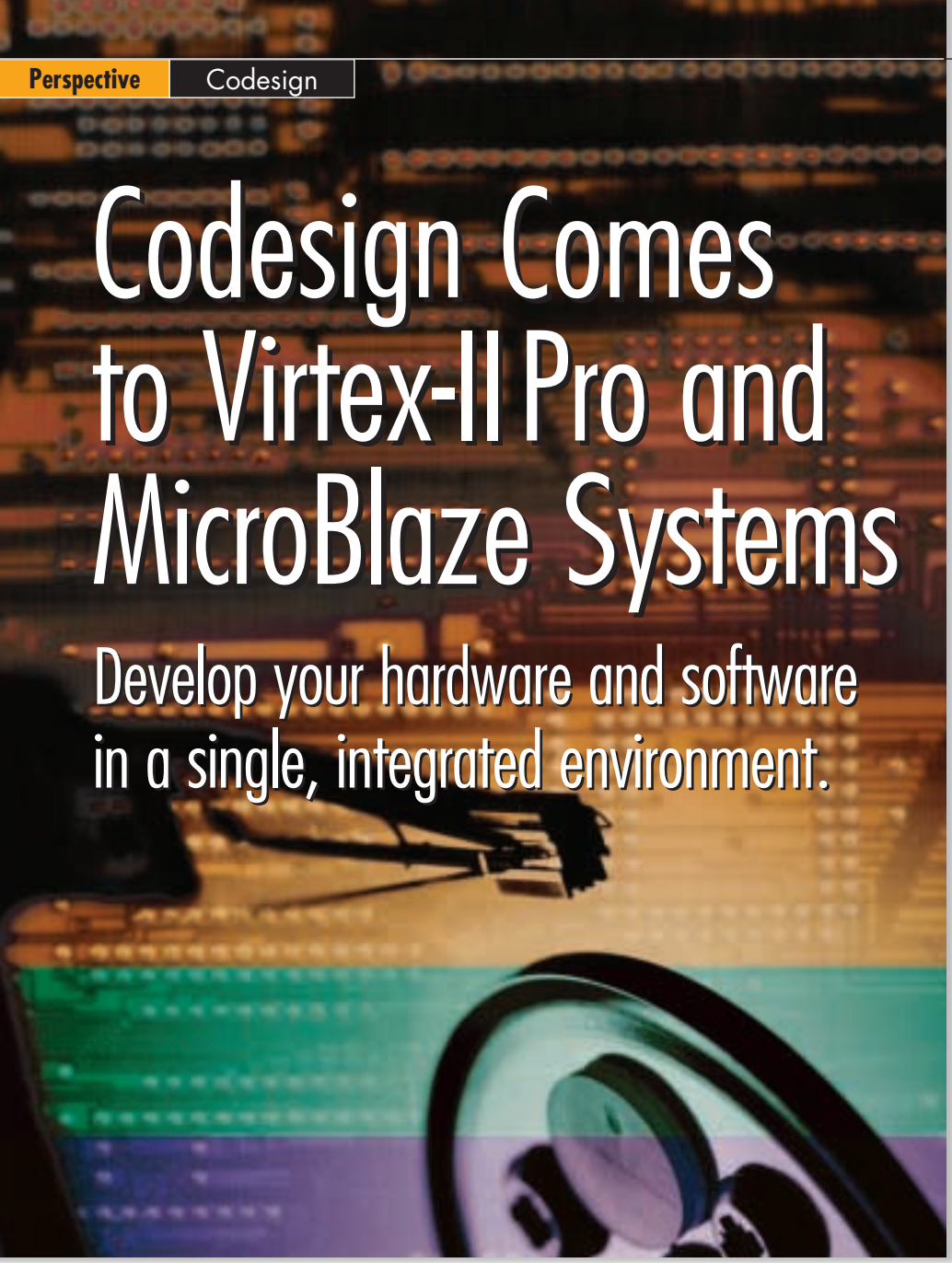

Codesign Comes to Virtex-II Pro and MicroBlaze Systems

Develop your hardware and software in a single, integrated environment.

by Chris Sullivan
Director of Strategic Alliances
Celoxica
chris.sullivan@celoxica.com

Virtex-II Pro™ FPGAs are powerful system-level devices, replacing microprocessors and ASICs in many new applications. This shift in design strategies necessitates a corresponding shift in the way programmable logic designs are created and deployed in electronic products. To efficiently manage your software and hardware design in these programmable systems, you must now move away from legacy ASIC design methods to a codesign methodology that gives you greater choice in the level of design abstraction.

Codesign

Codesign is a process in which you use similar methods, and sets of connected tools and languages, for both hardware and software design. Codesign helps shorten development time by enabling the concurrent development of hardware and software, and by allowing software to be developed on "virtual hardware platforms" before the final hardware is ready. In addition, a top-down approach enhances your ability to analyze and tackle system partitioning and verification by enabling you to explore the design space fully. This enables more informed consideration of hardware/software trade-offs and leads to better Quality of Design (QoD). Reducing the risks that arise from incorrect or changing specifications can help avoid the time-consuming and expensive optimization of an incorrect partition (which leads inevitably to a sub-optimal design) and increases your chances of first-time success.

Programmable Systems Require a Codesign Methodology

Historically, FPGA hardware was designed using techniques and languages borrowed from ASIC design methods – methods that are very different from those used to develop software or embedded systems. Up to now, there was a huge difference between these disciplines and their methodologies.

For example, current methods for embedded systems design require that hardware and software be specified and designed separately. Typically, C/C++ or a block-based methodology is used for the system specification. Once behavior has been fixed, the specification is then delegated to the (separate) hardware and software engineering teams, which code in different languages: HDLs (Hardware Description Languages) for the hardware, C/C++ for the software. While the system partition can be informed by profiling the specification or legacy software code, the partitioning is often decided in advance. And, because changes to the partition can necessitate extensive redesign elsewhere in the system (interfaces between the hardware and software, for example), that decision is adhered to as much as possible. The deficiencies of this methodology are clear:

- Lack of a unified hardware-software representation can lead to difficulties in verification of the entire system, and hence to incompatibilities across the hardware/software boundary.
- Defining a system partition in advance can lead to sub-optimal designs; incorrect partitioning requires costly refinement and is detrimental to QoD.
- Hardware partitions of the system specification or legacy software code require time-consuming (and sometimes error-prone) rewriting into HDL.

- Lack of a well-defined and flexible codesign methodology makes specification revision difficult and affects time to market.

While it is not yet possible to synthesize *efficient* hardware and software from a single language description, a codesign methodology that supports partitioning and co-verification, multiple languages, and tool interoperability is nevertheless invaluable when designing high-performance systems using Virtex II Pro FPGAs and MicroBlaze™ processors. Such a methodology makes it possible to:

- Prototype the system more easily and explore the design space better to identify the optimal design solution.
- Use generic hardware/software interfaces for system co-simulation and verification, using the software code as a testbench throughout.
- Implement changes to partitioning decisions – if required – much later in the design cycle.
- Target different hardware platforms more easily and even change the target platform later in the design cycle than would otherwise be possible.
- Drive system implementation from correct levels of abstraction.

The benefits of fusing separate design approaches into an effective and more "integrated software-compiled system design" flow that uses top-down design to tackle system partition, verification, and implementation are significant.

Working together, Celoxica and its strategic partners such as Wind River and Xilinx have developed a unique codesign flow and methodology (Figure 1) for Virtex-II Pro systems using MicroBlaze processors.

Software-Compiled System Design for Programmable Systems

Fundamental principles of the codesign methodology are:

- A top-down, idea-to-implementation flow
- A common higher-level language base for hardware and software design
- The distinction of processing fabric at correct levels of abstraction
- Interoperability with best-in-class hardware and embedded software tools
- Codesign API standards (for example, the DSM – Data Streaming Manager), which enable easy interfacing between software and hardware for partitioning, verification, and implementation.

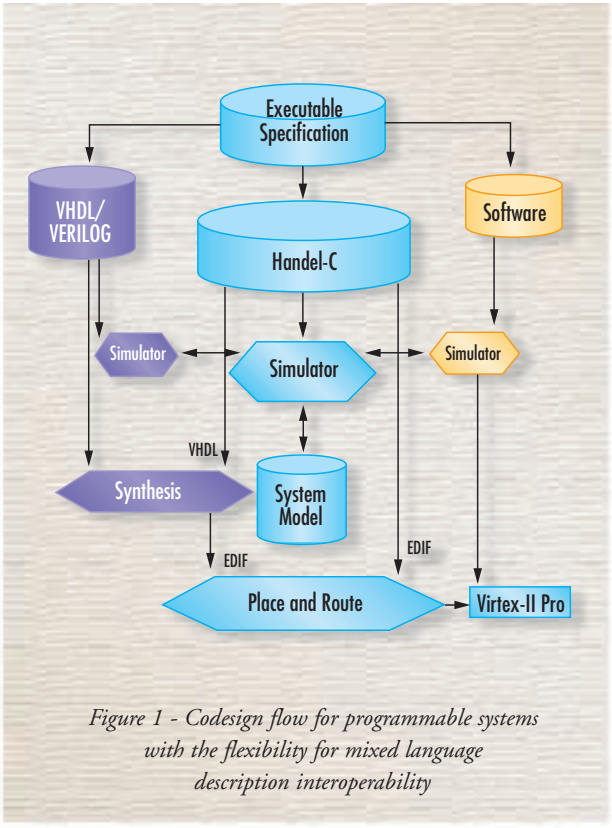

Figure 1 - Codesign flow for programmable systems with the flexibility for mixed language description interoperability

To make software-compiled system design possible, you need an environment that brings together the efficiencies of higher-level languages and the capabilities of powerful partition, verification, and design implementation.

DK Design Suite

The DK Design Suite enables you to enter system descriptions in higher-level programming languages, and to simulate and debug that code using a familiar, friendly integrated development environment (IDE). Block-based design and multiple languages are supported for simulation including C, C++, SystemC, HDLs, and Handel-C.

The package includes the Nexus-PDK co-verification environment, which also makes it possible to drive the entire functional verification process for the system with higher-level code.

Nexus PDK

Nexus-PDK is a powerful co-verification tool that allows you to simulate system functionality in multiple higher-level languages, and to continue to use these models through to design implementation by supporting co-simulation of software and hardware. Nexus communicates directly during simulation with popular third-party hardware RTL simulators and software ISS environments.

Handel-C

Handel-C, which is based on ANSI-C, has an added set of simple extensions for hardware development. These include:

- Flexible data widths
- Parallel processing
- Communication between parallel threads.

In addition, Handel-C uses a simple timing model that enables you to control pipelining without adding definitions for specific hardware. Handel-C also eliminates the need to code finite state machines exhaustively by providing the ability to describe serial and parallel execution flows.

Its familiar language has formal semantics for describing system functionality and complex algorithms that produce substantially shorter and more readable code than RTL-based representations. The level of design abstraction is above RTL (Register Transfer Level) but below the behavioral level, and everything that can be described in the language may be translated to hardware.

DSM

DSM (Figure 2) is a portable hardware-software codesign API that offers a simple and transparent interface for transferring multiple independent streams of data between hardware and software. DSM is independent of both bus/interconnects and operating systems. It consists of two parts: an OS-independent API for the FPGA application, and an API for ANSI-C or the software environment. In operation, each side opens a number of uni-directional ports; a "write to a port" on one side is then matched by a "read" on the other. In this way, multiple software applications can independently access multiple reconfigurable hardware resources using very few API calls.

In Figure 3 you can see how these solutions integrate with best-in-class embedded software tools from Wind River and Xilinx programmable systems to deliver a comprehensive software-compiled system design methodology.

The key elements of the methodology are:

- A minimal tool chain – comprising the Celoxica DK design suite, Wind River's XE (Xilinx Edition) embedded software tools, and Place and Route from Xilinx.
- A common language base – C and Handel-C, with the flexibility for interoperability with mixed language descriptions, such as HDLs and SystemC.
- API standards for common interfacing and platform abstraction – Celoxica PAL for platform abstraction, and Celoxica DSM for hardware/software integration.

Profiling and Partitioning

Profiling and partitioning are key to any codesign methodology and help identify optimal design methods early in the design cycle. In the software world, the profiler is mostly used as an analysis tool to examine the runtime behavior of a program. Profiler information helps you determine which sections of code are working efficiently and which are not. Profiling also gives you information about where the program spent its time, and which functions called which other functions while it was executing. In this way, profiling shows which pieces of the program are slower than expected and thus might be candidates for off-loading into hardware for coprocessor acceleration. It can also highlight which functions are being called more – or less often – than expected.

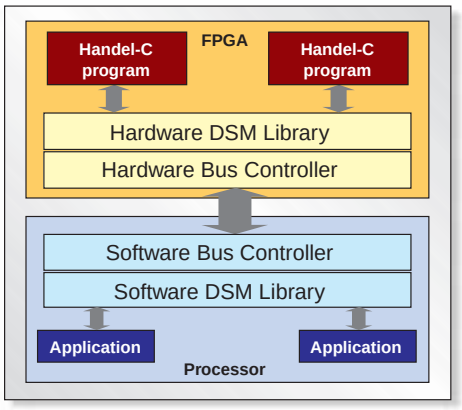

Figure 2 - DSM system overview

But profiling tools were developed to fine-tune software – making applications run better and identifying candidates for rewriting – not for system partitioning. Although profiling code is an extremely useful exercise for informing partitioning decisions, it should not be relied upon exclusively. For example, due to latency between the system boundary and interfaces, it makes sense to minimize dataflow between the hardware and software. And yet, software profiling tools do not explore dataflow over the hardware/software boundary. You can, however, deduce this dataflow through designer scrutiny of the code and by hardware/software coverification using API calls for run-time test.

To see how software-compiled system design can best be deployed for Virtex-II Pro FPGAs and MicroBlaze processors, let's use a simple design example within the context of codesign.

Codesign Methodology Design Example

In this example, we have a system that contains a GUI, an image compression engine, an encryption engine, and a control path through which we issue commands to the image compressor (Figure 4).

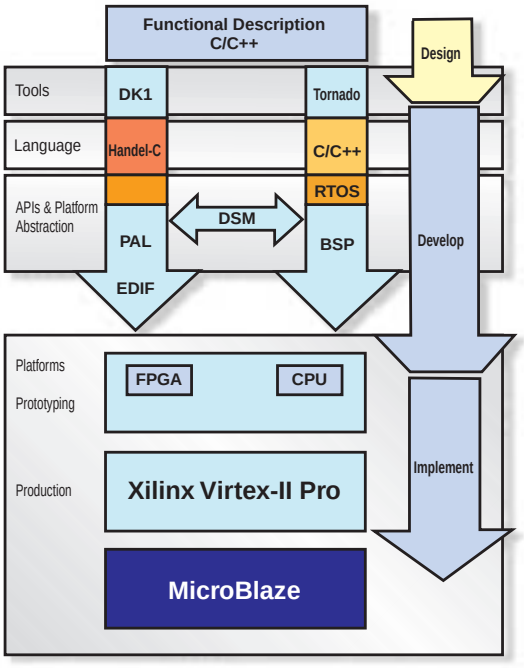

Figure 3 - Example HLL tool-chain

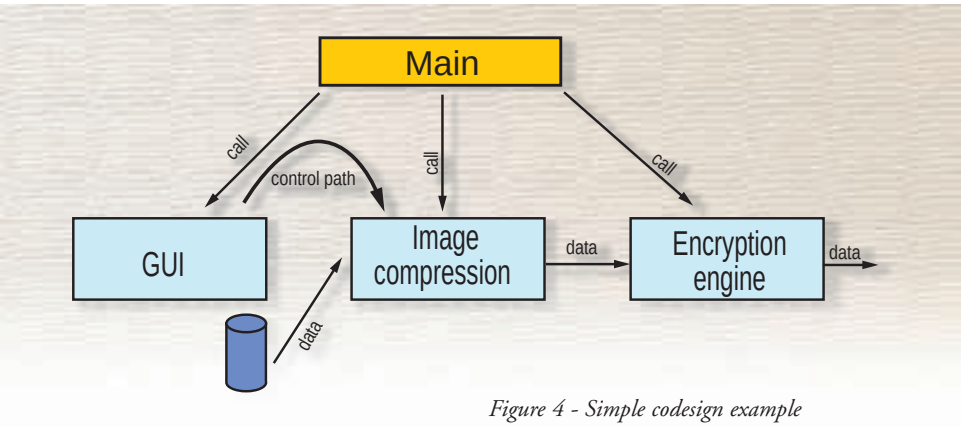

Figure 4 - Simple codesign example

1. First, we examine the system functionality against the project requirements, identify obvious system partitions, and also identify functions that will require further design investigation (such as those functions for which the optimum design partition is not immediately apparent).

   The GUI is an obvious candidate for software implementation; it is sequential and does not require processor-intensive resources. Likewise, the encryption engine is also a candidate for hardware implementation; it is parallel and integer-based. The partitioning of the compression function, however, is less clear and is targeted for profiling, iterative partition, and design exploration.

2. We move the compression function into software and obtain benchmarking information to provide a baseline for partition assessment. The software code can be used as a test bench throughout to support verification.

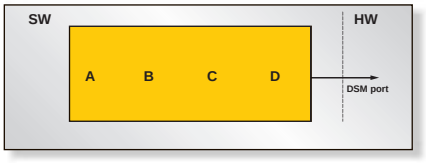

Figure 5 - DSM API port for hardware interface

3. With the function still in software, we use the DSM API to interface to the hardware component (Figures 5 and 6). We then begin to port blocks of the software to Handel-C for hardware prototyping, testing, and verifying at each stage. This process is relatively simple, because there is a common language base and, most importantly, a common level of abstraction for the software and the hardware. We also move the DSM port to enable the new partition to continue testing and verification at each stage (Figure 7).

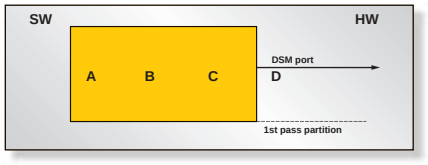

Figure 7 - DSM API port moved for new partition

4. Having completed the partition and debugging, we cosimulate to verify the effectiveness and efficiency of the partition, as measured against system constraints and design requirements.

5. We now enter what is effectively the partitioning cycle, in which we begin to reiterate and explore different partitions and design scenarios through testing and verification, using the simple procedure outlined in steps 3 and 4. This is an innovative process-driven approach to partitioning.

Hardware

```
// buffer to receive compressed data
ram DsmWord Buffer[ 256] ;

static unsigned DataCounter=0;

while(1) // loop forever
{
  do
  {
      // get output from SW
compression
      DsmRead (PortS2H, &Value);
      par
      {
          Buffer[ DataCounter]  = Value;
          DataCounter++;
      }

  } while(DataCounter!=0);


  // now encrypt the block of
data....
  EncryptData(Buffer);
}
```

Software

```
// buffer storing raw image
unsigned Image[ 1600][ 1200] ;

// buffer for compressed data (FIFO)
DsmWord CompData [ 256] ;

unsigned DataCounter,Count,ImageDone

do
{
// compress part of image (256 bytes output)
  CompressBlock(Image,CompData, ImageDone;

  DsmWrite (PortS2H, CompData, 256, &Count);

  if (Count!=256)
     printf("\n Error writing to HW");

} while(ImageDone==0);
  // loop till the end of the image
```

Figure 6 - Sample code showing DSM calls

6. The partitioning cycle produces a number of partition alternatives. We now consider these alternatives, map them to our design requirements or system constraints (such as device size, target platform, bandwidth, and so on), and select the optimum partition for QoD.

7. We simulate and verify the partitioned system, using compiled C/C++ combined with the Handel-C compiled for the Nexus PDK simulator. For speed and efficiency, the cosimulation uses DSM Sim and PAL (Platform Abstraction Layer) Sim as virtual interconnects and virtual peripherals, respectively.

8. The system is cosimulated and verified at a cycle-accurate level, using Nexus PDK, combined either with an ISS (Instruction Set Simulator) or ModelSim running a Swift model of the target processor.

9. We recompile the system for the target platform and implement the design. The target platform is supported by DSM and by a PAL layer that provides a portable API for accessing on-board peripherals, such as RAM, video, and generic data I/O. Thus, the application written using PAL and DSM APIs can be ported to new platforms simply by recompiling. This supports design reuse and application portability, and helps address the issue of design obsolescence.

Conclusion

According to Gary Smith, Dataquest's chief electronic design automation analyst, "Today the biggest challenge in EDA is to resolve the incompatibility of the hardware design methodology and the software design methodology." Software-compiled system design delivers an advanced methodology that offers significant advantages to hardware engineers, embedded software engineers, firmware engineers, and systems architects.

For more information see *www.celoxica.com* or contact *chris.sullivan@celoxica.com.* Σ

[1] P. Garrault, Synthesis Tool Enhancements for Virtex Architectures, Xilinx, 2002.

[2] Hardware/Software Co-Design Group, Polis A Framework for hardware-software co-design of embedded systems, EECS, University of California, Berkeley.