Hardware Assisted Real-Time Simulation

Douglas D. Hodson
Capabilities Integration Directorate
WPAFB, OH
douglas.hodson@wpafb.af.mil

Rusty O. Baldwin
Air Force Institute of Technology
WPAFB, OH
rusty.baldwin@afit.edu

John G. Weber
University of Dayton
Dayton, OH
john.weber@notes.udayton.edu

ABSTRACT

With the advent of Field Programmable Gate Arrays (FPGA) and System-On-a-Programmable-Chip (SOPC) technology, system designers and software developers can custom design underlying hardware platforms simulation application requirements. Specifically, this technology allows a developer to use custom hardware to perform efficiently what might otherwise be time-consuming software computations. This approach to real-time simulations has heretofore not been a viable alternative.

This paper introduces FPGAs, SOPC technologies and supporting vendor tools. It also discusses the use of these tools in defining custom hardware to execute specific time-constrained tasks to support robust real-time simulations.

ABOUT THE AUTHORS

Douglas D. Hodson is an Electrical Engineer at the Simulation and Analysis Facility, Wright Patterson Air Force Base AFB, OH. He is the technical lead of the Extensible Architecture for the Analysis and Generation of Linked Simulations (EAAGLES) software framework. This framework is currently being used to support the development of both virtual and constructive and stand-alone and distributed simulation applications. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He is also a graduate student and Adjunct Instructor at the Air Force Institute of Technology working towards his Ph.D. in Computer Engineering.

Rusty O. Baldwin is an Associate Professor of Computer Engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH. He received a B.S. in Electrical Engineering (cum laude) from New Mexico State University in 1987, an M.S. in Computer Engineering from the Air Force Institute of Technology in 1992, and a Ph.D. in Electrical Engineering from Virginia Polytechnic Institute and State University in 1999. He served 23 years in the United States Air Force. He is a registered Professional Engineer in Ohio and a member of Eta Kappa Nu, and a Senior Member of IEEE. His research interests include computer communication networks, embedded and wireless networking, information assurance, and reconfigurable computing systems.

John G. Weber is a Professor of Electrical and Computer Engineering at the University of Dayton. He has over 35 years of industrial experience in the design and development of embedded computer systems for digital avionics applications. He also developed real-time, hardware-in-the-loop simulations in support of the development and testing of digital avionics systems. He received a B.S.in Electrical Engineering in 1963 from St. Louis University, an M.S. in Electrical Engineering in 1964 from the University of Missouri-Columbia, and a Ph.D. in Electrical Engineering from the University of Missouri-Columbia in 1971.

Hardware Assisted Real-Time Simulation

Douglas D. Hodson
Capabilities Integration Directorate
WPAFB, OH
douglas.hodson@wpafb.af.mil

Rusty O. Baldwin
Air Force Institute of Technology
WPAFB, OH
rusty.baldwin@afit.edu

John G. Weber
University of Dayton
Dayton, OH
john.weber@notes.udayton.edu

INTRODUCTION

With the advent of Field Programmable Gate Arrays (FPGA) and System-On-a-Programmable-Chip (SOPC) technology, system designers and software developers can custom design underlying hardware platforms to meet application requirements.

Specifically, this technology allows a developer to use custom hardware to perform efficiently what might otherwise be time-consuming software computations. This approach to real-time simulations has heretofore not been a viable alternative.

Traditional methods to decrease software execution time include:

- Increasing CPU clock speed
- Replacing a processor with a higher performance processor
- Coding appropriate sections of software in assembly language

FPGA-based processors provide additional optimization opportunities capable of achieving much higher performance gains (Altera-C2H, 2006). Some techniques include:

- The ability to rapidly alter the FPGA design, allowing the designer to prototype a variety of architectures
- The ability to divide processing tasks by instantiating multiple processor cores
- The ability to augment a processor with custom hardware that off-loads the static processor-intensive operations into the FPGA fabric
- The ability to modify memory architectures for memory-intensive operations, such as using high-speed, point-to-point connections to fast memory buffers

FPGA Technology

A Field Programmable Gate Array (FPGA) is a configurable electronic component used to build digital hardware. The FPGA is a semiconductor device containing configurable logic components and interconnects, which may be used to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex memory elements such as flip-flops or latches. Current FPGAs may incorporate complex digital logic blocks such as memories, hardware multipliers, digital signal processing blocks, and even complete digital processors.

The designer may use the configurable logic to interface these more complex blocks together to create an entire system on a chip.

Configuring the FPGA is accomplished by describing the hardware functions to be realized using a Hardware Description Language (HDL). Two of the most popular languages are Verilog and VHDL. Either of these languages supports the description of the hardware in a variety of modes. The most basic mode is the structural mode. In this mode, the designer lists all of the devices (gates, flip-flops, etc) and their input and output connections. While this is useful in some applications, the power of the HDL isn't realized unless the behavioral mode is used. In this mode, the designer describes the behavior of the hardware using constructs borrowed from software programming languages (e.g. IF–THEN–ELSE, CASE, etc.)

A software tool (called a compiler) then translates the hardware description and synthesizes the required logic to fit on the selected device. The output of the compiler is a binary file which configures each of the devices on the FPGA and makes the required interconnections. The FPGA is then programmed via a serial interface which allows the binary file to be loaded into the on-board configuration RAM.

The FPGA is a completely reprogrammable device and instantiated hardware logic has access to all the pins on the chip to perform I/O operations.

A typical development board includes an FPGA chip connected to a variety of specialized discrete components that support in some type of I/O operation like LEDs, switches, buttons, USB interfaces, or ethernet controllers.

SOPC Primer

FPGAs have been historically used as "glue" logic to connect existing discrete digital components (e.g., CPU, memory). Since custom Very Large-Scale Integration (VLSI) chips are expensive to manufacture, FPGAs have also served as a convenient tool to prototype and test Application-Specific Integrated Circuits (ASIC) designs before production.

In the past few years, the capabilities of FPGAs have increased enabling complex hardware designs to be implemented on a single chip. The cost of FPGAs has dropped to a point where hardware manufactures now must consider the benefit of producing a custom VLSI chip compared to the inherent advantages of a field programmable chip. Custom VLSI chips in high volume are still more cost effective and faster, but the gap is closing.

These trends have led to a market demand for embedded computer hardware designs based exclusively on FPGAs. SOPC is the terminology used to describe the domain of implementing computer systems entirely on a single Programmable Logic Device (PLD), or typically an FPGA.

Altera and Xilinx are two prominent suppliers of FPGAs and their associated development tools. Each vendor offers a range of chips that include a set of resources (configurable logic units, registers, RAM memories) to satisfy different application markets. Some FPGAs include complete CPUs as one of the resources that can be interfaced to custom logic.

Altera offers the Quartus II system and design suite which synthesizes Verilog and VHDL code. The Quartus software also includes an integrated "SOPC Builder" tool to define hardware for a complete system-on-a-chip design.

Altera offers the Nios II processor as a "soft" logic core which may be instantiated on the FPGA configurable logic. SOPC Builder can generate a system library for the software development process. The Nios II Integrated Development Environment (IDE) which is based upon the Eclipse (a popular open source vendor-neutral development platform and application framework) development platform, can then be used to compile, link and download the final configuration file to the FPGA chip for debugging and execution.

A typical design flow follows these steps:

- Create a project in Quartus that includes information about how each pin on the FPGA is wired to the external discrete chips.
- Design a "system" using the SOPC Builder tool. After the system has been designed, the tool generates all the HDL files (VHDL or Verilog) needed for Quartus to compile.
- Compile the design with Quartus. This produces a small configuration file that is loaded into the FPGA to instantiate the hardware logic.
- Download this configuration file to the FPGA.
- Create a project with Nios. One of the project settings specifies the intended target hardware.
- Nios automatically creates a "system library" that provides an underlying framework for application development. For example, part of the framework can include the C run-time libraries. Optionally, a real-time operating system kernel can be compiled into the system library which supports executing multiple tasks.
- Write the software code to execute on the FPGA. Nios supports C and C++ code. (Note: Nios is using a version of GCC that has a Nios CPU as a target.)
- Compile the software.
- Download the software to the FPGA and execute. It should be noted that standard input (stdin) and standard output (stdout) are provided as interfaces using the download cable as a transmission medium.

Xilinx also offers similar capabilities through its ISE Foundation synthesis tool and the Platform Studio Integrated Development Environment. Both companies offer development boards with a wide range of capabilities based on a particular FPGA chip.

A market to develop and encapsulate hardware logic into reusable blocks, so called Intellectual Property (IP) cores, has developed. Both Altera and Xilinx offer a number of reusable cores with their design environments and make it easy for a developer to design and build new ones. Example cores include configurable CPUs (Nios from Altera, MicroBlase from Xilinx), timers, and memory interfaces.

One of the trade-offs in SOPC systems is while much of the hardware logic can reside on the FPGA itself, usually all of it does not. FPGAs typically provide memory in the form of on-chip block RAMs, but are much smaller than the high density memories that can be connected externally. Arrays of registers can be configured as memory, but is an inefficient use of those resources.

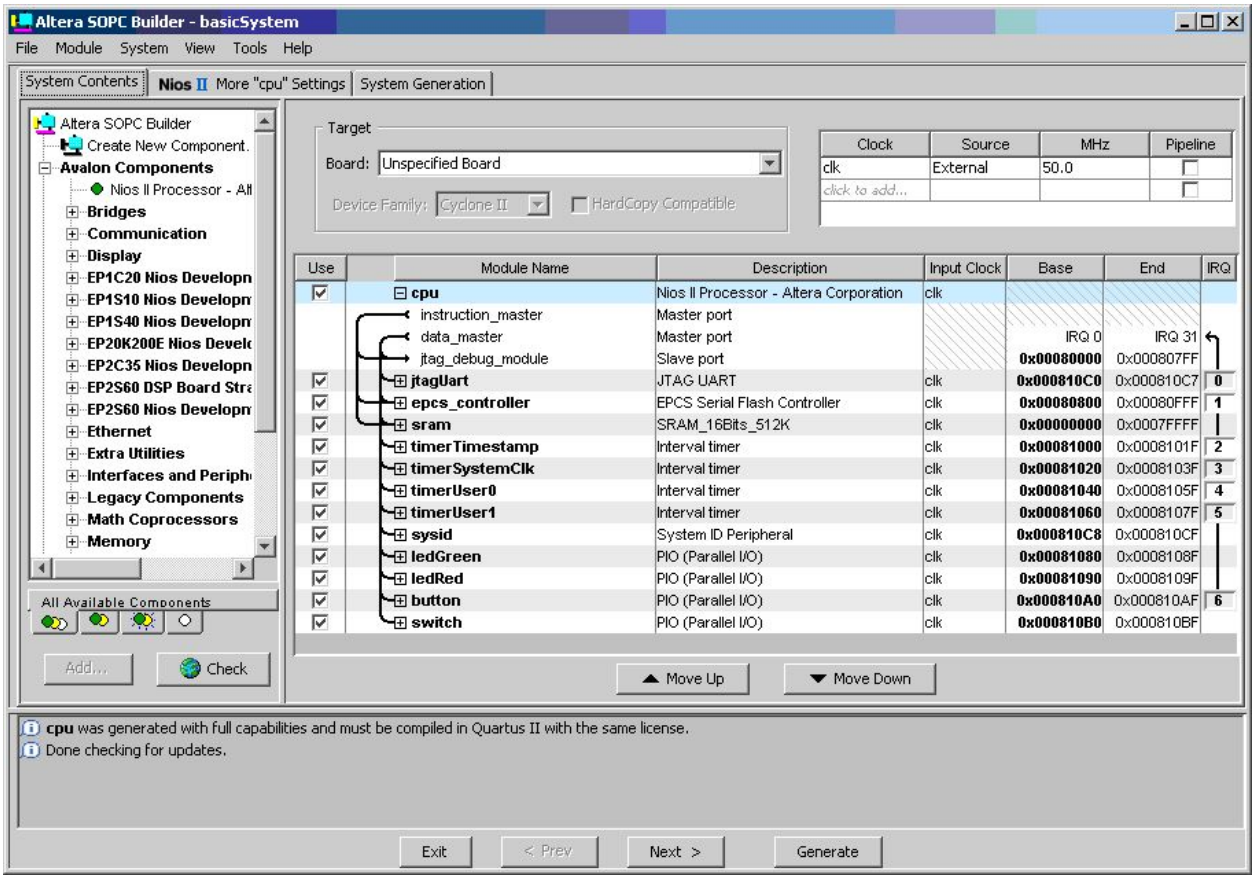

Figure 1: Altera's SOPC Builder Interface

A TRADITIONAL DESIGN

A traditional CPU-based design was constructed as an illustrative example of how to leverage these tools for real-time system development and the creation of new simulation applications. The example is followed by a discussion of ways to improve the design by exploiting the underlying hardware.

This example was constructed with the following development tools:

- An Altera DE2 development board with a Cyclone II FPGA. This board includes a wide range of I/O devices that can be accessed by the FPGA including ethernet, RS232, video in (NTSC/PAL) and out (VGA 10-bit DAC), USB 2.0, PS/2 and audio capabilities. The board features 8-MB SDRAM, 512K SRAM, 4-MB Flash and an assortment of switches, buttons and LEDs.
- Quartus II v6.0 and Nios II v6.0 software.

This design uses the Micrium's uC-OS real-time kernel which is part of the Nios development package. uC-OS is a priority-based operating system with 64 priority levels. Each level can be assigned to one task for a total of 64 possible tasks. Micrium reserves a few levels so the total number of user tasks is less.

The hardware design for this system is defined using Altera's SOPC Builder tool (see Figure 1). The design consists of a single Nios microprocessor connected to four interval timers that reside on the FPGA. The design also specifies external connections to SRAM, buttons, and switches. That is, the components that do not reside on the FPGA itself; they are physically connected via pins on the chip.

As the figure shows, the left window pane lists all the available components that can be used to assemble an SOPC design. This includes vendor supplied IP cores and custom user defined cores. The main window shows the components in the design and how they are connected. It is also used to specify hardware interrupt numbers and a system memory address map.

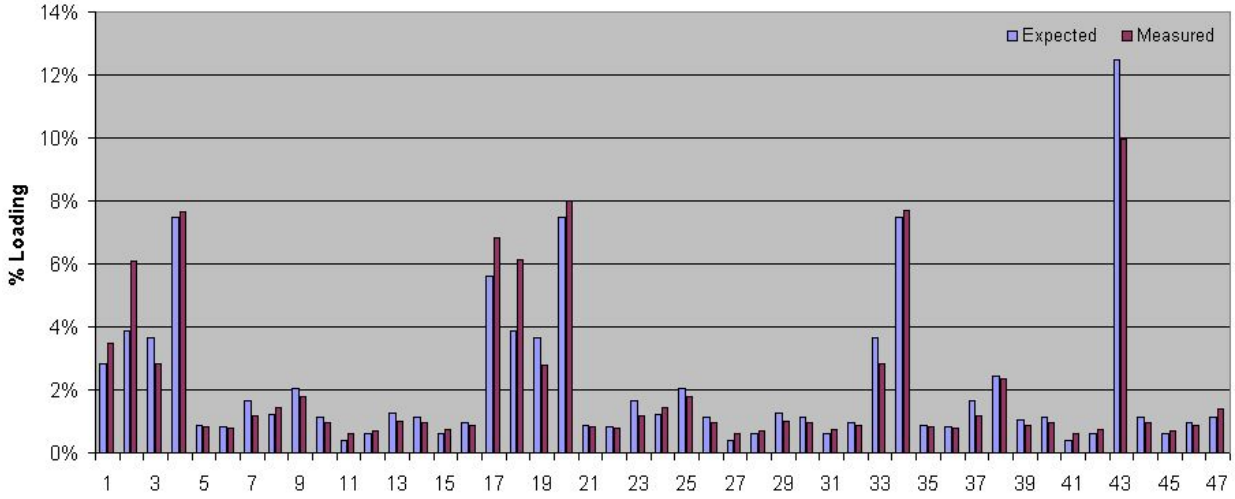

Figure 2: CPU Loading Across Tasks/Applications

Once the design is completed the tool will generate all the HDL files needed by Quartus so that the design can be compiled. The configuration file produced can then be loaded into the FPGA.

Nios was used to create a system library that contains interfaces to Altera's Hardware Abstraction Layer (HAL) API, libc (the C library) and Micrium's uC-OS real-time kernel. The HAL system library is a lightweight run-time environment that provides a device driver interface for programs to communicate with the underlying hardware (Altera-Nios, 2006). After the system library is created, the next step is to develop the applications.

An Example Application

As an example, an application was designed to exercise almost all of the 64 available task priorities supported by the uC-OS real-time kernel. The application consists of the following:

- 47 periodic tasks with frequencies that range from 10-100 Hz, each with a different execution time.
- 4 tasks associated with processing button interrupts.
- 1 task for simulating and generating aperiodic events. This simulates the behavior of random events entering the system for processing. For example, a pilot pressing buttons on a Multi-Function Display (MFD).
- 1 task for processing aperiodic events.
- 1 "startup" task to create all of the above tasks.

Each task was defined by a function with a name in the following form AppX, where X is a number that specifies the task/application. Using that naming convention, the 47 periodic tasks are App1, App2,...,App47. The task associated with simulating and generating aperiodic events is App50. The task associated with processing aperiodic events is App51. Tasks for processing button interrupts include App52, App53, App54 and App55. App0 is the startup task.

The application executed for 20 seconds and statistics were collected on the time spent executing each task. At the end of this time, the operating system printed the collected data to a console window provided by the Nios environment.

Given the above applications to be executed, an expected cpu loading of the 47 periodic tasks was compared to what is measured from the 20 second run (see Figure 2).

A high degree of correlation exists between the expected behavior of the 47 periodic tasks and what is measured. The "background" aperiodic tasks execute as well (not shown), but only after the higher priority "foreground" tasks have finished. This example confirms that simulation applications can be developed with existing FPGA development tools and run as expected.

EXPLOITING HARDWARE

The reconfigurability and programmability of FPGAs allow for considerable flexibility in how hardware is instantiated. Fundamental design choices are made as to what should be executed in software verses what can be implemented in hardware. These architectural trade-offs are weighted with consideration of manufacturing costs, maintainability, and application requirements.

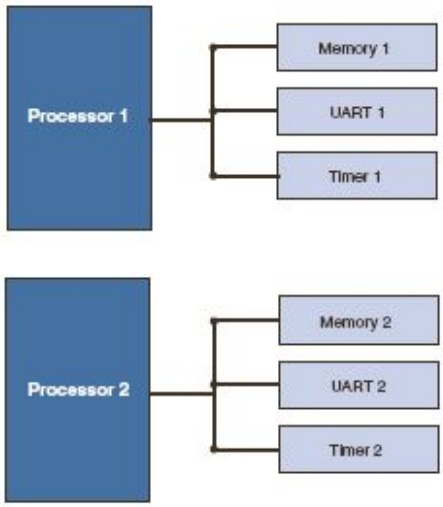

Figure 3: Autonomous Multiprocessor System

Hardware architectures can also be modeled, prototyped and simulated with a variety of langauges other than Verilog and VHDL as discussed later.

Altera offers several mechanisms to ensure the hardware footprint of a system fits application requirements. All of these approaches can be viewed as alterations to the "traditional" single, fixed instruction set CPU architecture. Approaches include instantiating two or more CPUs, customizing the CPU's instruction set, and directly translating C code into hardware.

Multi-CPUs

As shown in Figures 3 and 4, microprocessor systems designed with the SOPC Builder can be classified into two main categories: autonomous microprocessor and multiprocessor systems that share resources (Altera-Multi, 2005).

Autonomous systems look and operate just like two separate systems. Resource sharing usually comes in the form of sharing memory. Sharing data memory between multiple CPUs is more difficult than sharing instruction memory since data memory can be written to as well as read (Altera-Multi, 2005).

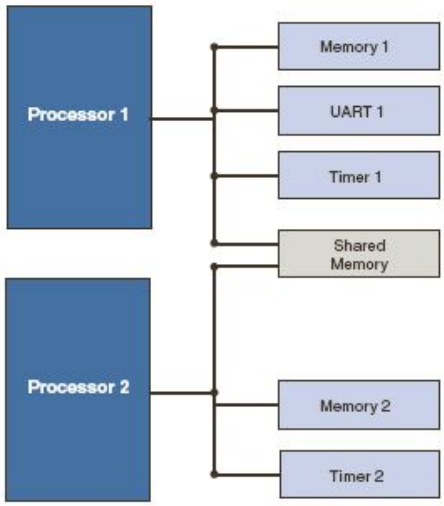

Figure 4: Shared Multiprocessor System

A protection mechanism that is available is a hardware mutex (mutual exclusion) core. This component is a shared resource and provides an atomic "test and set" operation. There are some cases when a mutex core may not be necessary; for example, when only one processor writes to a particular set of memory locations (Altera-Multi, 2005).

In virtual distributed simulation, one application for this type of hardware structure is to pipeline some simulation activities. For example, one CPU could be dedicated to processing DIS entity information from the network. This completely relieves the "main" CPU from any network activities.

Custom Instructions

System designer can accelerate time-critical software algorithms by adding custom instructions to the Nios instruction set as shown in Figure 5. System designers can use this feature for a variety of applications, e.g., to optimize software inner loops for digital signal processing (DSP), packet header processing, and computation-intensive applications (Altera-Custom, 2005).

There are different custom instruction architectures to suit various application requirements. The architectures range from a simple, single-cycle combinatorial architecture to an extended variable-length, multi-cycle custom instruction architecture (Altera-Custom, 2005).

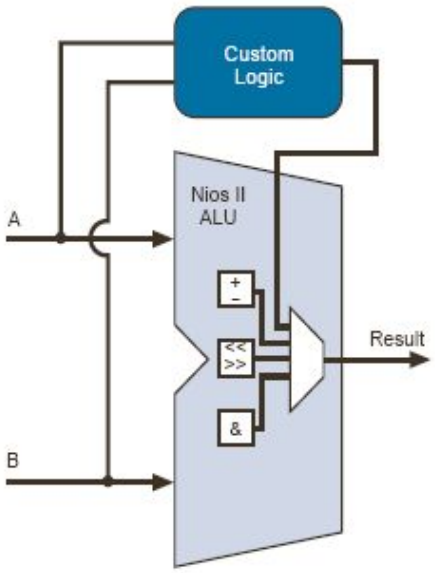

Figure 5: Custom Instruction Logic

Numerous applications can exploit this feature to reduce computational bottlenecks.

C to Hardware

Altera has recently introduced a new technology that allows a developer to create custom hardware accelerators directly from ANSI C source code. A hardware accelerator is a block of logic that implements a C function in hardware, which often improves the execution performance by an order of magnitude (Altera-C2H, 2006). Altera cautions and clearly emphasizes that this new compiler is not designed to be a general purpose tool for creating arbitrary hardware systems using C as a design language.

Hardware accelerators generated by this compiler have the following characteristics (Altera-C2H, 2006):

- Parallel scheduling - The compiler recognizes events that occur in parallel. Independent statements are performed simultaneously in hardware.
- Direct memory access - Accelerators access the same memories that the Nios II processor acesses during execution.
- Loop pipelining - The compiler pipelines the loop logic based on memory access latency and the amount of code that operates in parallel.
- Memory access pipelining - The compiler pipelines memory accesses to reduce the effects of memory latency.

Altera is careful in its claims about this technology. It is emphasized that the compiler is intended to augment the performance of programs that run on the Nios II processor and that it does not replace the processor.

Code to accelerate must be expressed as an individual C function. The C2H Compiler converts all code within and below the chosen function to a hardware accelerator block. If the function calls a subfunction, the C2H Compiler will convert the subfunction to a hardware accelerator. Therefore, subfunctions must also be good candidates for C2H acceleration (Altera-C2H, 2006).

PROTOTYPING ARCHITECTURES

While Verilog and VHDL remain the most popular languages for describing hardware for syntheses, other languages are emerging to specify hardware functionality at even higher behavioral or more abstract levels. All of these languages support the prototyping and simulation of system designs. Using these languages to support the synthesis of real hardware is limited to what language constructs a particular vendor supports.

Two important languages that deserve attention include SystemVerilog and SystemC. SystemVerilog tends to be favored by computer engineers because it builds upon an already familiar language while SystemC tends to be be embraced by computer scientists (Aldec, 2006).

SystemVerilog

SystemVerilog is a hardware description language based on Verilog. It is an extension of Verilog-2001; all features of that language are available in SystemVerilog.

Although it has some features to assist with design, the thrust of the language is in verification of electronic designs (Wikipedia, 2006).

SystemC

SystemC is a set of library routines and macros implemented in C++, which makes it possible to simulate concurrent processes, each described by ordinary C++ syntax. Instantiated in the SystemC framework, the objects described in this manner may communicate in a simulated real-time environment, using

signals of all the datatypes offered by C++, some additional ones offered by the SystemC library, as well as user defined.

The behaviours (processes) defined may be instantiated any number of times, and provisions are made for processes defined by hierarchies of other processes, as one would expect (Wikipedia, 2006).

SystemC is an open standard and can be freely downloaded from the internet.

FINAL THOUGHTS

This paper introduces some of the state-of-the-art tools currently available for FPGA system design. The two dominant manufacturers, Altera and Xilinx, both offer tools that enable a developer to customize the hardware platform of an SOPC design to fit requirements.

A simple SOPC design was constructed, and a software application that executes 54 of the 64 available tasks available utilizing a priority-based real-time operating system was presented. Measured loading across tasks matched well with what was expected, which provides some confidence that the technology is mature enough to build real systems.

Further optimizations in the form of multi-CPUs, custom instructions and direct C to hardware translations need to be explored to fully determine their potential.

ACKNOWLEDGEMENTS

The authors wish to thank the good folks at Altera for providing a DE2 development board to pursue research in this area. Altera also provided access to their latest development tools and training documentation.

ACRONYM LIST

API - Application Programming Interface

ASIC - Application-Specific Integrated Circuit

DSP - Digital Signal Processing

FPGA - Field Programmable Gate Array

HAL - Hardware Abstraction Layer

HDL - Hardware Description Language

MFD - Multi-Function Display

PLD - Programmable Logic Device

RAM - Random-Access Memory

SDRAM - Synchronous Dynamic Random Access Memory

SOPC - System-On-a-Programmable-Chip

SRAM - Static Random-Access Memory

VHDL - VHSIC Hardware Description Language

VHSIC - Very High Speed Integrated Circuit

VLSI - Very Large-Scale Integration

REFERENCES

Aldec (June 2006). Technical discussions with lead engineers.

Altera-Multi (2005). Creating Multiprocessor Nios II Systems Tutorial v1.0.

Altera-Custom (2005). Nios II Custom Instruction User Guide v1.2.

Altera-C2H (2006). Nios II C2H Compiler User Guide v6.0.

Altera-Nios (2006). Nios II Software Developer's Handbook v6.0.

Altera-Quartus (2006). Quartus II Handbook v6.0.

Hamblen, James, et al. (2006). Rapid Prototyping of Digital Systems. Springer.

Laplante, Phillip A. (2004). Real-Time Systems Design and Analysis, Wiley-Interscience.

Lui, Jane W.S. (2000). Real-Time Systems, Prentice-Hall.

Singhal, Sandeep, et al. (1999). Networked Virtual Environments, Design and Implementation, Addison-Wesley.

Wikipedia (2006). www.wikipedia.com.