

Designing Flexible, High-Performance Embedded Systems

Soft processing and customizable IP enable the best of both worlds.

by Jay Gould
Product Marketing Manager
Xilinx, Inc.
jay.gould@xilinx.com

Which do you want in your next embedded project: flexible system elements so that you can easily customize your specific design, or extra performance headroom in case you need more horsepower in the development cycle? Why put yourself under undue development pressure and settle for one or the other? Soft processing and customizable IP offer the best of both worlds, integrating the concepts of custom design and co-processing performance acceleration.

Discrete processors offer a fixed selection of peripherals and some kind of performance ceiling capped by clocking frequency. Embedded FPGAs offer platforms upon which you can create a system

with a myriad of multiple customizable processor cores, flexible peripherals, and even co-processing offload engines. You now have the power to design an uncompromised custom processing system to satisfy the most aggressive project requirements and punch a hole in that performance ceiling, while maximizing system performance by implementing accelerated software instructions in the FPGA hardware. With FPGA fabric acceleration, the sky's the limit.

Flexibility

In addition to the high-performance PowerPC™ hard-processing core available in Xilinx® Virtex™ Platform FPGAs and the small footprint PicoBlaze™ microcontroller core programmed with assembly language, Xilinx offers you the choice of designing with a customizable

general-purpose 32-bit RISC processor. The MicroBlaze™ soft processor is highly flexible because it can be built out of the logic gates of any of the Virtex or Spartan™ families, and you can customize the processing IP peripherals to meet your exact requirements.

With customizable cores and IP, you create just the system elements you need without wasting silicon resources. When you build a processing system in a programmable device like an FPGA, you will not waste unused resources in a discrete device, nor will you run out of limited peripherals if you require more than what are offered (say your design requires three UARTs and your discrete device offers only one or two). Additionally, you are not trapped by your initial architecture assumptions; instead, you can continue to dramatically modify and tune your system

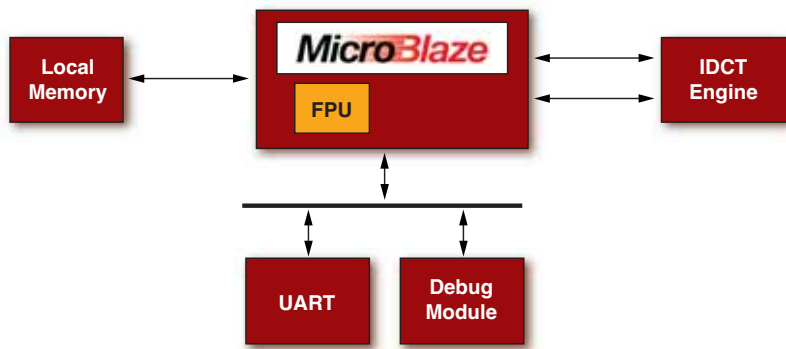


Figure 1 – Simple MicroBlaze block diagram

architecture and adapt to changes in newly required features or changing standards.

One FIR filter design example from the workshop materials for the 2006 Embedded System Conference had a MicroBlaze system configured with an optional internal IEEE 754-compliant floating-point unit (FPU), which facilitates a significant performance increase over software-only execution on the processor core. By including optional MicroBlaze components, you can quickly improve the performance of your application.

A side benefit of these optional internal components is that they are fully supported by the MicroBlaze C compiler, so source-code changes are unnecessary. In the FIR filter design example, including the FPU and recompiling the design meant immediate performance improvements, as calls to external C library floating-point functions are automatically replaced with instructions to use the new FPU.

Utilizing specialized hardware-processing components improves processor performance by reducing the number of cycles required to complete certain tasks by orders of magnitude over software recoding methods. The simple block diagram in Figure 1 represents a MicroBlaze processing system with an internal FPU IP core, local memory, and choice of other IP peripherals such as a UART or a JTAG debugging port. Because the system is customizable, we could very well have implemented multiple UARTs or other IP peripheral cores from the Xilinx processor IP catalog, including a DMA controller, IIC, CAN, or DDR memory interface.

The IP catalog provides a wide variety of other processing IP (bridges, arbiters, interrupt controllers, GPIO, timers, and memory controllers) as well as customization options for each IP core (baud rates and parity bits, for example) to optimize elements for feature, performance, and size/cost. Additionally, you can configure the processing cores with respect to clock frequency, debugging modules, local memory size, cache, and other options. By merely turning on the FPU core option, here at Xilinx we built a MicroBlaze system that optimized the aforementioned FIR implementation from 8.5 million CPU cycles to only 177,000 CPU cycles, enabling a per-

formance improvement of 48x with no changes to the C source file.

In a second example, we'll build on an additional design module, implementing an IDCT engine for an MP3 decoder application that will accelerate the application module by more than an order of magnitude.

You can easily create both processor platform examples referenced here with a development kit like the one depicted in Figure 2. The Integrated Hardware/Software Development Kit includes a Virtex-4 reference board that directly supports both PowerPC and MicroBlaze processor designs. The kit also includes all of the compiler and FPGA design tools required, as well as an IP catalog and pre-verified reference designs.

With the addition of a JTAG probe and system cables, the kit allows you to have a working system up and running right out of the box before you start editing and debugging your own design changes. Development kits for various devices and boards are available from Xilinx, our distributors, and third-party embedded partners.

Locate Bottlenecks and Implement Co-Processing

The MicroBlaze processor, one of EDN's Hot 100 Products of 2005, utilizes the IEC (International Engineering Consortium)



Figure 2 – Integrated Hardware/Software Development Kit

award-winning Xilinx Platform Studio (XPS) embedded tool suite for implementing the hardware/IP configuring and software development. XPS is included in our pre-configured Embedded Development Kits and is the integrated development environment (IDE) used for creating the system. If you have a common reference board or create your own board description file, then XPS can drive a design wizard to quickly configure your initial system.

Reduce errors and learning curves with intelligent tools so that you can focus your design time on adding value in the end application. After creating the basic configuration, you can spend your time iterating on the IP to customize your specific system and then develop your software applications.

XPS provides a powerful software development IDE based on the Eclipse framework for you power coders. This environment is ideal for developing, debugging, and profiling code to identify the performance bottlenecks that hide in otherwise invisible code execution. These inefficiencies in the code are often what makes a design miss its performance requirement goals, but they are hard to detect and often even harder to optimize.

Using techniques like “in-lining code” to reduce the overhead of excessive function calls, you can improve application performance 1%-5%. But with programmable platforms, more powerful design techniques now exist that can yield performance improvements by an order or two in magnitude.

Figure 3 shows a montage of Platform Studio views for performance analysis. XPS displays profiling information in a variety of forms so that you can identify trends or individual offending routines that spike on performance charts. Bar graphs, pie charts, and metric tables make it easy to locate and identify function and program inefficiencies so that you can take action to improve those routines that leverage the most benefit for total system performance.

Soft-Processor Cores with Their Own IP Blocks

For the MP3 decode example that I described earlier, we built a custom system (Figure 4), starting with the instantiation of multiple MicroBlaze processors. Because the

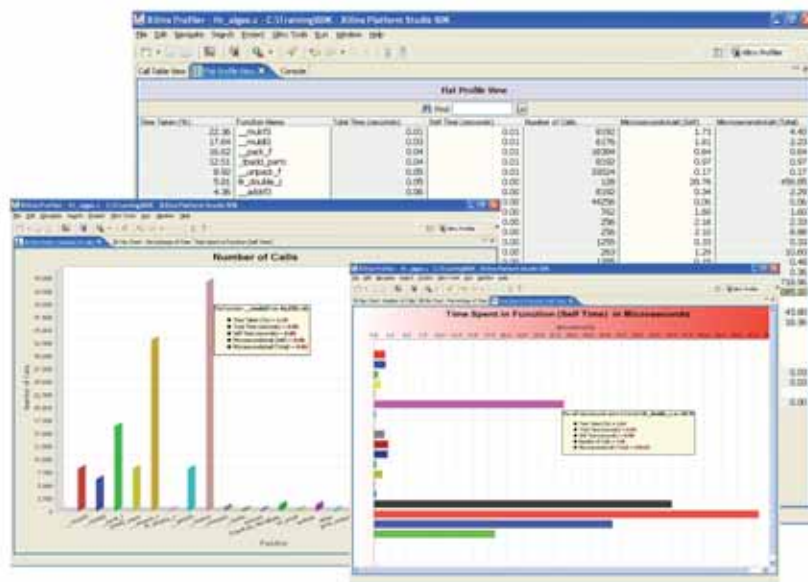


Figure 3 – Platform Studio embedded tool suite

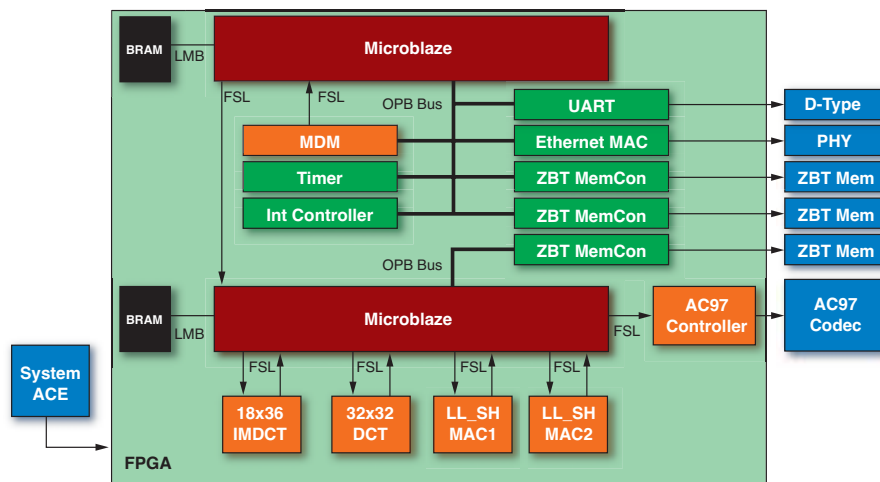


Figure 4 – MicroBlaze MP3 decoder example

MicroBlaze processor is a soft-core processor, we can easily build a system with more than one processor and balance the performance loading to yield an optimal system.

In Figure 4 you can clearly see the top MicroBlaze block with its own bus and peripheral set separate from the bottom MicroBlaze block and its own, different peripheral set. The top section of the design runs embedded Linux as an OS with full file system support, enabling access to MP3 bitstreams from a network. We offloaded the decoding and playing of

these bitstreams to a second MicroBlaze processor design, where we added tightly coupled processor offload engines for the DCT/IMDCT (forward and inverse modified discrete cosine transform) functions and two high-precision MAC units.

The IMDCT block covers data compression and decompression to reduce transmission-line execution time. DCT/IMDCT are two of the most computationally intense functions in compression applications, so moving this whole function to its own co-processing block greatly

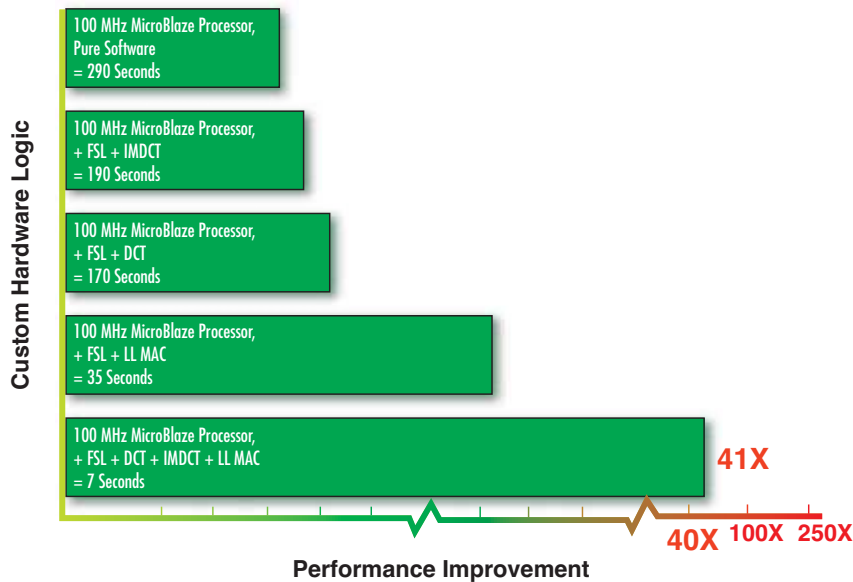


Figure 5 – Co-Processing acceleration results

improves overall system performance. Whereas we implemented an internal FPU in the earlier FIR filter example, the MP3 example has implemented MicroBlaze customizations and added external dedicated hardware within the FPGA.

Co-Processing + Customizable IP = Performance

By offloading computationally intense software functions to co-processing “hard instructions,” you can develop an optimal balance for maximizing system performance. Figure 4 also shows a number of IP peripherals for the Linux file system module, including UART, Ethernet MAC, and many other memory controller options. The coder/decoder application block, by comparison, uses different IP customizable for different system capabilities.

The second MicroBlaze soft core is slaved to the first MicroBlaze processor and acts as a task engine decoding the MP3 bitstream. The decoder algorithm with the addition of specific IP cores is connected directly in the FPGA fabric hardware resources through a Xilinx Fast Simplex Link (FSL) connection interface. This co-processing design technique takes advantage of the parallel and high-speed nature of FPGA hardware compared to the slower, sequential execution of instructions of a stand-alone processor.

Direct linkage to the high-performance FPGA fabric introduces fast multiply accumulate modules (LL_SH MAC1 and LL_SH MAC2 on Figure 4) to complement dedicated IP for the DCT and IMDCT blocks. The long-long MAC modules provide higher precision while offloading the processing unit. You will also notice high-speed FSL links utilized for the AC97 controller core to interface to an external AC 97 codec, allowing the implementation of CD-quality audio input/output for the MP3 player.

The co-processing system depicted in Figure 4 results in a cumulative 41x performance acceleration over the original software application with the additive series of component boosts. Comparing a pure “software-only” implementation (see the top horizontal bar as illustrated in Figure 5) to each subsequent stage of the hardware instruction instantiations, you can see how the performance improvements add up. Moving software into the IMDCT alone yields a 1.5x improvement, while adding the DCT as a hardware instruction moves it up to 1.7x. Another larger improvement is realized by implementing a long-long multiply accumulate to reach 8.2x.

Implementing all of the software modules in hardware through co-processing techniques, the total end result rolls up to

yield an amazing 41x improvement – with the added benefit of reducing the application code size. Because we removed multiple functions requiring a large number of instructions and replaced them with a single instruction to read or write the FSL port, we have far fewer instructions and thus achieved some code compaction. In the MP3 application section, for example, we saw a 20% reduction in the code footprint.

Best of all, design changes through intelligent tools like Platform Studio are easy, quick, and can still be implemented well into the product development cycle. Software-only methods of performance improvement are time-consuming and usually have a limited return on investment. Balancing the partition of software application, hardware implementation, and co-processing in a programmable platform, you can attain much more optimal results.

Conclusion

Based on the examples described in this article, we were able to easily customize a full embedded processing system, edit the IP for the optimal balance of feature/size/cost, and additionally squeeze out huge performance gains where none appeared possible. The Virtex-4 and Spartan-3 device families offer flexible soft-processor solution options that can be designed and refined late into the development cycle. The award-winning MicoBlaze soft-processor core combined with the award-winning Platform Studio tool suite provides a powerful combination to kick-start your embedded designs.

Co-processing techniques, such as implementing computationally intense software algorithms as high-performance FPGA hardware instructions, allow you to accelerate your performance of modules by 2x, 10x, or as much as 40x+ in our common industry example. Imagine what it can do for your next design – think about the head room and flexibility available for your design late in the development cycle, or being able to proactively plan improvements to the next generation of your product.

For more information on Xilinx embedded processing solutions, visit www.xilinx.com/processor.