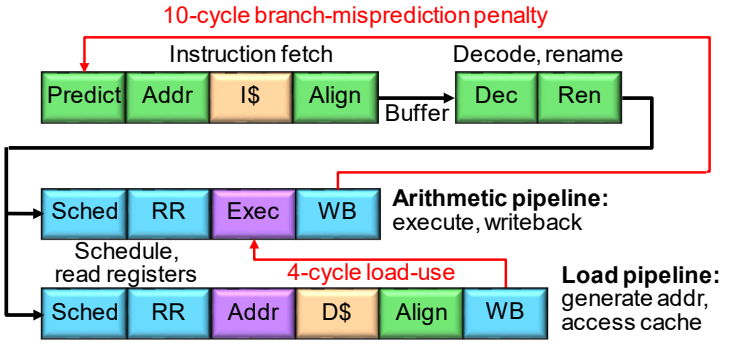

Figure 2. Maxion pipeline. The CPU requires 10 stages for basic integer instructions and 12 stages for a load instruction that hits the data cache.

Show a Little Compression

The four decoders can handle all basic RISC-V instructions including the M (integer multiply/divide), A (atomic), F (32-bit floating point), and D (64-bit floating point) extensions. Esperanto added a set of special registers for debugging, performance monitoring, and other low-level functions. Although Boom doesn't implement the C (compressed) extension, the company added this capability to Maxion for broader compatibility with other RISC-V CPUs. Compressed instructions also increase code density and therefore the cache-hit rate, boosting performance.

The C extension defines a set of 16-bit instructions that cover the most common functions and can access only a quarter of the register file (see *MPR 3/28/16,* "RISC-V Offers Simple, Modular ISA"). These instructions can mix freely with standard 32-bit instructions, which are necessary to perform the full set of functions. Mixing instruction sizes, however, creates challenges for a high-performance CPU. For example, the C extension allows 32-bit instructions to be unaligned, meaning a single instruction can span a cache-line boundary or even a page boundary. The branch predictor must comprehend half-word addresses, and the CPU may issue redirections for partially decoded instructions. Furthermore, Esperanto had to extend the standard RISC-V verification tests to cover all these corner cases.

Maxion detects 16-bit instructions as they come from the I-cache. Because every 16-bit RISC-V instruction maps to a single 32-bit RISC-V instruction, the hardware simply expands compressed instructions before loading them into the fetch buffer. Although this approach adds some complexity to the front end, it greatly simplifies the decoders, which handle only 32-bit encodings. Once the instructions are decoded, the CPU renames the 32 architected integer registers to 128 physical registers; floating-point registers map to a separate 64-entry physical register file (PRF). The instructions then dispatch to the out-of-order core. Decoding, renaming, and dispatching squeeze into two pipeline stages, as Figure 2 shows.

Maxion offers five function units: two simple integer units; one complex integer unit that includes multiply, divide, and branch capability; one load/store unit; and one floating-point unit (FPU). Each has a 16-entry scheduler, except the two simple integer units share a single scheduler, resulting in a total of 64 entries. Each scheduler can issue one instruction per cycle (two for the shared scheduler) to its function unit(s); instructions issue when their operands are available and thus can proceed out of program order. After being issued, instructions read their operands from the PRF, execute for one or more cycles, then write their results back to the PRF. The reorder buffer (ROB) retires up to four instructions per cycle in program order; it can track 128 instructions in the reorder window.

Shouldering the Load

The complex integer unit resolves all branches, updating the branch predictors. If the branch result differs from the original prediction, the pipeline must be flushed and restarted from the new target address, causing a 10-cycle delay, as Figure 2 shows. The 64-bit-wide FPU is fully pipelined for multiply-accumulate (MAC) instructions, which have a four-cycle latency for both single- and double-precision operands.

The load/store unit calculates the effective address and places it in the 32-entry load queue or 32-entry store queue as appropriate. Entries that clear the queue proceed to the 32-entry data TLB. A 1,024-entry L2 TLB services both DTLB and ITLB misses, albeit with a one-cycle bubble; a hardware page-table walker (PTW) resolves L2 TLB misses. The CPU then accesses the data cache and (for a load) returns the result to the register file. The loaded value is available for subsequent instructions on the next cycle, yielding a load-use delay of at least four cycles, as Figure 2 shows. Accesses that miss the data cache move to the unified L2 cache. In the meantime, subsequent accesses in the load/store queues can access the data cache (hit under miss).

At 64KB, Maxion's data cache is larger than its instruction cache and implements ECC protection to avoid data loss. A 4MB L2 cache services both I- and D-cache misses and is also ECC protected; presumably, multiple Maxion cores share this cache. The CPU contains stride prefetchers for the L1 and L2 to reduce cache misses.

As Figure 3 shows, the layout allocates the most area to branch prediction (including various memories), the floating-point unit, and the prefetcher. Accurate branch prediction minimizes the number of 10-cycle misprediction penalties. Similarly, the prefetcher avoids long accesses to the L2 cache and DRAM, so spending die area on these units provides a strong performance boost.

Reaching Arm's Middle

On the basis of its testing and simulations, Esperanto expects Maxion's performance per clock (IPC) to fall between that of Arm's Cortex-A57 and Cortex-A72 CPUs on the SPECint_2006 benchmark, as Figure 4 shows. But the A72 will be four years old by the time Maxion reaches production. A more appropriate comparison is with Cortex-A76, Arm's most recent high-end mobile CPU (see *MPR 6/4/18,* "Cortex-A76 Revamps Core Design"). By our estimates, based on published benchmark results, the A76 has delivers about 50% better IPC than the A72 and thus leads the projected Maxion score by a similar amount.

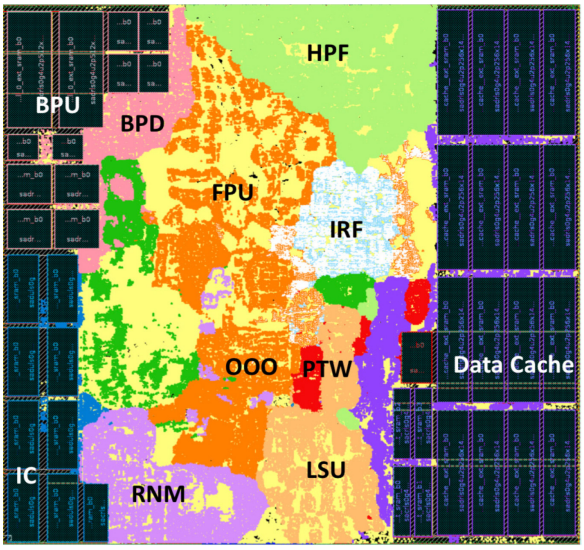

Figure 3. Preliminary Maxion layout. The CPU includes a 32KB instruction cache (IC), branch prediction (BPU and BPD), decode and rename logic (RNM), out-of-order logic (OOO), integer units (IRF), a floating-point unit (FPU), a load-store unit (LSU), a page-table walker (PTW), a prefetcher (HPF), and a 64KB data cache.

To achieve this IPC advantage, the A76 implements a considerably more powerful microarchitecture. Although like Maxion it's a four-issue machine, the A76 provides a second floating-point unit and, critically, a second load/store unit. Memory operations are often a performance limiter,

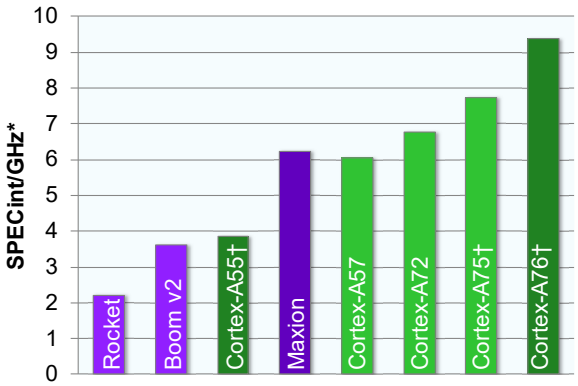

Figure 4. Per-clock performance. Esperanto projects its Maxion CPU will deliver per-clock performance similar to that of Cortex-A72, but it will still lag current high-end Arm CPU designs. *SPECint_2006, GCC. (Source: Esperanto, except †The Linley Group estimate)

For More Information

For an overview of Esperanto's plans, access its website at *www.esperanto.ai*. Extensive documentation on the Boom microarchitecture is at *github.com/riscv-boom/riscv-boom-doc*. Details on the Maxion design appear in a paper Esperanto presented at the recent RISC-V Summit, which will be available for download at *riscv.org*.

and handling two loads per cycle is a hallmark of modern high-performance CPUs. The A76 also has nearly twice as many scheduler entries as Maxion, creating a much larger reorder window. It features a private L2 cache with six-cycle latency, markedly faster than Maxion's large shared L2.

The other half of the performance equation is clock speed. Given that Cortex-A75 achieves 2.85GHz in 10nm, Arm expects Cortex-A76 (which uses the same pipeline) to reach 3.0GHz in 7nm—much faster than Maxion's projected 2.0GHz. To enable greater clock speeds, these Arm cores employ a 14-stage integer pipeline that's considerably longer than Maxion's 10-stage design. In particular, the Arm pipeline allows an extra cycle for both instruction- and data-cache access, relieving that bottleneck. Esperanto expects that Maxion's clock could exceed 2.0GHz in some configurations, but its shorter pipeline is unlikely to approach the speed of the A76. Combining its greater IPC and faster clock, the A76 will likely double Maxion's performance.

On the plus side, we expect Maxion to consume less than 0.5mm^2 in 7nm, making it about half the size of Cortex-A76, as Table 1 shows. (For comparison, we removed the A76's private L2 cache from its area.) Although some of this savings may come from the simpler RISC-V ISA, we believe most comes from the smaller scheduler and single load/store unit. Another big savings lies in Maxion's lone 64-bit FPU versus the A76's dual 128-bit FPUs. Those wider units implement Arm's Neon SIMD instruction set, whereas Maxion omits any vector instructions. To implement similar features, the bulky FPU in Figure 3 would need to be at least four times larger. The smaller area and lower clock speed should reduce Maxion's power, possibly enough to match the mobile-optimized A76 in efficiency.

	Esperanto Maxion	Arm Cortex-A76	Arm Cortex-A55
Instruction Set	RISC-V	Arm v8	Arm v8
Architecture Size	64 bits	64 bits	64 bits
Extensions	Compression	Comp, SIMD	Comp, SIMD
Max Decode	4 instr	4 instr	2 instr
FP/SIMD Units	1x64 bits	2x128 bits	2x64 bits
Load/Store Units	1 L/S	2 L/S	1L + 1S
Reorder Window	128 instr	~160 µops	None
Pipeline Length	10 stages	14 stages	7 stages
Max Speed*	2.0GHz+	3.0GHz	2.2GHz‡
SPECint/GHz†	6.2	9.4‡	3.9‡
Est SPECint†	12.5	28.1‡	8.5‡
Die Area*	<0.5mm2‡	0.9mm2‡	0.25mm2‡
First Products	mid-2020‡	4Q18	1Q18

Table 1. Comparison of high-performance CPUs. Maxion's size and performance falls between that of Arm's current "big" core (A76) and "little" core (A55). *In TSMC 7nm; †SPECint_2006. (Source: vendors, except ‡The Linley Group estimate)

Table 1 also compares Maxion with Cortex-A55, a CPU that targets power efficiency rather than maximum performance. The A55 is a simple in-order design that's limited to two instructions per cycle, although it's flexible in its ability to pair instructions (see *MPR 6/5/17*, "Cortex-A55 Improves Memory"). Despite the A55's relatively short seven-stage pipeline, it exceeds 2.0GHz in some 10nm products, possibly because Arm tuned this pipeline through several generations. Esperanto is withholding Maxion's die area and power until it completes the physical design, but the RISC-V core will be hard pressed to match the A55's power efficiency and small size.

Maximizing Performance per Watt

Using Arm as a well-known point of comparison, Maxion's performance should fall about halfway between that of the "big" Cortex-A76 and the "little" Cortex-A55. This ratio is consistent with the designs' relative microarchitecture complexity. Since Arm positions the A76 for best performance and the A55 for optimum efficiency, Maxion should meet its goal of strong performance and power efficiency, although it's difficult to assess the latter without any power data. Even though it falls in the middle of Arm's performance range, Maxion should considerably outperform any other announced RISC-V CPU, including SiFive's new 7 Series (see *MPR 11/12/18*, "SiFive Raises RISC-V Performance").

The Maxion core appears well suited to running Linux and a high-level AI driver that parcels out tasks to a multitude of Minions. Because AI workloads are highly parallel, the Esperanto chip will instantiate several Maxions for this purpose instead of creating a CPU with the most single-thread performance. Data-center operators are concerned about electricity cost, so they prefer processors with superior power efficiency.

Choosing RISC-V has helped Esperanto's development. By starting with the open-source Boom design, the startup says it completed Maxion's RTL is less than nine months. Instead of designing a wholly proprietary instruction set, it gains access to the growing base of RISC-V development tools and software.

RISC-V proponents, in turn, are excited about a new CPU that extends RISC-V performance at least into the range of Arm's Cortex family, but they're likely to be disappointed with their access to the core. Esperanto says it'll license Maxion, but we expect it to serve only a few customers, likely those with high volumes. IP licensing requires a considerable investment in tools, validation, and customer support, and the revenue is typically small. The company

has no plans to open source the Maxion design, leaving most RISC-V users to seek another path to higher performance.

Instead of licensing, Esperanto is focused on completing its first chip and selling it in AI accelerators. For this market, RISC-V offers little value to the end user; most customers develop AI models in a high-level framework such as TensorFlow and don't care about the hardware architecture. The company's challenge is to deliver an AI accelerator that's more efficient than those from other well-funded startups such as Graphcore, Habana, and Wave, all of which are at least a year ahead of Esperanto in reaching production. Maxion is a good start, but Esperanto's success will depend far more on the performance of its Minions. ♦