Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

MathStar Challenges FPGAs

New Reconfigurable-Logic Chips Have Massively Parallel Arrays

By Tom R. Halfhill {7/24/06-02}

Project managers searching for less costly alternatives to ASICs and SoCs are increasingly turning to FPGAs. If the anticipated volumes of the finished product aren't too great, FPGAs can save millions of dollars in nonrecurring engineering (NRE) expenses and chip manufacturing, despite the higher unit cost of a programmable-logic device. And the flexibility of making field upgrades without replacing an ASIC or SoC can be very attractive for some applications.

This trend would seem to favor more FPGA start-ups. However, two big companies—Altera and Xilinx—dominate the FPGA market, leaving room for only a few smaller players to carve out a niche. Finding paths around the Big Two has become a challenge. One company taking up the challenge is MathStar, founded in 1997 and based in Hillsboro, Oregon.

Microprocessor Report first wrote about MathStar in 2003, when the fabless semiconductor company unveiled its reconfigurable-processor technology at our Fall Microprocessor Forum. (See *MPR 12/15/03-03*, "Roll Your Own Array Processor.") MathStar sampled the first device based on that technology in September 2004 and revised it several times in response to customer feedback. Samples of the improved chip became available in August 2005, and MathStar is working on a second chip for delivery next year.

The long incubation hasn't been wasted. MathStar went public last October (Nasdaq: MATH), raising $27.6 million. Also, the company says it has landed 12 design wins—an impressive debut for an unorthodox product. One win is Honeywell, which is porting a MathStar design to a radiation-hardened fabrication process for aerospace systems.

Object Arrays, Not Gate Arrays

MathStar calls its device architecture a field-programmable object array (FPOA). It consists of SRAM-based programmable logic, much like a conventional FPGA, but it's programmable at a higher level of abstraction. Instead of tinkering with gate arrays, designers work with a massively parallel array of preconfigured function units.

Most of these units are identical ALUs or multiply-accumulate (MAC) units that can run autonomously. Others are register files shared by the ALUs and MACs. The first FPOA device has 400 of these 16-bit units woven together in a tightly coupled interconnect fabric. Around the periphery of the chip are SRAM banks, external memory interfaces, high-speed parallel I/O ports, and general-purpose I/O (GPIO) ports.

What's missing is a control processor. An FPOA requires some kind of host controller, such as a RISC processor or microcontroller. This requirement puts the FPOA in a gray area between FPGAs and full-fledged microprocessors. It's a fully programmable machine, but it needs an external controller to initially load the bitstream and retrieve the results. Most FPGAs need host controllers, too, although some have hard processor cores on chip, and others have enough capacity to implement a soft processor core in their gate arrays.

By preconfiguring the FPOA with custom-designed function units, MathStar says it can achieve higher performance than an FPGA with generic programmable logic. The

company's first device, the MOA1400D, runs as fast as 1.0GHz—much faster than a conventional FPGA. Because each of the 400 function units can perform one operation per clock cycle, the maximum theoretical throughput is 400 billion operations per second (BOPS). Moreover, MathStar guarantees that the device always runs at its maximum clock rate. That's a big advantage over conventional FPGAs, which often must run at a clock rate slower than their maximum frequency when programmers implement a complex design in the gate array.

Another advantage of MathStar's architecture is that it's suitable for fabrication in a mature CMOS process that doesn't suffer from the current leakage of the latest, most expensive processes. TSMC manufactures the MOA1400D in its 0.13-micron low-voltage process (LVOD), which holds typical power below 20W when running major applications. The fastest 1.0GHz devices will cost $285 in 1,000-unit quantities.

Building Blocks Are 'Silicon Objects'

It's obvious that MathStar put some thought into the FPOA architecture. The basic building blocks of the array are the 16-bit function units and register files, which MathStar refers to as Silicon Objects (trademarked, no less). These are full-custom logic blocks, not standard cells or synthesized models in generic gate arrays. Custom circuit design enables higher clock frequencies while conserving silicon and power. Figure 1 shows the three types of Silicon Objects populating the first FPOA chips.

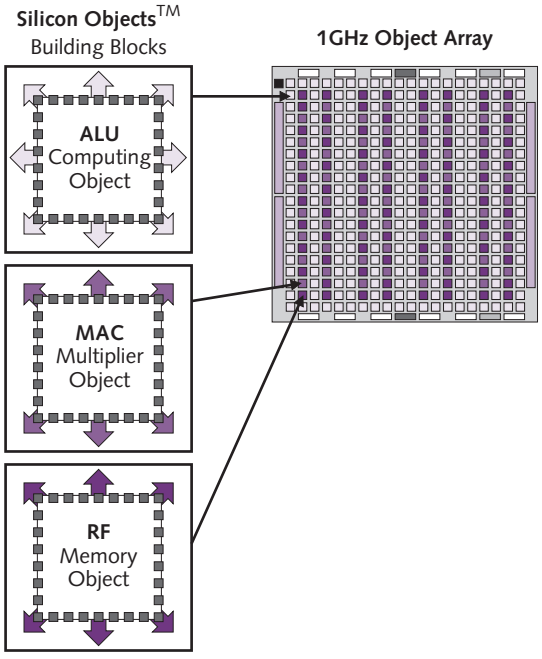


Figure 1. Initially, MathStar has created three types of Silicon Objects: 16-bit ALUs, 16-bit multiply-accumulate (MAC) units, and 64-entry register files. Silicon Objects are designed to be interchangeable, so different FPOA chips can have different complements of these objects in their arrays.

Each Silicon Object is exactly the same physical size and has the same I/O interfaces to the interconnect fabric, so they are interchangeable within the array. That is, MathStar can create new FPOA chips with arrays of different dimensions and different complements of Silicon Objects. Once created, the array is fixed for that device; customers can program the array but cannot alter its arrangement or complement of Silicon Objects. In this respect, an FPOA is more like a massively parallel microprocessor than a bucket-of-gates FPGA. Its programmable logic is programmable at a higher level, closer to the application software, rather than at the logic-gate level.

Higher-level programmability implies less flexibility. However, an FPOA is more than a massively parallel array of 16-bit function units. The tightly woven interconnect fabric allows designers to gang the Silicon Objects together in ways that aren't possible with multiple function units in a conventional processor. In this respect, an FPOA is more like an FPGA than a microprocessor. For example, designers can harness together multiple Silicon Objects to perform 32-bit or wider operations, or to execute multiple tasks in parallel. Alternatively, designers can link together chains of Silicon Objects to execute serial tasks in a pipelined fashion. Whereas an FPGA is a blank slate of logic gates, an FPOA is a blank slate of function units, register files, and interconnects.

As Figure 2 shows, the MOA1400D has 256 ALUs, 64 MAC units, and 80 register files, for a total of 400 Silicon Objects, arranged in a 20×20 array. All the ALUs are identical and always carry out their 16-bit operations in one clock cycle. All the MAC units are identical and always carry out their 16-bit operations at a pipelined rate of one per clock cycle. All the register files are identical, with 64 entries of 20-bit registers per file. Registers are dual ported and can store operands, intermediate results, or final results. Register files can also function as circular FIFO buffers or dual-ported general-purpose memories, depending on the needs of the application. For additional data storage, the MOA-1400D has 12 banks of internal RAM (IRAM) positioned at the edges of the array. Each bank is 19KB in size, organized as $2K \times 76$ bits, for a total of 228KB.

Although Silicon Objects operate on 16-bit data, their datapaths are actually 21 bits wide. Each operand carries five control bits that invoke combinatorial logic in the ALUs to carry out various operations. One control bit is a "valid" bit, which allows the ALUs to conditionally execute or conditionally bypass instructions. Registers don't need to store the valid bit, which is the reason they are 20 bits wide. MAC units don't have conditional instructions, so they ignore the valid bit.

ALUs support 38 possible instructions, and each ALU has a local queue for storing eight instructions awaiting execution. Developers can program each ALU individually and let it run autonomously. Some tight loops will fit entirely in a single instruction queue, allowing the ALU to loop endlessly

without relying on an external instruction stream. That's a key advantage over conventional microprocessors, which must constantly fetch instructions, even if the instructions stay resident in a cache. If a task is too large for an eight-instruction queue, developers can assign more ALUs to the task—as many as needed. Because multiple ALUs can share neighboring register files with zero latency, they can access each others' operands as local data without delays.

MAC units don't have an instruction set, but the control bits can reconfigure them on every clock cycle to perform a 16- × 16-bit multiply, accumulate, or multiply-accumulate. (Another control option is to multiply signed 15-bit fixed fractional numbers.) Multiplies execute in two cycles, and accumulation requires an additional cycle, but pipelining allows the unit to finish a MAC every cycle. The result of a 16- × 16-bit multiply is a 32-bit quantity, and accumulation produces a 40-bit result. As mentioned before, developers can gang together multiple MAC units to perform operations wider than 16 bits.

Dense On-Chip Fabric Connects Objects

The most critical architectural feature of a massively parallel processor is the on-chip network that links the computation units together. Without an efficient network, the chip's vast processing resources are wasted. The engineering trade-offs are communications versatility, interconnect speed, design complexity, and manufacturing cost.

Ideally, every node in a massively parallel array would be able to communicate with every other node in a single clock cycle. Unfortunately, the necessary wiring quickly gets out of hand in a large array. Wire delays (signal propagation) are another limiting factor. More wiring and shorter signal paths also inflate manufacturing costs, because they often require additional metal layers and smaller, more-expensive fabrication processes.

Consequently, massively parallel designs make compromises. Generally, they allow nearby nodes to communicate in one or a few clock cycles and require distant nodes to wait a little longer. This, in turn, creates a challenge for developers or

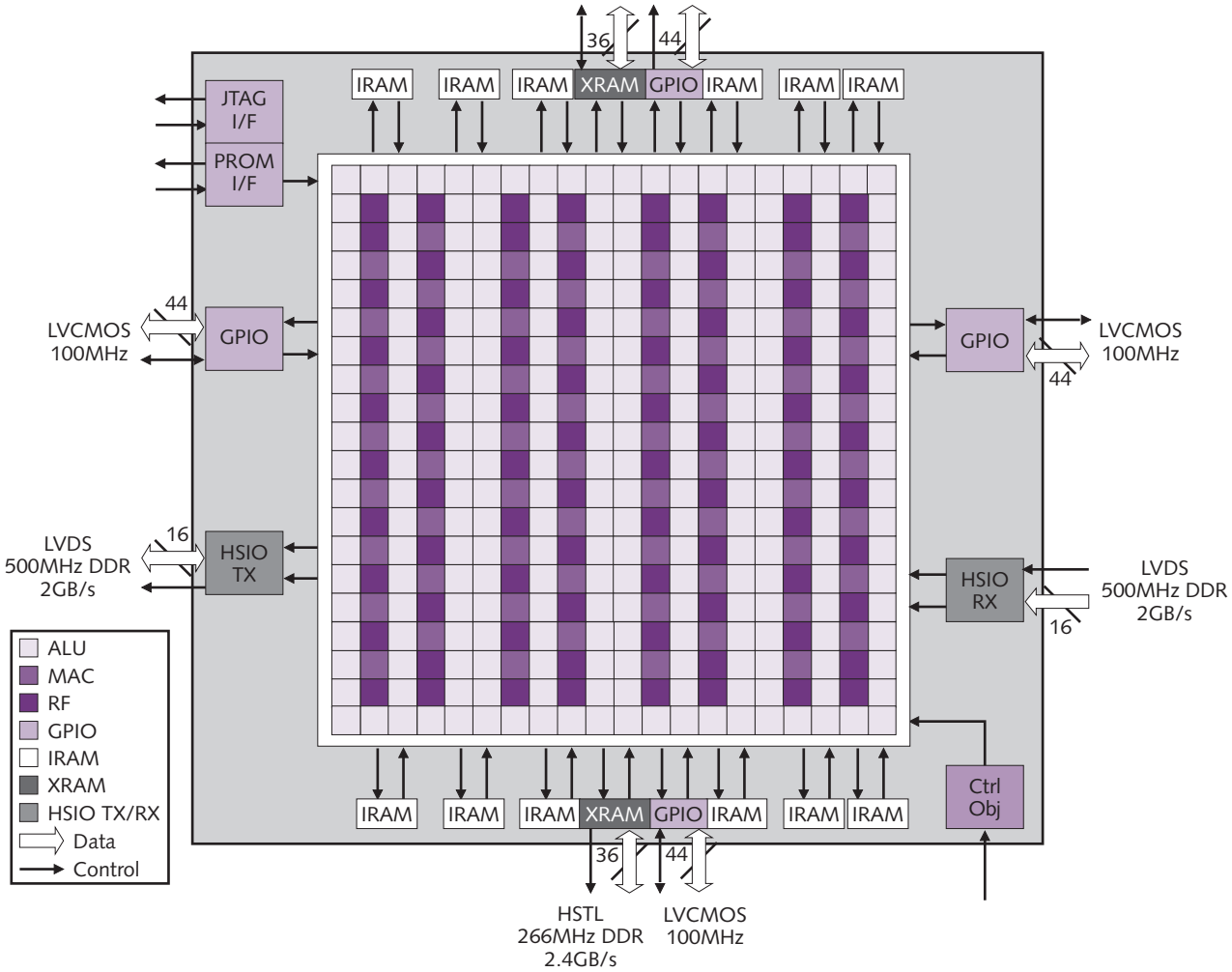


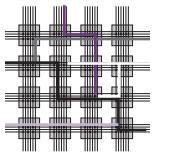
Figure 2. MathStar's first FPOA chip, the MOA1400D, has a 20 × 20 array of Silicon Objects, including 256 ALUs, 64 MAC units, and 80 register files. Future FPOAs may have larger or smaller arrays, as well as different complements of Silicon Objects, depending on their target applications.

the development tools they use. They need to map the software onto the array in a manner that leverages the fast local interconnects without losing too much performance when making longer hops through the network.

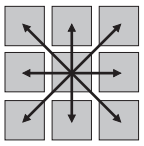
MathStar has patented its interconnect technology, but it's generally similar to the interconnects in other massively parallel processors. Each Silicon Object is a network node in a dense wiring fabric that allows multiple function units to share register files and communicate with other nodes at various speeds, depending on their distance from each other. (The entire fabric always runs at the chip's maximum clock speed, so distance is the only variable.) All the interconnects are 21 bits wide, to carry the 16-bit operands and five control bits mentioned earlier.

The fastest connections in the FPOA fabric are called nearest-neighbor links, which have zero latency. Each object has eight of these links: one each for north, south, east, and west, and one for each of the four diagonal directions. Actually, each connection has separate paths for input and output, so there are 16 nearest-neighbor buses for each object.

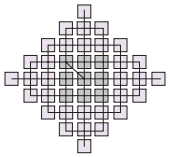
For longer hops, MathStar provides a "party-line" interconnect. Each object in the array has ten of these buses: three north, three south, two east, and two west. Impressively, the party-line connections allow an object to communicate with another object up to four nodes away in a single clock cycle—only 1ns at 1.0GHz. The compromise is that party lines don't run diagonally, as nearest-neighbor lines do. An object that's three nodes away in a diagonal direction will take two cycles to reach instead of one cycle. Still, that's pretty good for such a complex fabric. In the MOA1400D's 20×20 array, the worst-case signal delay is a corner-to-corner series of hops, which takes seven cycles. Figure 3 illustrates the FPOA's on-chip interconnects.



21-bit Interconnect Lanes
- 16 data bits
- Five control bits
- 1GHz speeds



Nearest Neighbor Interconnect
- Eight per object
- Range of one object (N/E/S/W, diagonals) in one clock cycle



Party Line Interconnect
- 10 per object
- Range of up to three objects away in one clock cycle

Figure 3. MathStar's on-chip interconnect fabric allows nodes to communicate with neighboring nodes in any direction with zero latency. Signals can travel as many as four hops in a single cycle, but diagonal communications take longer. The FPOA fabric compares favorably with the interconnects in other massively parallel devices.

External I/O Has Room for Improvement

Off-chip I/O is sufficient but not luxurious. For main memory, the MOA1400D has two reduced-latency DRAM (RLDRAM) controllers that can run as fast as 266MHz (effective DDR). Each controller has a 36-bit data interface, providing 2.4GB/s of maximum bandwidth. Programmable clock divisors allow the RLDRAM controllers to run at slower speeds if desired.

For high-speed parallel I/O, the MOA1400D has a pair of transmit/receive ports using low-voltage differential signaling (LVDS). Each transmit or receive port has a 16-bit data interface and runs as fast as 500MHz (effective DDR), providing about 4.0GB/s of aggregate bandwidth. For general-purpose I/O, the MOA1400D has two banks of GPIO ports, each with 48 bidirectional pins and programmable speeds up to 100MHz.

There's also a PROM controller and a JTAG interface on chip. The PROM controller loads the configuration image from external PROM at startup and initializes the object array. Reconfiguring the entire array—including the time it takes to preload the IRAM banks—requires 10–20ms. If the IRAM is not preloaded, reconfiguring the array takes only 2.5–5.0ms. That's much faster than a comparable FPGA. Reconfiguring a Xilinx Virtex-4 LX200, even without preloading the IRAM, takes 250ms; the Virtex-4 SX55 takes almost 500ms. MathStar's JTAG interface provides another way to configure the MOA1400D's array, usually for verification and debugging during development.

One missing feature is a dedicated DMA controller. Although MathStar says developers can create one using only seven Silicon Objects, that solution might not be acceptable if an application needs all 400 objects for other purposes. In addition, the MOA1400D lacks the integrated Ethernet ports and other high-function I/O controllers found in some high-end FPGAs. Inexpensive external controllers can fill those roles, at the cost of increasing the system's chip count.

MathStar has disclosed a second FPOA chip. Its array has 400 Silicon Objects, as the MOA1400D does, but the chip will have faster external I/O and support a greater amount of internal and external memory. The official announcement of this device will reveal more details. MathStar says future FPOA chips will probably integrate a DMA controller, new types of objects, and support for next-generation memory and I/O.

Fabric Maintains Consistent Timing

An important feature of MathStar's interconnect fabric is that special development tools inherently know the signal-propagation characteristics of the FPOA. Developers need not worry about the low-level details of signal propagation through the array. The tools automatically warn developers if they try implementing an algorithm that can't execute properly on the FPOA because of insufficient performance headroom. Therefore, developers don't have to achieve timing closure with these devices.

Consistent timing and early warnings about timing problems are significant departures from conventional

FPGAs. Normally, FPGA developers must run the chip at a slower clock rate to accommodate a more complex algorithm. If they ramp down the FPGA's clock speed too far, the algorithm may not execute quickly enough for the target application. To resolve the timing problem and achieve closure, developers must make last-minute adjustments to bring the algorithm into line with the fastest clock speed the FPGA can achieve. This tinker-and-test process is often frustrating, requiring several time-consuming iterations during a late stage of the project.

In contrast, an algorithm's complexity doesn't affect the clock frequency of an FPOA, and MathStar's development tools warn ahead of time if an algorithm is too complex to run at that speed. A 1.0GHz FPOA will always run at 1.0GHz, whereas a 500MHz FPGA might have to run at 250MHz. If a complex algorithm won't work at 1.0GHz and a faster FPOA isn't available or would cost too much for the target application, the only recourse is to modify the algorithm—but at least the developers will find out immediately, not later in the project.

MathStar says its higher-level tools accelerate development in another way, too. FPGA tools work at the gate level, whereas FPOA tools work at the Silicon Object level. FPOA developers manipulate an object array of ALUs, MAC units, and register files, not arrays of gates. Whether this difference eases development, however, depends on the developer's point of view. MathStar prefers to compare FPOAs to FPGAs, not to general-purpose processors, DSPs, SoCs, or programmable ASICs. MathStar's comparison is fair, because, fundamentally, FPOAs are programmable-logic devices. If a developer's reference point is an FPGA, then programming an FPOA shouldn't seem especially difficult.

But as the cost of programmable logic falls, and as the cost of designing SoCs and ASICs rises, FPGA vendors are increasingly pitching their devices as alternatives to custom chips. Programming a general-purpose processor core or DSP is very different than programming an FPGA. The former is programmable in a high-level language like C or C++, or, at worst, in assembly language. The latter requires a hardware-design language (HDL)—a completely different skill. Although programming the Silicon Objects in an FPOA is a higher-level task than programming gate arrays, it too requires an HDL. Don't expect to port existing C or assembly-language code to an FPOA simply by recompiling the source. This is a job for hardware engineers, not for software programmers.

There's another catch—again, from the viewpoint of software coders. As mentioned before, MathStar's development tools inherently know the target FPOA's clock timing and signal-propagation characteristics. That's good knowledge, but a corollary is that HDL code compiled for one type of FPOA may not be portable to another type. One array may have a different number of Silicon Objects, or a different complement of objects, or different clock timings. MathStar says the HDL code will probably port to FPOAs having larger and faster arrays, but it's not guaranteed. Developers coming from the FPGA world—MathStar's target customers—probably won't consider this a problem. Developers accustomed to programming DSPs and general-purpose processors may be more intimidated.

No Need for Conventional Logic Synthesis

Despite those caveats, developing for an FPOA should be easier than developing for an FPGA. In addition to simplifying the problem of timing closure or eliminating it altogether, FPOAs eliminate the need for conventional logic synthesis—or, at least, the gate-level logic synthesis required for FPGAs.

In an FPOA, the logic is already "synthesized" in the form of Silicon Objects (which, as mentioned before, are actually custom-designed logic blocks, not synthesized models or standard cells). Developers work at a somewhat higher level of abstraction by programming the ready-made objects. Although this still requires HDL, it's one step removed from gate-level programming and synthesis.

As Figure 4 shows, eliminating conventional logic synthesis and some other steps related to gate-level design can significantly shorten the development cycle. MathStar's front-end design tool is Summit Visual Elite, an electronic system-level (ESL) tool from Summit Design, a MathStar partner. Using Visual Elite, developers can create a solution and a cycle-accurate simulation. Visual Elite also has co-simulation capabilities, supporting models written in Verilog, VHDL, and C. When targeting an FPOA, its compiled output is MathStar's own proprietary language, Object HDL, which resembles functional Verilog.

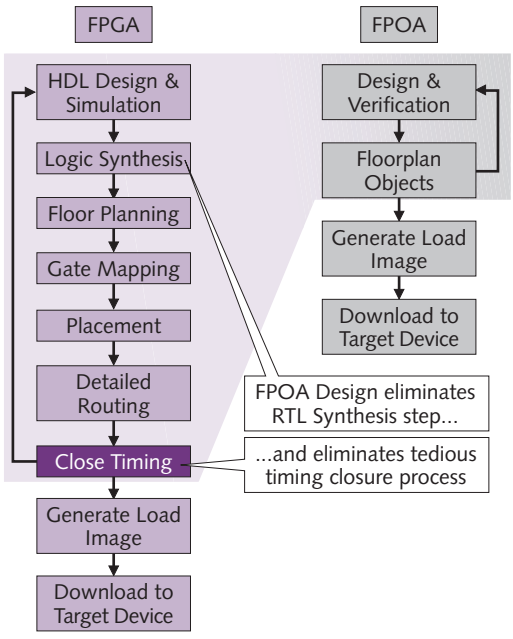


Figure 4. MathStar's development flow for FPOAs differs in two important respects from development for an FPGA. There's no need for conventional logic synthesis, because developers manipulate prefabricated Silicon Objects; and timing closure shouldn't be an issue, because an FPOA's fixed clock frequency is independent of the design's complexity.

The next tool in the FPOA chain is MathStar's COAST (Connection and Assignment Tool), which maps the HDL to the object array and defines the pathways through the interconnect fabric. Next, MathStar's Object Compiler converts the floor-plan map into an image suitable for downloading into the FPOA. Finally, MathStar's FPOA Debugger lets developers perform in-circuit verification and real-time debugging through the JTAG port. *MPR* has not worked with these tools, but MathStar's description seems reasonable, and the whole process appears to be more streamlined than conventional development on an FPGA.

Figure 5 shows how MathStar's floor-planning tools have mapped the logic for an MPEG-2 multistream video decoder onto the 20 × 20 object array of the MOA1400D. This design uses all but 80 of the chip's 400 Silicon Objects. The largest blocks of objects parse the data headers, handle motion compensation, implement a variable-length decoder (VLD), and accelerate inverse discrete cosign transforms (iDCT). Additional blocks augment the on-chip external memory controller and provide interfaces for video output and a host processor. This design is reconfigurable at run time, so it can decode up to four streams of standard-definition MPEG-2 video or one stream of high-definition MPEG-2 video.

The MPEG-2 decoder is part of MathStar's Professional Video Library, a package of licensable intellectual property (IP) cores for FPOAs. These "cores" are preconfigured designs for common applications, so developers won't have to reinvent the wheel. Another package is the Machine Vision Library, which includes a new JPEG2000 core capable of encoding 200 megapixels per second when running in a 1.0GHz MOA1400D. MathStar claims it's the fastest JPEG2000 encoder ever implemented in a single programmable-logic device. Other cores in the Machine Vision Library include a flat-field error-correction core, which is capable of processing 500 megapixels per second, and an RGB-to-YCC color-space converter, which can process up to one gigapixel per second. The company is working on additional cores.

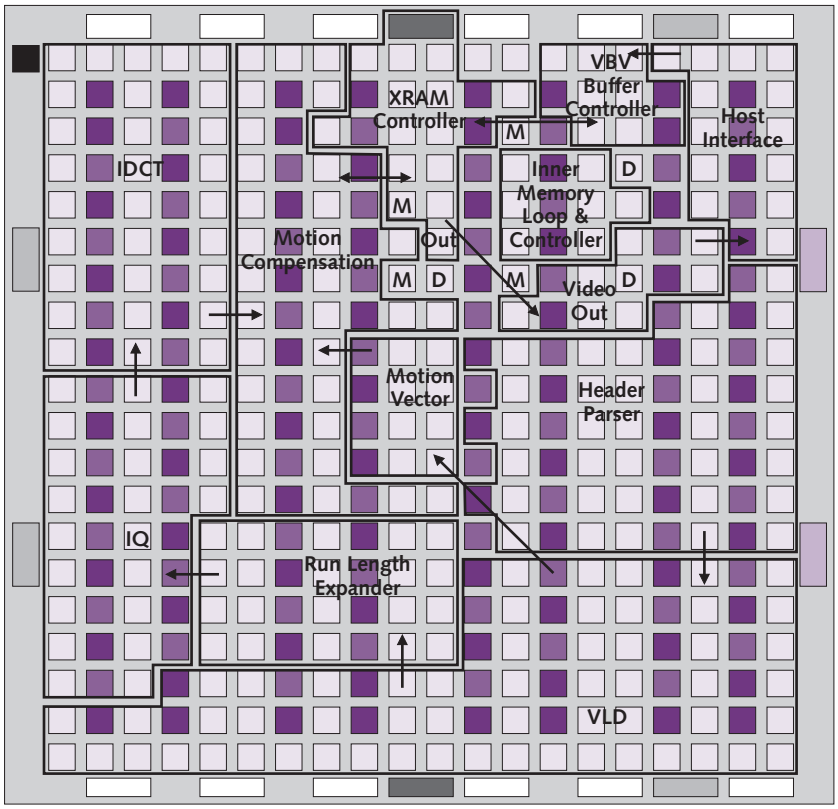


Figure 5. Development tools from MathStar and Summit Design help automate the process of configuring the massively parallel object array of an FPOA chip for the target application. In this example, the tools have mapped a multistream MPEG-2 decoder onto the array of the 1.0GHz MOA1400D processor, using 320 of the 400 available objects. (Some objects within the outlined blocks either aren't used or are reserved to add future features or more performance.)

Competitors: From A to X

FPOAs look like an attractive alternative to spinning custom silicon, but MathStar prefers to position itself against FPGAs. Indeed, MathStar envisions its technology as nothing less than the future of programmable logic. That lofty ambition pretty much narrows the list of competitors to Altera and Xilinx, which are virtually the alpha and omega of programmable logic.

Certainly, there are other players in this field, and some have interesting alternatives to conventional FPGAs, too. One is Actel, whose new Fusion chips combine flash-based programmable logic with integrated mixed-signal components, SRAM, and optional soft-processor cores. (See *MPR 12/19/05-02*, "Actel Releases First Fusion Chip.") Another example is Teja Technologies' FPGA Platform, a package of development tools, software, and hardware IP that allows software engineers to build a packet processor in an FPGA without using HDL. (See *MPR 4/3/06-02*, "Teja's FPGA Play.")

Massively parallel processors are even more plentiful, although they usually don't have reprogrammable logic. Some examples we have covered recently are Connex Technology's 1,024-element video chip (see *MPR 1/9/06-01*, "Massively Parallel Digital Video"); Elixent's flexible D-Fabrix architecture (see *MPR 6/27/05-02*, "Elixent Improves D-Fabrix"); Silicon Hive's configurable processor cores (see *MPR 6/20/05-01*, "Busy Bees at Silicon Hive"); and PicoChip Design's communications processors (see *MPR 10/14/03-03*, "PicoChip Makes a Big MAC"). One massively parallel architecture quite similar to MathStar's is the NEC Dynamic

Reconfigurable Processor (DRP), which can reconfigure its 512-processor array as often as every clock cycle. (See *MPR 11/25/02-04*, "New NEC Array Speeds Data.")

So MathStar is addressing two general markets. For the first market, MathStar offers reconfigurable logic instead of standard parts or custom chips, especially for systems not expected to reach high volumes. The second market consists of customers that need massively parallel processing arrays to run specialized tasks having a great deal of inherent instruction or data parallelism. The first market pits MathStar against FPGA vendors; the second pits MathStar's FPOA architecture against a diverse group of other extreme architectures.

Competing against giants like Altera and Xilinx seems quixotic, but MathStar argues that conventional FPGAs won't scale to higher clock frequencies and denser logic as well as FPOAs will. It remains to be seen how well MathStar's devices scale, but the fundamental argument makes sense. As FPGAs get faster and denser (thanks to advances in fabrication technology), developers will use them for more-complex applications. But complex algorithms force FPGAs to run at slower speeds than their maximum clock frequency, unlike FPOAs. In addition, complex gate-level development becomes more difficult, and gate utilization often decreases. The general trend in software development since the days of ENIAC has been toward higher levels of abstraction above the metal, and MathStar's array of Silicon Objects is a level above the gate arrays of conventional programmable-logic devices.

However, MathStar's object array requires developers to work at a lower level of abstraction than programmers using languages like C and C++, at least with MathStar's existing tools. Of course, other massively parallel architectures present the same challenge. To the extent that competitors can make their exotic architectures more easily programmable, MathStar will be at a disadvantage. *MPR* has covered many extreme architectures over the years, and their biggest burdens are always programmability and the steep learning curve of something new.

With a dozen design wins under its belt, MathStar obviously has some momentum. To win more converts, MathStar needs to release detailed case studies of those designs, keep improving its development tools, and establish a track record of shipping enhanced devices that prove FPOAs are indeed more scalable than FPGAs. ◇

Price & Availability

MathStar's first field-programmable object array (FPOA) intended for commercial production, the MOA1400D, has been sampling since August 2005. Volume production is scheduled for this October. Initial devices will be available at speeds of 400MHz, 800MHz, and 1.0GHz. The fastest part will cost $285 in 1,000-unit quantities; MathStar hasn't announced prices for the slower parts. MathStar has disclosed that the next-generation device will have the same-size array as the MOA1400D but better I/O and other features; a detailed announcement will probably come later this year. For more information about MathStar, visit *www.mathstar.com*.