

MicroBlaze and PowerPC Cores as Hardware Test Generators

Combining FPGA embedded processors with C-to-RTL compilation can accelerate the testing of complex hardware modules.

by David Pellerin
CTO

Impulse Accelerated Technologies
david.pellerin@impulsec.com

Milan Saini

Technical Marketing Manager
Xilinx, Inc.
milan.saini@xilinx.com

Regardless of whether you are using a processor core in your FPGA design, using a Xilinx® MicroBlaze™ or IBM™ PowerPC™ embedded processor can accelerate unit testing and debugging of many types of FPGA-based application components.

C code running on an embedded processor can act as an in-system software/hardware test bench, providing test inputs to the FPGA, validating the results, and obtaining performance numbers. In this role, the embedded processor acts as a vehicle for in-system FPGA verification and as a complement to hardware simulation.

By extending this approach to include not only C compilation to the embedded processor but C-to-hardware compilation as well, it is possible – with minimal effort

– to create high-performance, mixed software/hardware test benches that closely model real-world conditions.

Key to this approach are high-performance standardized interfaces between test software (C-language test benches) running on the embedded processor and other components (including the hardware under test) implemented in the FPGA fabric. These interfaces take advantage of communication channels available in the target platform.

For example, the MicroBlaze soft-core processor has access to a high-speed serial interface called the Fast Simplex Link, or FSL. The FSL is an on-chip interconnect feature that provides a high-performance data channel between the MicroBlaze processor and the surrounding FPGA fabric.

Similarly, the PowerPC hard processor core, as implemented in Virtex-II Pro™ and Virtex-4™ FPGAs, provides high-performance communication channels through the processor local bus (PLB) and on-chip memory (OCM) interfaces, as illustrated in Figure 1.

Using these Xilinx-provided interfaces to define an in-system unit test allows you to quickly verify critical components of a larger application. Unlike system tests (which model

real-world conditions of the entire application), a unit test allows you to focus on potential trouble spots for a given component, such as boundary conditions and corner cases, that might be difficult or impossible to test from a system-level perspective. Such unit testing improves the quality and robustness of the application as a whole.

Unit Testing

A comprehensive hardware/software testing strategy includes many types of tests, including the previously-described unit tests, for all critical modules in an application. Traditionally, system designers and FPGA application developers have used HDL simulators for this purpose.

Using simulators, the FPGA designer creates test benches that will exercise specific modules by providing stimulus (test vectors or their equivalents) and verifying the resulting outputs. For algorithms that process large quantities of data, such testing methods can result in very long simulation times, or may not adequately emulate real-world conditions. Adding an in-system prototype test environment bolsters simulation-based verification and inserts more complex real-world testing scenarios.

Unit testing is most effective when it focuses on unexpected or boundary conditions that might be difficult to generate when testing at the system level. For example, in an image processing application that performs multiple convolutions in sequence, you may want to focus your efforts on one specific filter by testing pixel combinations that are outside the scope of what the filter would normally encounter in a typical image.

CoDeveloper generates FPGA hardware from the C-language software processes and automatically generates software-to-hardware and hardware-to-software interfaces. You can optimize these generated interfaces for the MicroBlaze processor and its FSL interface or the PowerPC and its PLB interface. Other approaches to data movement, including shared memories, are also supported.

hardware modules – to be described and interconnected using buffered communication channels called streams.

Impulse C also supports communication through signals and shared memories, which are useful for testing hardware processes that must access external or static data such as coefficients.

Data Throughput and Processor Selection

When evaluating processors for in-system testing, you must first consider the fact that the MicroBlaze processor or any other soft processor requires a certain amount of area in the target FPGA device. If you are only using the MicroBlaze processor as a test generator for a relatively small element of your complete application, this added resource usage may be of no concern. If, however, the unit under test already pushes the limits in the FPGA, you may want to target a bigger device during the testing phase or consider the PowerPC core provided in the Virtex-II Pro and Virtex-4 platforms as an alternative.

Synthesis time can also be a factor. Depending on the synthesis tool you use, adding a MicroBlaze core to your complete application may add substantially to the time required to synthesize and map the application to the FPGA, which can be a factor if you are performing iterative compile, test, and debug operations.

Again, the PowerPC core, being a hard core that does not require synthesis, has an advantage over the MicroBlaze core when design iteration times are a concern. The 16 KB of data cache and 16 KB of instructions cache available in the PowerPC 405 processor also makes it possible to run small test programs entirely within cache memory, thereby increasing the performance of the test application.

If a high test data rate (the throughput from the processor to the FPGA) is your primary concern, using the MicroBlaze core with the FSL bus or the PowerPC with its on-chip-memory (OCM) interface will provide the highest possible performance for streaming data between software and hardware components.

By using CoDeveloper and the Impulse C libraries, you can make use of multiple

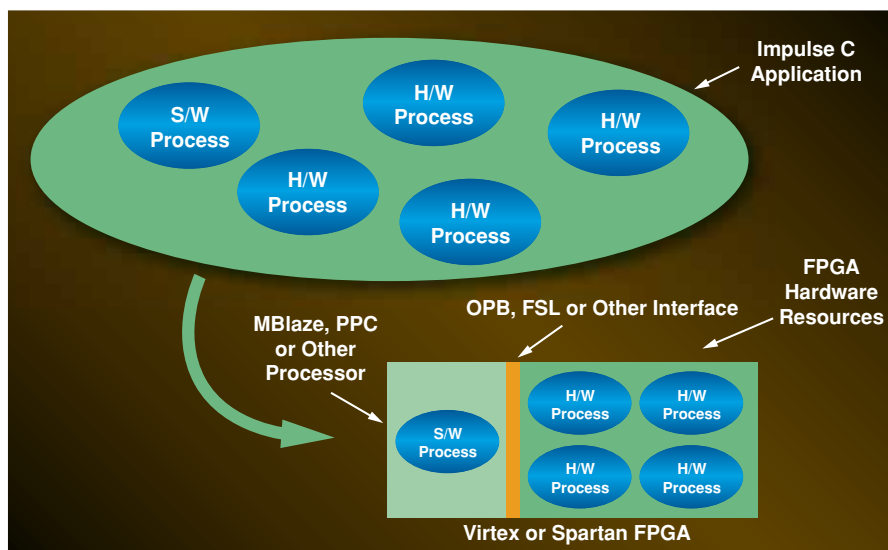


Figure 1 – Hardware and software test components are mapped to the FPGA target and communicate across the OPB or FSL bus to the MicroBlaze or Power PC processor.

It may be impossible to test all permutations from the system perspective, so the unit test lets you build a suite to test specific areas of interest or test only the boundary/corner cases. Performing these tests with actual hardware (which may for testing purposes be running at slower than usual clock rates) obtains real, quantifiable performance numbers for specific application components.

Introducing C-to-RTL compilation into the testing strategy can be an effective way to increase testing productivity. For example, to quickly generate mixed software/hardware test routines that run on the both the embedded processor and in dedicated hardware, you can use tools such as CoDeveloper (available from Impulse Accelerated Technologies) to create prototype hardware and custom test generation hardware that operates within the FPGA to generate sample inputs and validate test outputs.

Desktop Simulation and Modeling Using C

Using C language for hardware unit testing lets you create software/hardware models (for the purpose of algorithm debugging) in software, using Microsoft™ Visual Studio™, GCC/GBD, or similar C development and debugging environments. For the purpose of desktop simulation, the complete application – the unit under test, the producer and consumer test functions, and any other needed test bench elements – is described using C, compiled under a standard desktop compiler, and executed.

Although you can do this using SystemC, the complexity of SystemC libraries (in particular their support for data-flow abstractions through channels) makes the process of creating such test benches somewhat complex. CoDeveloper's Impulse C libraries take a simpler approach, providing a set of functions that allow multiple C processes – representing parallel software or

