CURRENT STREAMS-C FOR THE EPXA10DB, A GENERIC TOOL FOR THE DEVELOPMENT OF COMPLEX CONTROL AND SIGNAL PROCESSING SYSTEMS

Scott Miller
New Mexico Institute of Mining and Technology

Abstract: The Altera EPXA10DB is a development platform for reconfigurable computing, coupling an ARM microprocessor and a modest Altera Programmable Logic Device (PLD) on the same chip. Streams-C is a "C" style language that addresses hardware and software generation by extending the C language. A new port of Streams-C for the EPXA10DB is being developed to provide a standalone, custom-hardware-accelerated computing platform. This may prove to be useful in rapidly developing complex signal processing systems.

I. INTRODUCTION

Reconfigurable computing is a powerful technology that blurs the distinction between hardware and software, allowing software to program hardware, hardware to implement software. No longer is it necessary to have single application-specific integrated circuits; a Programmable Logic Device (PLD) can be programmed to be a vast range of different circuits. Unfortunately, the partitioning of any given algorithm into parts that run as software and parts that are hardware at different times is a collection of NP-hard problems[1]. This creates a greater problem; although a reconfigurable system provides great potential, if no usable development system exists no one will use the system.

In the recent past, engineers made use of circuit-level development tools to implement algorithms. This provided exacting control over the design, but became clumsy and inefficient as the size of the problems increased[2]. To address this issue, tools were developed to provide a graphical interface to this process as well as to make use of circuit libraries. Several of these tools took the form of high-level computer languages. The Very High Speed Integrated Circuit [VHVIC] Hardware Description Language (VHDL)[3] creates an abstraction from the circuit-level design, yet can be readily compiled into a circuit-level implementation.

Unfortunately, it remains difficult, even with tools such as VHDL, to develop and debug parallel applications for PLDs. A higher level of abstraction is needed, one that provides general synchronization and communication among processes. The Streams-C language provides this solution, allowing transparent access to many of these constructs through a familiar C-style language interface[4]. This access takes the form of compiler directives in standard C comments as well as several library functions. By hiding the underlying implementation of the hardware and software partitioning, development of complex algorithms to utilize the power of the available hardware is greatly simplified.

The hardware platform used for this research is the Altera EPXA10DB, a development board that provides an Excalibur part, a standard microprocessor coupled to a PLD, as well as many physical interfaces. These interfaces can include an Ethernet connection, two Universal Asynchronous Receive/Transmit (UART) connections, and a high-speed serial interface. With these interfaces acting data sources, the Streams-C interface can be used to make this platform a generic tool for the development of complex control and signal processing systems.

To clarify the distinction between the portions of algorithms that run mainly as a hardware implementation in the Excalibur’s PLD and the portions that run mainly as a software implementation on the Excalibur’s ARM microprocessor, the terms HW and SW will be used respectively. This paper highlights the features of the EPXA10DB as an embedded, standalone system, the current progress in the development of Linux for the EPXA10DB, and the current state of Streams-C.

II. DISCUSSION

A. The Streams-C Language

The Streams-C Language is an extension of the C language to handle the partitioning of parallel tasks into HW and SW. Specifically, the sc2 implementation of Streams-C[2] developed by Maya Gokhale at the Los Alamos National Laboratory is being ported to a new platform for this development project. The sc2 compiler makes use of Linux-pthreads and part-specific hardware cores to provide stream-based communication among concurrent processes in HW and SW.
The original sc2 implementation was developed for use on targets such as the Annapolis Microsystems Firebird, a PCI or VME board operated by a standard PC[4]. This board features multiple, interconnected Xilinx Virtex PLDs as well as SRAM local to the board. To interface to the HW, the Firebird communicates via the host PCI bus to the SW processes.

It is important to note that Streams-C does not automatically parallelize a given source program nor does it automatically partition HW and SW. Ultimately, the language provides a set of libraries accessible through compiler directives and “runtime” function calls to accomplish this task. These libraries provide for processes, streams, and signals[4].

Processes are the major unit of execution, a unit with local memory, registers, and other resources. These units run independent of other units and rely on streams and signals to communicate. The stream provides a high-bandwidth interconnection used to synchronously transfer information. A signal is used to communicate low-bandwidth, asynchronous information useful for synchronizing process execution, starting processing, etc. All of the above can be executed as HW, SW, or a combination thereof.

A typical Streams-C source file is presented in Figure 1. This file is a part of the test suit of files used to test the original sc2 compiler. Note that the sc2 compiler directives begin with three forward slashes. Processes are created as SW by default but can be put into HW by setting the process type to “HP”. After declaring the various processes, the connections among the various processes need to be enumerated for the sc2 compiler.

The process used to convert a Streams-C source file is illustrated in Figure 2. The sc2 compiler is immediately responsible for separating the program into a software binary (an executable file) as well as the necessary hardware configuration. These two files are then transferred over an Ethernet connection to the EPXA10DB flash memory. This connection is convenience afforded by the operating system discussed in the next section; the operating system allows the EPXA10DB to mount to a standard Network File System (NFS) share and transfer it into its own Version-2 Journaling Flash File System (JFFS2)[5]. Once on the EPXA, a simple shell loading script programs the hardware and then starts the software binary.

Figure 1 An excerpt from one of the test programs provided with the original sc2 compiler (strm1.sc). This illustrates the general structure and syntax of a Streams-C program.
B. Implementation of sc2 on the Excalibur’s ARM Microprocessor

The implementation of sc2 on the Excalibur’s ARM processor had three major requirements. The first requirement was that a working version of the Linux operating system be present. The second requirement was the sc2 simulation library, a library to allow Streams-C programs to be executed entirely in SW (no HW processes.) The third requirement was the sc2 runtime (Sim/RT) library, allowing SW processes and providing the communication necessary to HW processes. The third requirement will be discussed in the next section on communication.

Because the EPXA10DB is a relatively novel platform with respect to Linux, modifications of the readily available versions for use on the EPXA10DB were used. Specifically, the Linux 2.4.19 Kernel with the ARM-Linux “-rmk7” patch provides the base kernel[6]. To load the kernel and start the ARM processor from the EPXA10DB’s on-board flash memory, a modified version of ARMBot is used[7]. This, when coupled with a root filesystem, satisfies the prerequisite needs for sc2.

As the sc2 simulation library is not hardware-dependant, this library needed only to be recompiled for the EPXA10DB’s operating system. This was not done on the EPXA10DB, as it was much more convenient to use the same cross-compiling toolchain used for the development of the EPXA10DB’s Operating System.

Tests of the simulation library were focused on benchmarking the performance of the stream communication. For a program with two processes, one looping back a stream, the SW/SW communication performance was measured to be sustained at 6.9 kilobytes per second.

C. Communication between the ARM Microprocessor and the PLD

As Streams-C provides transparent communication among processes, high performance of the communications implementation is crucial to making Streams-C a useful tool. This task is still only in the design phase of development and is not currently implemented for the EPXA10DB.

The original sc2 communication was implemented as a number of registers accessed via a PCI bus. Because of the limitations imposed by the target’s communications, several register writes were required to communicate a single byte of data. To receive data from the target, the registers need to be continuously polled; no other mechanism to inform that new data was available. One of the major advantages to using a part such as the Excalibur is that the processor and PLD are very tightly coupled. As a number of communications interfaces exist on the Excalibur, the communication required by Streams-C could be implemented in many different ways.

The method chosen was to make use of the Altera’s proprietary Altera Hardware Bus (AHB) communications to create a number of registers. This method, from the sc2 compiler’s view, is almost identical to the original implementation. However, the library implementation can use the features of the AHB to eliminate the need to poll for new data.

D. Implementation of sc2 on the Excalibur PLD

Perhaps the most powerful section of the sc2 compiler is its ability to compile a subset of the C language into Register-Transfer-Level (RTL) VHDL. Once at this level, the hardware-independent RTL VHDL can be quickly compiled for a specific hardware target. Although the tools for this code generation were successfully compiled, the results could not be tested. The sc2 HW processes require a hardware core library that provides communication as well as several hardware control structures; the HW/SW communications remains
in design as documented in the previous section.

III. CONCLUSIONS

This system shows significant promise but still requires much development. However, the benefits of having a familiar “C” interface to develop complex systems are significant:

- Hardware designs can directly incorporate a fully-featured operating system. As the SW processes are run in Linux user-space, they can pass data to other computers via standard networking protocols, easily log data to any file system mounted on the device (including network filesystems,) and can also make use of the protected, dynamic memory management.
- Developing hardware-accelerated designs does not require knowledge of circuit-level techniques.
- Systems can make use of readily available tools and development environments for the “C” language.

ACKNOWLEDGEMENTS

The Author wishes to thank Mike Boorman and Maya Gokhale of Los Alamos National Laboratory for providing this research opportunity, Lorie Liebrock and Kevin Wedeward for supporting his research at New Mexico Institute of Technology, David Dubois of Los Alamos National Laboratory for hardware design advice, Iain Scott and Clive Davies of Altera Inc. for support with the EPXA10DB, as well as Russel King and the rest of the ARM-Linux developers.

REFERENCES


Scott Miller is currently an undergraduate student of the New Mexico Institute of Mining and Technology. He is pursuing a BS in Electrical Engineering, a BS in Computer Science, and a minor in Mathematics. He currently is a Software Engineering Intern for General Dynamics Decision Systems in Phoenix, AZ and still works closely with the Los Alamos National Laboratory CCN-5 Special Projects Team.