

## **Multi-Processor Solutions with FPGAs**

**By**

**Bob Garrett**

**Senior Marketing Manager, Nios Marketing  
Altera Corporation**

Embedded designers seeking high performance processing inevitably face the cost/performance/power “Bermuda triangle” where the best of intentions can achieve any two of the key objectives, but fails to achieve all three. Custom ASIC designs are suitable for those few who can afford the time, expense, and risk involved, but as device geometries continue to shrink and ASIC design costs continue to grow, fewer and fewer applications can justify the expense of a full-custom design.

FPGA-based embedded systems featuring multiple soft-core processors offer a powerful set of new options for the embedded designer. No longer are ASIC designers alone in their ability to configure performance-optimized systems-on-chip with custom tailored feature set. Now developers can change the performance characteristics of their embedded system right up to the time the product goes into final test. Developers also can extend product life cycle, getting to market quickly and upgrading both software and hardware features remotely over the Internet.

While the term “multi-processor” may conjure up memories of academic papers on “parallel processing,” commercial applications of multiple CPUs in a single device are much more straightforward. When starting a new design, developers must meet certain performance criteria. Partitioning duties among multiple soft processors not only provides the design flexibility to adapt to last-minute design changes caused by evolving standards or competing products, but also the ability to keep pace with this performance criteria.

Multiple soft processors can be used as a divide-and-conquer strategy to increase overall system performance or offload tasks from an existing processor. Designers typically use 400-800 MHz discreet processors to perform the myriad device tasks required, both simple and demanding. Using multiple soft processors enables a more efficient use of processing power by partitioning tasks based on time and power requirements, while providing the same or better overall performance.

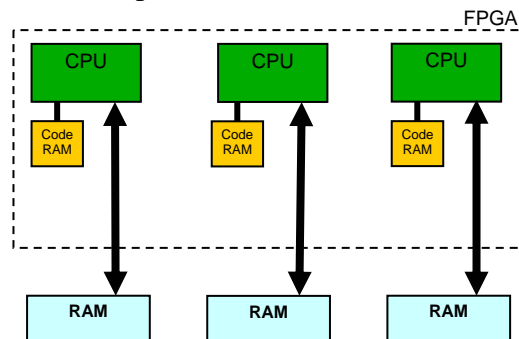
The number of soft-core processors that can be implemented in a single FPGA is limited only by the device’s resources (i.e., its logic and memory). High-density FPGAs, for example, can contain hundreds of soft-core processors. Likewise, different types of soft-core processors can be implemented, such as 16- or 32-bit, performance optimized, logic-area optimized, and so on.

The coding algorithm can be split among multiple processors, depending on the tasks involved. Time critical tasks can be assigned to dedicated processors, while less-demanding duties can be shared on one or more other CPUs. This flexibility enables logical grouping of tasks, and potentially higher performance levels while running at a reduced clock frequency reducing system power consumption.

### **EMBEDDED PROCESSORS IN FPGAs**

Building a custom device containing an exact set of peripherals, memory interfaces and processing functions is not hard to imagine: ASIC designers have been doing it for years. Efforts to create an economically viable custom FPGA-based embedded processor device were unsuccessful until the late 1990s when FPGAs had enough on-chip memory, programmable logic and raw performance. Today, embedded IP functions designed specifically for FPGAs—including CPUs, signal processing engines, peripherals and standard communications interfaces—are readily available and offer both cost and performance benefits over traditional discrete embedded devices.

Essentially, designers partition the problem the same way they might if they were building a multiprocessor system on a printed circuit board, each assigned to a specific task. For example, one processor might perform general system housekeeping such as monitoring cabinet fans, man-machine interface or the system console, while the others handle communications, signal processing, statistics gathering, or other system tasks.

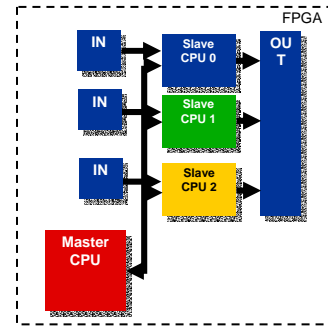


The multiple processor approach can reduce overall device cost by moving individual processors from the device board onto the FPGA, which decreases device board size. This also enables less signal routing between processors requiring fewer interconnects, and more low-level processors running at lower clock frequencies, which reduces layers on the circuit board.

This approach can also reduce software design costs, which represent 80 percent of overall system design expenses due to time-consuming code writing. If the task can be partitioned to multiple processors, it is easier for engineers to write, debug, and maintain codes. This potentially represents a huge backend savings, enabling faster code development and debugging and, when the product matures, easier code maintenance because it is much easier to analyze.

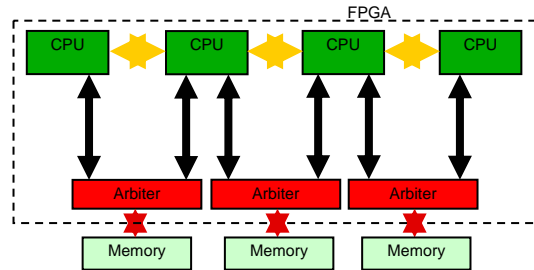
## Multi-Channel Applications

Multi-channel applications can be scaled to meet system throughput by using multiple processors in a single chip, each dedicated to handling a portion of the overall channel throughput. Each processor may run the exact same code, or some may change algorithms on the fly to adapt to system requirements. In some cases, a master processor is added to handle general housekeeping chores such as system initialization, statistics gathering, and error handling.



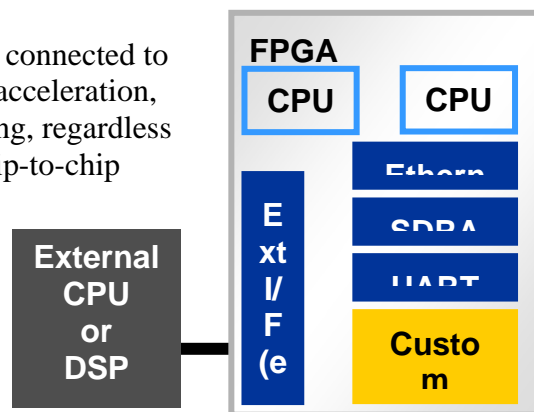
## Serially Linked Processors

Combining several processors in a chain lets system architects treat each as a stage in a larger processing pipeline. Each CPU, responsible for one piece of the overall processing task, can share data memory (arbitrated or dedicated memory interfaces if off-chip, or dual-ported memory if on-chip) to pass results from the output of one stage to the input of the next.



## Processor Companion Chip

Discrete processor and DSP chips connected to an FPGA can also benefit from hardware acceleration, peripheral expansion, and interface bridging, regardless of whether a CPU is inside the FPGA. Chip-to-chip interface IP is readily available today to provide external access to peripherals, acceleration logic, and I/O interfaces contained within the FPGA.



## ESTABLISHING PROCESSOR PERFORMANCE

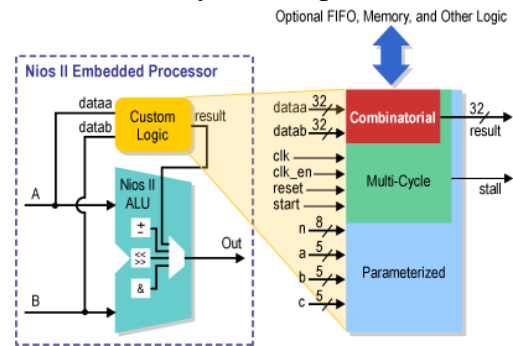
Establishing processor performance requirements for embedded systems can be challenging, particularly when the application software is still in flux. Industry-standard benchmarks provide some guidance, but nothing is certain until the software is complete. This tends to make designers cautious about under-calling their performance needs and may result in selecting a higher performance (and higher price) device than necessary. If a designer could accurately predict the

performance required, processor selection would be much simpler. Such estimates would consider performance required by time-critical tasks as well as the load created by one or more low-priority tasks.

FPGA-based embedded systems can provide scalable performance, allowing last-minute changes to boost system performance based on customer demands. Compute-intensive algorithms, converted to logic in an FPGA, can run orders of magnitudes faster than the same algorithm run in software by a microprocessor or digital signal processor. More importantly, hardware resources can be applied to performance-hungry algorithms where they are needed most, potentially reducing the need for a high-performance CPU, reducing clock frequency, reducing power consumption, and simplifying the board design.

## EXTENDING THE INSTRUCTION SET

The ability to extend a processor's instruction set to include application-specific algorithms implemented in hardware is offered by several processor IP vendors and in FPGA implementations. Using the processor's normal load / store operations, data can be fed to a custom logic block that essentially becomes a part of the processor's arithmetic logic unit (ALU). In some cases, custom instructions can support multi-cycle operation and access other system resources such as FIFOs and memory buffers. Typical applications of custom instructions include bit manipulation, complex numeric and logical operations.



While reliant on processor load and store operations, custom instructions can provide significant performance benefits over running the same algorithm using normal ALU resources. For example, a cyclic redundancy check (CRC) of a 64-KByte buffer executed 27 times faster when implemented as a custom instruction than when run in software<sup>1</sup>. Performance results vary from application to application, but in general are much faster than software alone.

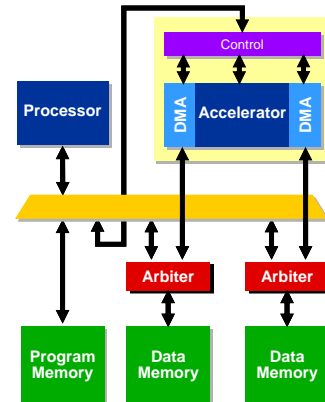
How the developer makes use of a custom instruction varies by processor IP vendor. On one end of the spectrum, a new compiler must be generated when adding custom instructions. This custom compiler then infers custom instruction calls based on some application criteria. A more simplistic approach relies on the user to call the instruction explicitly in their C code just as they would a subroutine. This approach seems more natural since the software designer will most likely know best when to use a custom instruction.

<sup>1</sup> Embedded Systems Programming magazine, February 2004. "Accelerating Algorithms in Hardware" by Lara Simsic

## ALTERNATIVE APPROACHES FOR BOOSTING SYSTEM PERFORMANCE

Other approaches for boosting overall system include utilizing hardware accelerators (a.k.a. co-processors), processor companion chips or custom systems on chips.

Unlike custom instructions, hardware accelerators operate as independent logic modules that take direction from the embedded CPU and process entire buffers of data without CPU intervention. A simple block diagram includes the processing module with two  $\frac{1}{2}$  DMA channels (one to read input data, the other to store the results), and a control interface for the CPU to set-up, start, stop, and poll the unit during operation. This architecture is much better suited to tasks that involve large blocks of data where the process of the CPU loading data and storing the results would become the performance bottleneck.



Hardware accelerators can achieve several orders of magnitude performance increase over tasks run in software due to their autonomous nature and the fact that the acceleration function is designed in hardware.

Discrete processor and DSP companion chips connected to an FPGA can benefit from hardware acceleration, peripheral expansion, and interface bridging, regardless of whether a CPU is inside the FPGA. Chip-to-chip interface IP is readily available today to provide external access to peripherals, acceleration logic, and I/O interfaces contained within the FPGA.

## CUSTOMIZING TO EXTEND PRODUCT LIFE

Bringing a product to market quickly often results in first-generation versions that ship with fewer features or less optimal performance than planned. The tradeoffs associated with delaying new product releases are usually not attractive. Upgrading system firmware (i.e., flash memory) to fix bugs, add features and/or improve performance is an established practice. Systems built with programmable logic add the capability to change hardware characteristics as well, since the same flash devices that contain system software can contain one or more FPGA configuration images. Flash devices containing system configurations are easily rewritten by a processor in the FPGA, after which it can initiate a reconfiguration that updates the system hardware in milliseconds.

Products that are network enabled now have the ability to remotely reconfigure the hardware to add new features, increase performance and fix bugs over the Internet. While a novelty just a few years ago, this has become a common practice with FPGA-based designs. Fail-safe schemes based on multiple FPGA images stored in flash memory with a default “safe” design that loads in

the case of data corruption ensures a working system device. The worst-case scenario, in which the device fails to configure, results in the default design being loaded instead, at which time it can report the failure and continue to operate as before.

## CONNECTING SYSTEMS

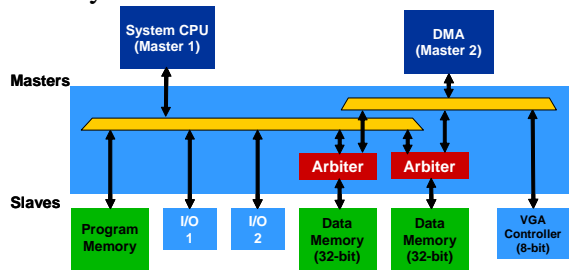
Processors must communicate with memory, peripherals, co-processors, and external devices. Software drivers utilize provided mailboxes to enable multiprocessors to communicate via shared memories.

Connecting several blocks of complex IP functions can pose challenges that, at first, may appear modest at first, but can impact system development time and performance. The list of logic functions required to “glue” the system together includes address decoders, data multiplexers, interrupt controllers, wait-state generators, arbitration logic, bus width sizing, signal timing and level matching, and clock domain crossing logic.

These IP functions are well within the capabilities of a good hardware designer, but hand-coding Verilog or VHDL for these (arguably) low-value IP functions seems a waste of good engineering talent. Routine tasks like these are perfect candidates for computer automation. Modern FPGA system-generation tools handle these chores seamlessly, letting designers focus on more important aspects of their project.

In traditional embedded systems, processors share a system bus with other “master” components that periodically request sole access to the shared bus. This interconnect topology creates a potential bottleneck when two masters need to access the bus at the same time. FPGAs are rich in routing resources, allowing masters to have dedicated multiplexed data paths to slave components.

Changing from master-side to slave-side arbitration allows simultaneous transactions to occur unless two masters try to access the same slave device at the same time. Internal switch fabrics control all connections between masters and slaves with different bus widths, master and slaves with different clock domains, and latency automatically between masters and slaves.



Using this switch-style topology allows all masters to communicate to their dedicated slave devices simultaneously, dramatically boosting overall system throughput.

## **SUMMARY**

Using multiple soft processors in FPGAs enables a more efficient use of processing power by partitioning tasks based on time and power requirements, while providing the same or better overall performance compared to discreet processors. Multiple soft processors also can be used in a divide-and-conquer strategy, limited only by the logic and memory resources of the targeted FPGA, to increase overall system performance or offload tasks from an existing processor.

Mainstream application of these techniques has been enabled by a new class of hardware development tools, Intellectual Property cores, and FPGA infrastructure. Development and growth in this area by major FPGA vendors continues at a fast pace due to the success of these technologies within the embedded community. It's time to have a look at what these possibilities could mean in your next application.