>: Document Type White Paper

Celoxica
>: cutting a LONG story SHORT //

>: Title

Handel-C For Hardware Design

The Value Of Software Design And Debug Methods To Designers Addressing Reconfigurable Logic

>: Date August 2002

Introduction

As systems increase in size and complexity, designers will benefit from a new breed of tools that complement those they use today. These new tools simplify the process of describing functionality in hardware through the application of a high level approach to EDA that is inspired by the software world. Fusing software and hardware methodologies, these tools introduce three aspects of software development to the designer: a C-based language for describing functionality; a design system with symbolic debugging; and libraries of predefined functions including access to peripherals and processors in hardware via common APIs.

A C-Based Language Approach

Hardware design methodologies, from schematic capture to HDLs, have evolved from a bottom-up approach in which hardware functionality is developed by describing circuit structure. To this end such approaches have focused on maintaining low-level design control but there are limitations if such methods are used exclusively to address a large design area. One of the first problems arises because most functionality is comprised of both sequential and parallel logic and yet HDLs have evolved from an exclusively parallel world for describing the hardware rather than describing the desired function. What is needed is a language the raises the level of abstraction sufficiently to enable the designer to describe in the briefest possible way the desired function rather than its underlying structural detail.

While Register Transfer Level (RTL) subsets of HDLs, such as VHDL and Verilog, do provide a functional interpretation of the hardware description to enable the generation of hardware structure at compile time, their parallel nature requires the design engineer to add extra logic for sequential execution, for example; an FSM expressed as a case statement. Code is cut up in pieces and put into the case statement depending in which clock cycle it is to be executed. The order of execution is then controlled by conditional settings of the state in each of these pieces. This is essentially GOTO programming. With the inadequacies of GOTO programming, something that was abandoned decades ago in the software world, the code becomes less readable and potentially dangerous, for instance creating infinite loops.

By introducing a language that is similar to ANSI-C to the hardware design process, designers gain a language that is sequential by default with a high-level flow that is geared for programming functionality. But hardware is parallel and this needs to be accounted for if a software-influenced hardware design methodology based entirely on C is going to succeed at the RTL level. There are two high-level alternatives: behavioral compilation; or, the addition of a simple way to express parallelism.

Behavioral compilers seem ideal, automating the processes between software input and hardware output. But today's compilers give the user little control over the quality of the output. They have been likened to a "long, dark tunnel" where the input is hard to relate to the output should the result need optimizing.

In contrast, adding a construct for parallelism delivers a level of abstraction over RTL without compromising manual control over the result. Handel-C is an example of such a hardware programming language with a control flow that is clear to the designer. Handel-C closely corresponds with a typical software flow and provides the essential extensions required to describe hardware. These include flexible data widths, parallel processing and communications between parallel threads. Sequential by default, Handel-C has a `par` construct. When a block of code is qualified by `par`, statements are executed concurrently and synchronized at the block end. This simple construct allows for the expression of mixed sequential and parallel flows in compact and readable code. The hardware designer can begin with software programs compiled in the Handel-C compiler to generate a sequential solution. Then the designer can gradually use parallel constructs and pipelining to optimize the design. While the user sets the key variable widths, the language can deduce most of those that remain.

A high level approach such as the Handel-C methodology provides a level of abstraction that makes it easier for designers to address a larger design space and find the correct solution for making a design smaller and/or faster.

The Design Environment

Today’s hardware design methodologies are computer-based interpretations of early tools such as bread-boards and logic analyzers. Although hardware design has progressed in abstraction to RTL, the methodologies reflect their origins from the structural world. There are significant benefits to an approach that is derived from the software world.

The DK1 design suite has the look and feel of a software environment. The debugger provides in-depth features normally found only in software development. These include breakpoints, single stepping, variable watches, and the ability to follow parallel threads of execution. The hardware designer can step through the design just like a software design system using this approach. Co-simulation and verification facilities are built into the tool-chain, facilitating co-design with instruction set simulators, VHDL simulators such as ModelSim. A key benefit of this is that hardware/software partitioning decisions can be changed at any stage in the design process.

Synthesis in this design system correlates to software compilation in that it is very fast; software designers are accustomed to compiling changes to their designs very quickly and testing the results. This enables them to take many turns, make smaller changes and quickly recompile. The speed of Celoxica’s design system brings this benefit to hardware design as well.

Predefined Hardware Libraries

Predefined libraries much like the standard libraries of ANSI-C and other software environments to hardware design create opportunities for simplifying the development of new functionality as well as encouraging design reuse. As well as relying on third parties and other teams for IP, users have the ability to create their own functional blocks and import others.

Such an approach can combine the ability to easily address registers, memory and peripherals. Built into handle-C are capabilities for accessing internal and external memory as well as registers. Via libraries of predefined functions, common APIs shield users from low level interfaces to ease the integration of FPGAs to physical resources including both peripherals and processors – the latter enabling hardware/software codesign. It is an approach that can mean significant time savings, giving the designer more time to concentrate on core functionality.

This truly reflects the value and efficiency inherent to working at a higher level of abstraction. Much of the work has already been done for the hardware designer, so the design requires much less code and design effort. This also means that the hardware designer can do many more experiments to try out many more ways of implementing a solution. In other words, the designer can explore a larger design space.

A Real-World FPGA Design Example

A European Commission funded “ESPRIT project” called SEHaD (Software Engineering for Hardware Design) afforded Celoxica the opportunity to demonstrate the value of Handel-C. As part of the project, two Ericsson design teams participated in a parallel design effort to develop an IPv6 header compression function on an FPGA. One team used Handel-C and the other a traditional hardware design methods and tools (Verilog/Leonardo).

The Application

Routers are critical components of Internet Protocol networks. In order to handle the increasing demands on performance, modern routers are increasingly based on hardware implementations of the protocol stack. For transfer of real-time information such as voice over the Internet, the payload-to-packet size ratio is unsatisfactory in terms of both real-time characteristics and bandwidth utilization. IPv6 has accentuated this problem further. In order to alleviate this for point-to-point links, schemes to represent unchanging parts of the headers in a packet stream by an 8 to 16-bit number (CID) have been devised. The compressed header contains this number and absolute or delta representations of non-constant data fields of the headers in the stream. The reverse operation is called decompression and is performed at the receiving end of the link.

On each Internet link, many sessions are transmitted simultaneously. Each session payload-to-packet ratio. The idea behind header compression is to enable the transmitting and the receiving router to agree on a short representation of the headers. The headers are compressed to the short form when transmitted and decompressed to the original form when received. This reduces the required bandwidth of the link and improves real-time characteristics of the data stream. As an example: Compressing IPv6, half-rate voice packets will lead to a 65% bandwidth saving. consists of a stream of IP packets with almost identical headers and often a poor payload-to-packet ratio. The idea behind header compression is to enable the transmitting and the receiving router to agree on a short representation of the headers. The headers are compressed to the short form when transmitted and decompressed to the original form when received. This reduces the required bandwidth of the link and improves real-time characteristics of the data stream. As an example: Compressing IPv6, half-rate voice packets will lead to a 65% bandwidth saving.

The Results

Two designers comprised the team using Handel-C. One had some experience using an earlier version of Handel-C and some design experience in the application area, while the other, a college graduate, had no experience with Handel-C or the application.

The table below summarizes the results of the Handel-C implementation compared to the results in the parallel project.

	Handel-C/ DK1	Verilog/Leonardo
Design time	4 man-months	12-16 man-months
Program size	40 pages	200 pages
Compile time	3 minutes	1.5 hours
Size (Virtex 1000-4)	35 % logic, 30 % memory*	NA
Size (Virtex 2000E-8)	17 % logic, 15 % memory*	All logic, all memory**
Speed (Virtex 1000-4)	28 – 33 MHz	NA
Speed (Virtex 2000E-8)	44 MHz	49 MHz

* Distributed memory. No block memory used
** A conscious choice. Used all logic to increase speed. All block memory used.

The most striking result is the difference in design time, a factor 3-4 shorter design time using Handel-C, with similar results in terms of speed and a greatly reduced area. An analysis of the results of the project attributed the difference in design time to three key advantages demonstrated by Handel-C: its support for sequential logic, its compact representation of functionality, and its software-like design methodology which provided fast turnarounds in the design environment.

According to the SEHaD report, these results were consistent with the experience of other users in the project, as well as of smaller examples done by Ericsson. "The difference in program size reflects the conciseness of the C syntax and the higher level of abstraction of Handel-C. An example of this is the style of describing serial/parallel code in Handel-C versus controlling sequencing using finite-state machines in Verilog or VHDL. The small program size and algorithmic expressiveness of Handel-C made it easy for us to make drastic experimental changes to the Handel-C code. This allowed us to explore a larger solution space."

Another major difference commented on in the report findings involved the software-like design style supported by Celoxica's DK1 design suite, which integrates project management, simulation, synthesis and optimization into a single development environment. "Verification was primarily done using the software debugger paradigm. Compilation to real hardware was done very early and after each major addition of functionality using an incremental design style. This created confidence in the solution and enabled us to run large test sets in real time."

Conclusions

For designing functionality of similar complexity, the world of software design is simpler than its hardware counterpart. The possibility of using software design methodologies to create hardware solutions may seem almost heretical. But as demonstrated in the above example it resolves many issues inherent to modern IC development. It shortens design time by a factor of 3-4 times, the language and design systems are easily adopted and the small efficient code makes radical system-level changes simple.

In the software design paradigm, we have what we need to raise the level of abstraction sufficiently to describe in the briefest possible way the desired function rather than its underlying structural detail and overcome many of the difficulties and inefficiencies in contemporary hardware design.

Revision History

Date	Version	Revision
August 2002	1.1	New Layout And Contact Details