Software-Compiled System Design: A Methodology for Field-Programmable System-on-Chip Design

Jeff Jussel and Chris Sullivan

Celoxica Ltd., Abingdon, Oxfordshire, UK

Abstract

With the proliferation of Field-Programmable System-on-Chip (FPSoC) devices such as Altera Excalibur and Xilinx Virtex II Pro, more system integrators are now facing the challenge of merging hardware and software design. This paper describes software-compiled system design, a methodology supporting the design of electronics containing both programmable logic and software-driven microprocessors. To facilitate hardware and software design convergence, this methodology includes C-based design descriptions, hardware/software co-design partitioning and analysis, multi-level and multi-language co-verification, and direct compilation to programmable logic. The paper presents a practical implementation of this methodology using the Celoxica DK Design Suite of tools as well as a design example employing a JPEG2000 algorithm.

1 Introduction

As FPGAs have developed from logic prototyping devices into fundamental system elements, there has been enthusiasm for the concept of using high-performance processors, closely coupled to – or immersed inside – the FPGA fabric to create a Field Programmable System-on-Chip (FPSoC).

In the FPSoC architecture, the microprocessor typically runs system applications while the FPGA manages computationally intensive tasks. The availability of both processor and the equivalent of millions of gates of programmable logic give designers new opportunities to quickly develop, prototype and implement complete systems, delivering both better performance and flexibility.

As FPSoC entails hardware *and* software design, these complete system devices also present designers with all the challenges of system design. In this paper we will explore these design challenges, and present a description of a methodology created to address those challenges, Software-Compiled System Design. The paper also presents a system design example implementing a complex design example on an FPSoC device as validation of this methodology.

The paper necessarily also provides brief descriptions of the tool set used to implement the methodology for this design example. Though Celoxica developed this methodology and provides a set of tools to support it, the methodology itself is independent of any particular design language or tool vendor. The vision is to represent a methodology entailing successful system design practices that may be applied to all system designs that include custom logic and software-driven microprocessors. As the Software-Compiled System Design methodology provides a direct path from higher-level languages, through co-design and co-verification to direct implementation of FPGA logic, the methodology is especially beneficial for FPSoC design.

2 Challenges of System Level Design

Not too long ago the challenges of system design were mainly discussed in the context of System-on-Chip (SoC) devices that combined millions of gates with embedded processors. However, in the unrelenting march of Moore's law, programmable logic designers now have the opportunity to put millions of gates into FPGA devices. Many of them are doing just that as half of FPGA designs now include over 500K gates[1]. As design complexity increases, so does the adoption of higher-level languages – some simulation usage of C and C++ is now included in 74% of designs[1]. The propagation of FPSoC devices combining mega-logic with embedded processors will further spur the demand for system design tools that improve design productivity while maintaining quality of results (QoR).

Traditionally FPGA hardware is designed using techniques, methodologies, and languages borrowed from ASIC design[2]. Software development is undertaken using techniques, methodologies, and languages designed to describe software systems. There has been a divide between the disciplines and it is obvious that hardware and software methodologies do not talk together[3].

As an example, current methods for embedded systems design require that hardware and software be specified and designed separately. System specifications typically use C or C++ and then create a paper document that delegates the functional design to the respective hardware and software teams. The design implementation is then coded in different languages. The software team maintains the use of C or C++, while the hardware team translates the paper specification into VHDL or Verilog. This process precludes co-design and the partitioning is often decided *a priori* based only on historical divisions. And once the partition decisions are made they can not be revisited, as changes to the partition can necessitate expensive redesign of both the hardware and software,. System verification in this process is similarly hindered by the gap between the hardware design flow and the original specification. The lack of continuity and direct implementation between design phases necessitates exhaustive functional re-verification at every step.

The deficiencies of the current state of system design are clear:

- Lack of a unified hardware-software representation leads to difficulties in verification of the entire system, and to incompatibilities across the hardware-software boundary;
- Defining system partitions in advance leads to sub-optimal designs or requires costly redesign;
- Hardware implementations of the system specification require time-consuming and possibly error-prone rewriting into HDL;
- Lack of a well-defined and flexible co-design methodology makes specification revision difficult and affects time to market.

3 Software-Compiled System Design

To address the challenges of system-level design generally, and FPSoC specifically, we present Software-Compiled System Design (SCSD), a methodology converging hardware and software techniques in the design of electronic systems. This methodology provides a cohesive path from system specification and functional algorithm identification, through partitioning and verification, to system implementation. The resulting output is a functionally verified design implemented in an FPSoC device.

Software-Compiled System Design is defined by the existence of four elements in the design process. First, the algorithm design begins with functional modeling using higher-level language (HLL) design descriptions. Specifically the system descriptions utilize C-based languages such as C, C++, SystemC, SpecC or Handel-C to name a few of the options available to designers. Next using the SCSD methodology, the designer flexibly partitions the design between hardware and software, finding the optimal split before beginning implementation. To meet the third requirement in the SCSD methodology, the system is verified at the highest level to create a functionally correct design before implementation. Finally, the SCSD methodology lives up to the 'software-compiled' portion of its name by providing direct paths to implementation through compilation from C-based descriptions into both software and hardware.

The high-level representation of the Software-Compiled System Design flow is shown in Figure 1 as it is implemented using the Celoxica DK Design Suite.

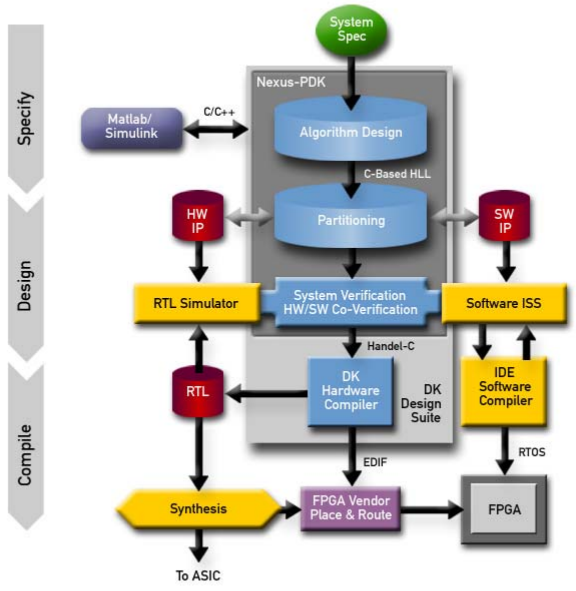


Figure 1: The Software-Compiled System Design flow

3.1 C- based Descriptions

The success of the SCSD methodology is predicated on the use of higher-level language descriptions for system functionality. The exact language used for modeling at this level is less important than the capability to provide compatibility with system architectural analysis and to support easier, verifiable partitioning.

The SCSD methodology supports C-based descriptions that are able to represent the functionality of a design block, independent of the eventual implementation in either hardware or software. While VHDL and Verilog are mature and capable hardware description languages, they are not optimal for compiling software elements of the design to run on processors. C-based description languages enable a common development path between hardware and software elements of the design. This trait of the SCSD methodology allows designers to easily target and retarget functionality between hardware and software implementations.

Often C-based languages are also the functional modeling language of choice for system architects due to their high level of abstraction and interdisciplinary acceptance. Algorithmic models developed in C, or generated from a Matlab/Simulink environment may be used to drive the design. By using these descriptions as the source for functional implementation, the SCSD methodology avoids time-consuming and potentially error prone translations from paper specifications.

There are many C-based languages that could fulfill the requirements for software-compiled system design. So for modeling both hardware and software or maintaining continuity from the algorithm development, the SCSD methodology supports many C-based languages for system functional development. In the Celoxica implementation of the SCSD methodology users have found benefit modeling systems in C, C++, SystemC, SpecC and Handel-C, often employing different languages for different blocks in a mixed-language environment to take advantage of specific modeling strengths in the languages, or to avoid re-writing existing IP. However, as the system design gets closer to implementation, the tools often dictate the specific language for compilation. Embedded software is compiled into the processor from C or C++. Similarly, the Celoxica tool flow employs Handel-C as a simplified route for direct compilation of FPSoC hardware using ANSI-C syntax with simple hardware implementation directives.

3.2 Co-design and Partitioning

Fundamental elements of any co-design methodology, profiling and partitioning are used to help identify optimal design methods early in the design cycle. In the software world, the profiler is used as an analysis tool to examine run time behavior of a program. Applying a similar approach, the profiler in the SCSD methodology can help find the optimal system partition by analyzing hardware and software performance and the interfaces between the two.

Software-Compiled System Design provides a practical means for the co-design of hardware and software. System functionality is developed using technology-neutral C-based descriptions to create a working model of the design. The design is then divided into blocks representing functions that will be implemented in either hardware or software. Using the system specification as a driver, the performance of these blocks and their interaction can be analyzed through profiling tools.

Software block analysis checks for efficient utilization of the processor and performance bottlenecks where parallelization may be added by moving the function into hardware. Hardware blocks are analyzed for execution throughput and the hardware efficiency (gates or FPGA elements). Hardware blocks that are too large or execute a primarily serial function may be moved into software. In addition to the functional blocks, the partitioning analysis must also review the interface between hardware and software to find and remove communication bottlenecks. Due to latency between the system boundaries and interfaces, system profiling can be used to minimize dataflow between the hardware and software. By performing this design analysis and enabling rapid re-partitioning, the SCSD methodology can be used to improve design performance and efficiency.

A key to making this FPSoC design methodology successful is the ability to easily retarget code between software and hardware implementations. In addition to using C-based languages to converge hardware and software modeling practices, the difference in hardware and software interfaces must be addressed. Typically, the FPGA logic is connected to the microprocessor in a memory mapped or programmed I/O fashion. This creates the need to redevelop individual communication protocols and data marshalling routines for each functional block that is moved between hardware and software. This problem is overcome in the SCSD methodology by using an API interface to insert an abstraction layer between the application code and the hardware-software interface.

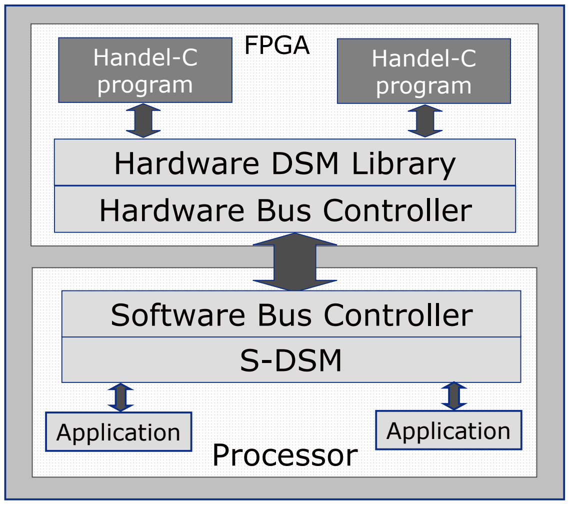


Figure 2: DSM provides an abstraction layer that simplifies the retargeting of code between hardware and software implementations.

The Celoxica Data Streaming Manager (DSM) provides this API for use between Handel-C representations of hardware and ANSI-C software models. DSM offers a simple and transparent interface for transferring multiple independent streams of data between hardware and software (see Figure 2). DSM simplifies system partitioning and final implementation by abstracting away the interface issues, allowing hardware and software descriptions to share common code and communicate through simple system calls. The DSM API provides a structure that minimizes the overhead to this approach and can be easily retargeted to support different processors. The DSM portability means that multiple partitions can be rapidly evaluated and verified with the original system models used as a test bench.

3.3 Co-verification

Much has been written regarding system level verification strategies and this paper supports the view that functional verification should be done up front in the design process. The Software-Compiled System Design methodology is compatible with specification-driven system verification strategies. The methodology can be described as a functional design process (see Figure 3). The design starts with the specification, and continues with representations derived from the original models. A common test suite may be used to verify the functionality at all levels of the design. The end result of the SCSD methodology is a functionally correct design model that can be compiled directly into FPSoC designs.

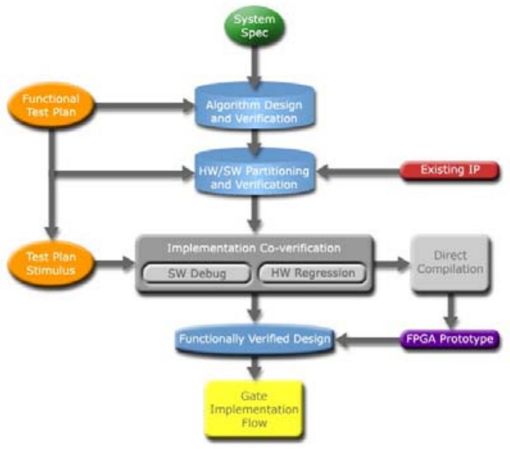


Figure 3: The SCSD methodology supports a specification-driven system verification design flow

To support RTL hand-off to traditional implementation flows, the SCSD methodology also outputs functionally verified HDL models. Using this process, functional correctness is developed at a high-level of abstraction, and maintained via a reference test suite throughout implementation. System performance is also addressed through the SCSD methodology by using profiling and partitioning. This solves major verification and performance issues at the system level, using more efficient levels of abstraction, and significantly reduces the pressure on the RTL implementation flow.

Co-verification technology is critical to the success of the SCSD methodology to support simulation of multiple languages and multiple design levels. Using this approach, C-based representations at the system algorithm level may be simulated simultaneously with RTL hardware representations and software device drivers. The simulation environment must be capable of verifying the models in all the various C-based system representations. The verification environment should also handle verification of all the models generated during system design. This generally requires connected co-simulation between RTL simulators, system-level simulators and software Instruction Set Simulator (ISS) environments. Models of algorithms from a Matlab/Simulink may be run with existing hardware models in VHDL, or with programs running on the processor in an ISS. This allows the designers to modulate the abstraction levels to efficiently simulate the entire system design at each stage of the verification process.

Celoxica implements the SCSD methodology with their Nexus-PDK environment to provide both multi-level and multi-language co-verification. When debugging system performance, it is important that the design verification environment provide a path for both re-partitioning of the system, and for direct implementation to hardware. Nexus-PDK provides a cycle-accurate simulation environment that allows ANSI-C programs (representing system software) and Handel-C applications (representing hardware modules) to interact using DSM. The tool also provides multi-language support for C, C++, SystemC and Handel-C simulation; and third-party co-verification with HDL and ISS simulation, enabling simulation of complete systems. To support partitioning with co-verification, a utility monitors the data passing between applications during simulation. The API functionality used with the SCSD methodology allows system verification to begin immediately and continue seamlessly throughout the functional design process. Once the functional design is completed, it can easily be transferred to a target FPSoC platform for final testing and implementation.

3.4 Direct Compilation

To make co-design and co-verification successful, the methodology must provide a direct, simple, and proven path to implementation. This path should also support the convergence of the hardware and software methodologies so that functionality and performance can be addressed at the system level, instead of at low levels of abstraction. Just as embedded software portions of a system design are compiled to run on a processor, the hardware portions may be 'compiled' directly for implementation into FPGA logic.

For Software-Compiled System Design, Celoxica has worked with partners to develop a direct flow for FPSoC devices. The DK Design Suite compiles C-based descriptions of hardware directly into FPGA logic. Figure 4 shows the interaction of the combined tool set implementing this flow.

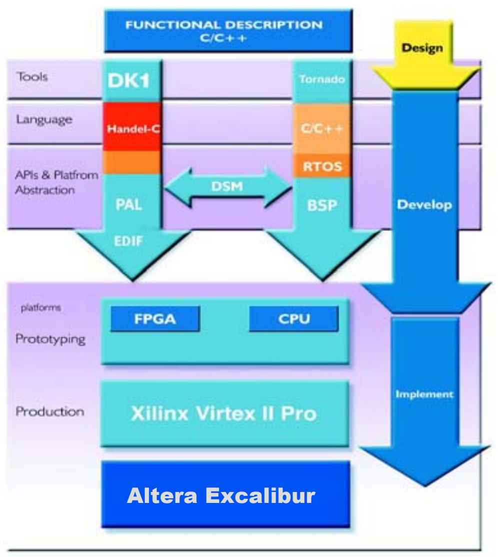


Figure 4: Tool and data interaction in the Software-Compiled System Design flow implementing FPSoC

The SCSD methodology provides an ideal environment for the rapid development of FPSoC applications. Due to the benefits of system partitioning and verification, excellent quality of results can be achieved, substantially beating the results of RTL flows where designers can spend their time optimizing a poor system partition.

3.5 Methodology Benefits

Fusing the techniques of hardware and software methodologies provides significant benefits to designers of FPSoC devices and other systems containing both hardware and embedded processors. The SCSD methodology makes it possible to:

- Prototype the system and easily explore the design space to identify the optimal design solution;
- Partition and re-partition at any time in the design cycle;
- Drive system verification from the specification throughout the functional design process;
- Modulate the design abstraction levels for optimal simulation efficiency;
- Generate human-readable VHDL and Verilog representations of the hardware from C-based descriptions;
- Directly compile FPGA hardware from C-based descriptions for fast design implementation and iteration.

The design case study cited below provides an example of the benefits of Software-Compiled System Design in action.

4 Design Case Study: JPEG2000

To demonstrate the abilities of Software-Compiled System Design, Celoxica and Xilinx partnered on the co-design of a JPEG2000 for implementation in a Virtex II Pro device. In particular, we set out to address the design challenge of system partitioning, co-verification, and the ease of hardware and software integration.

JPEG2000 is a standards-based image-coding system that uses state-of-the-art compression techniques based on wavelet technology. Its architecture lends itself to a range of uses from consumer electronics such as digital cameras, through to medical imaging, remote sensing, surveillance systems and scanners.

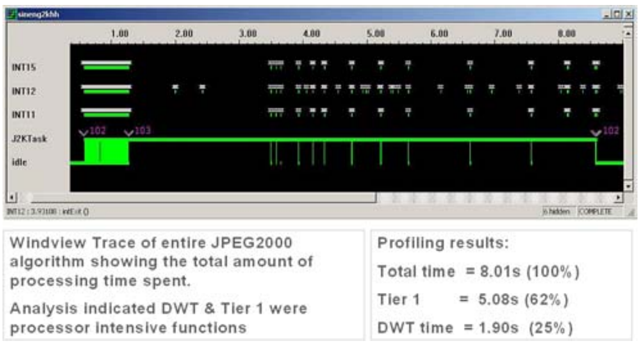
The major engineering goals for this project were to maximize the overall system performance while demonstrating an efficient and effective co-design environment for FPSoC design. The JPEG2000 project used an ANSI-C software specification as the starting point for the system design and employed Software-Compiled System Design using Celoxica, Wind River and Xilinx tools to implement the methodology. The project was completed in four phases with verification continuously applied at each stage: Profile, Partition, Design, and Implement. The design was completed with one engineer over the course of about 2 working weeks.

4.1 Phase 1: Profile and Verify

The project started with the JPEG2000 ANSI-C source code, an application that could benefit from acceleration and flexibility of the FPSoC hardware solution. This code was considered the functional specification for this design. This example is not unique, as a multitude of similar applications could also benefit from the SCSD methodology.

To drive the system verification flow, the JPEG2000 specification code was simulated as software through an appropriate processor target, in this case the IBM PowerPC 405GP. From this exercise the functionality of the system was simulated and verified to establish a test bench that remained constant throughout the design.

The design engineer then performed code profiling to establish where the program spent its execution time and which functions called other functions during execution. This profiling quickly identified the compute-intensive functions in the program. Using Wind River's WindView visualization and diagnostic tool, the DWT (Discrete Wavelet Transform) and Tier 1 encoder were determined to be the processor-intensive functions, consuming 87% of processing time (see Figure 5). Those two functions were selected for further partitioning and analysis.



Windview Trace of entire JPEG2000 algorithm showing the total amount of processing time spent.

Analysis indicated DWT & Tier 1 were processor intensive functions

Profiling results:

Total time = 8.01s (100%)

Tier 1 = 5.08s (62%)

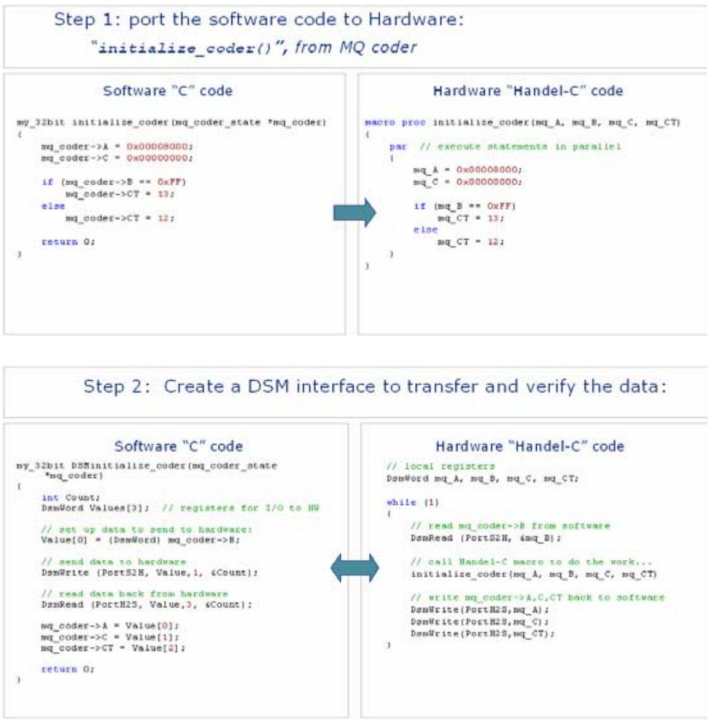
DWT time = 1.90s (25%)

Figure 5: WindView trace of the JPEG2000 algorithm

4.2 Phase 2: Partition and Verify

Software-Compiled System Design is unique in providing the designer with a flexible partitioning methodology linked to system verification. Using the Celoxica tools and DSM technology the designer can confidently explore and innovate in the design space to identify the optimal system partition for the best Quality of Design (QoD).

In the JPEG2000 project, DSM validated the profiling information determined in Phase 1 of the design. Using DSM, the design engineer analyzed the data flow, burst length and frequency between the hardware and software and fine-tuned the partition to optimize the project goals. The DSM API facilitated the process of design partitioning between hardware and software (see Figure 6).



Step 1: port the software code to Hardware:
"**initialize_coder()**", *from MQ coder*

Software "C" code

```
my_32bit initialize_coder(mq_coder_state *mq_coder)
{
    mq_coder->A = 0x00008000;
    mq_coder->C = 0x00000000;

    if (mq_coder->B == 0xFF)
        mq_coder->CT = 13;
    else
        mq_coder->CT = 12;

    return 0;
}
```

Hardware "Handel-C" code

```
macro proc initialize_coder(mq_A, mq_B, mq_C, mq_CT)
{
    par  // execute statements in parallel
    {
        mq_A = 0x00008000;
        mq_C = 0x00000000;

        if (mq_B == 0xFF)
            mq_CT = 13;
        else
            mq_CT = 12;
    }
}
```

Step 2: Create a DSM interface to transfer and verify the data:

Software "C" code

```
my_32bit DSMinitialize_coder(mq_coder_state
    *mq_coder)
{
    int Count;
    DsmWord Values[3];  // registers for I/O to HW

    // set up data to send to hardware:
    Value[0] = (DsmWord) mq_coder->B;

    // send data to hardware
    DsmWrite (PortS2H, Value,1, &Count);

    // read data back from hardware
    DsmRead (PortH2S, Value,3, &Count);

    mq_coder->A = Value[0];
    mq_coder->C = Value[1];
    mq_coder->CT = Value[2];

    return 0;
}
```

Hardware "Handel-C" code

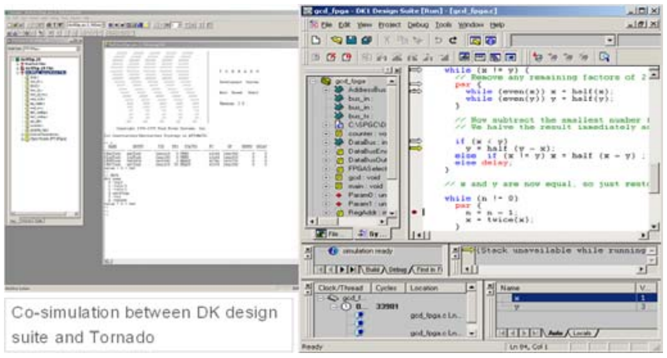
```
// local registers
DsmWord mq_A, mq_B, mq_C, mq_CT;

while (1)
{
    // read mq_coder->B from software
    DsmRead (PortS2H, &mq_B);

    // call Handel-C macro to do the work...
    initialize_coder(mq_A, mq_B, mq_C, mq_CT)

    // write mq_coder->A,C,CT back to software
    DsmWrite(PortH2S,mq_A);
    DsmWrite(PortH2S,mq_C);
    DsmWrite(PortH2S,mq_CT);
}
```

Figure 6: DSM is used to ease the process of moving design functionality between software (ANSI-C) and hardware (Handel-C) representations

4.3 Phase 3: Design and Verify

With the optimal partition determined and verified, the optimization of the design began. With SCSD, the performance and size optimization can be done by combining hardware and software functionality, looking at the system in its entirety. Optimizations of the JPEG2000 software included combining multiple function calls into single calls. Blocks destined for hardware were optimized by employing more parallelism, and editing functions to infer pipelining and efficient hardware constructs.

DSM was used to provide a cycle-accurate simulation environment that allowed the hardware and software to interact, keeping them connected throughout design optimization. The software in the system was run as a native executable on a PC, with the hardware being run in C and Handel-C using the simulation capabilities of Celoxica's Nexus-PDK. Co-simulation between the hardware and software was performed in tandem the Wind River Tornado environment (see Figure 7).


Co-simulation between DK design suite and Tornado
gcd_fpga - DK1 Design Suite [Run] - [gcd_fpga.c]
File Edit View Project Debug Tools Window Help
simulation ready
Stack unavailable while running
Clock/Thread Cycles Location
Name
Ready

```
    while (x != y) {
      // Remove any remaining factors of 2
      par {
        while (even(x)) x = half(x);
        while (even(y)) y = half(y);
      }

      // Now subtract the smallest number f
      // We halve the result immediately ac

      if (x < y)
        y = half (y - x);
      else  if (x != y) x = half (x - y) ;
      else delay;
    }

    // x and y are now equal, so just rest

    while (n != 0)
      par {
        n = n - 1;
        x = twice(x);
      }
```

Figure 7: Celoxica's Nexus-PDK and the Wind River Tornado environment running co-verification of the system hardware and software in tandem

Since the system specification was described in ANSI-C, the designer implemented the software in ANSI-C, and added Handel-C hardware extensions to the code to represent the hardware. These extensions provided efficient control over the area, timing, clocks, RAM, ROM and interfaces of the FPGA logic.

4.4 Phase 3b: Specification Change

At this stage in the design a specification change was introduced. A novel lifting algorithm was developed that performs a two-dimensional DWT to improve processing time. The algorithm was readily available as an HDL IP block and the decision was made, in the context of minimizing time and maximizing IP investment, to integrate the IP into the design as a black box. The integration was simplified by using the 'interface' declaration from a Handel-C block in order to connect the third-party IP, complete with RTL co-verification, into the Software-Compiled System Design flow.

4.5 Phase 4: Implement and Verify

The target platform for the JPEG2000 project was the Wind River SBC405GP and Proteus FPGA daughter card, a Virtex II Pro prototyping platform. The design was implemented in this environment for timing simulation, emulation and block optimization. Final implementation was retargeted to the Virtex II Pro ML300 evaluation platform.

Re-targeting from the development platform to the final evaluation platform was simplified using the Celoxica Platform Abstraction Layer (PAL), which provides an API to shield the application code from the low-level hardware interfaces. This built on a library of low-level interfaces specific to each target platform called the Platform Support Library (PSL), which was accessed using the PAL API by the Handel-C application code representing this FPGA logic.

Software object code for the PowerPC processor was compiled directly from C into the PPC405GP under VxWorks. The hardware implementation was compiled directly from Handel-C using the EDIF output generated by the Celoxica DK Design Suite. This EDIF netlist was optimized for Virtex II Pro for maximum efficiency and best Quality of Results (QoR). The DK Design Suite also produced VHDL and Verilog output of this design hardware, though the RTL path was not used for implementation this project.

4.6 Design Results

The DWT results for the JPEG2000 project are shown below in Table 1. These results are compared against the performance of the handcrafted VHDL authored by a JPEG2000 domain expert. The Software-Compiled System Design methodology provided a systematic approach to the problem leading to not only substantial savings in design time, but also to improved design quality.

To achieve the results, the DWT portion of the JPEG2000 design was compiled directly from Handel-C to FPGA hardware. The design was optimized to achieve maximum system speed while maintaining or reducing the hardware utilization area. To validate the methodology, the results were compared against the same DWT functionality as originally hand-coded in VHDL. Using the SCSD methodology, the designer was able to achieve a comparable size design with faster performance results in less time.

JPEG2000	Original	Celoxica SCSD		
Results	(VHDL)	Pass 1	Pass 2	Final Results
Slices	800	646	546	758**
Device Util.	7%	6%	5%	7%
Speed (MHz)	128	110	130	151
Lines of code	435	386	386	395
Design time (days)	20*	6	7 (6+1)	7 (6+1)

*VHDL design time does not include partitioning
**Final SCSD result Includes HDL IP block

Table 1: JPEG2000 case study DWT function implementation metrics

The engineer on this JPEG2000 project, while an expert in Celoxica tools and Handel-C implementation, had no prior experience with the JPEG2000 algorithm and yet was able to translate the algorithm into a working hardware implementation in less than half the time of the original VHDL implementation. The engineer also easily met the system design constraints for size and performance as seen in the table of results.

The difference is that with the SCSD methodology the designer was able to work at a higher, more efficient level of abstraction, and then concentrate on optimizing the entire system. These results provide clear validation of the SCSD methodology raising the level of abstraction to increase designer productivity without compromising design quality or performance.

5 Conclusion

Software-compiled System Design is a proven methodology for the design of programmable systems. It employs high levels of abstraction through the use of C-based design languages to provide solutions for system partitioning, co-verification and the integration of hardware and software into FPSoC devices.

The Celoxica implementation of the SCSD methodology connecting the DK Design Suite to common embedded software and FPGA design tools provided the tool set for a real-life system design test. The performance results from this test demonstrated significant improvements in overall design productivity, while also improving system performance and quality of design. Overall improvements in the quality of design were realized by informed and accurate partitioning decisions. Better up front system verification and direct compilation to FPGA hardware improved the design productivity. Finally, the system methodology allowed the designer to find speed and area optimizations in the system by analyzing the entire design including interactions between the FPGA logic in the microprocessor.

The bottom line is that the Software-Compiled System Design methodology offers real competitive advantages for designers of programmable systems.

References

[1] G. Smith, "ASIC Design Times Spiral Out of Control", *Gartner Dataquest Research*, April 2002
[2] P. Garrault, Synthesis Tool Enhancements for Virtex Architectures, Xilinx, 2002
[3] G. Smith, "EDA 2002: The Acceleration of RTL Design", *Gartner Dataquest Research*, October 2002