A Web–Based Multiuser Operating System for Reconfigurable Computing

Oliver Diessel, David Kearney, and Grant Wigley

School of Computer and Information Science
University of South Australia
Mawson Lakes SA 5095
{Oliver.Diessel, David.Kearney, Grant.Wigley}@unisa.edu.au

Abstract. Traditional reconfigurable computing platforms are designed to be used by a single user at a time, and are acknowledged to be difficult to design applications for. These factors limit the usefulness of such machines in education, where one might want to share such a machine and initially hide some of the technical difficulties so as to explore issues of greater value. We have developed a multitasking operating system to share our SPACE.2 coprocessing board among up to 8 simultaneous users. A suite of pre–configured tasks and a web based client allows novices to run reconfigurable computing applications. As users develop a knowledge of the FPGA design process they are able to make use of a more advanced PC client to build and upload their own designs. The development aims to increase access to the machine and generate interest in the further study of reconfigurable computing. We report on the design, our experience to date, and directions for further development.

1 Introduction

Definitions of reconfigurable computing are currently unclear. Rather than clarify the issue, we define usage of the term as it relates to this paper as describing the computations one performs on computers that include reconfigurable logic as part of the processing resource. The reconfigurable logic presently of interest is some form of Field Programmable Gate Array (FPGA) technology. Current reconfigurable machines may have the reconfigurable resource tightly coupled with a von Neumann processor integrated on a single chip, or loosely coupled to a von Neumann host via a general purpose bus such as PCI. While the coupling does influence performance in different application domains, the reason for employing reconfiguration is the same for both, namely, to speed up computations by implementing algorithms as circuits, thereby eliminating the von Neumann bottleneck and exploiting concurrency. In tightly coupled machines it may be viable to reconfigure on–chip logic at a fine grain of computation — perhaps even at the instruction level. However, the overheads of communicating over relatively low bandwidth buses demand that loosely coupled machines be reconfigured at a coarser grain. Applications on this latter class of machine are generally implemented as a front–end program executing on the sequential host,

and a sequence of one or more configurations that are loaded and executed on a reconfigurable coprocessing board.

Loosely coupled reconfigurable coprocessing boards such as PAM [5], Splash 2 [2], and SPACE.2 [3] are single tasked devices despite usually being attached to hosts running multitasking operating systems. A possible reason for this is that most FPGAs are programmed in a slow configuration phase that establishes the circuitry for the whole chip at once. It is therefore easier to control performance and access if it is done for a single task at a time. However, the advent of dynamically reconfigurable FPGAs such as the Xilinx XC6200 and Atmel AT6K families, which allow part of the FPGA to be reconfigured while the rest of the chip continues to operate, has increased the interest in techniques and applications that exploit these chips' facility for multitasking.

Early work in this area was carried out on the loosely coupled DISC computer [6], which made use of a well–defined global context to allow the reconfigurable logic to be reconfigured and shared by multiple relocatable tasks during the run–time of an application. However, use of the DISC array is restricted to a single task at a time to avoid contention on globally shared control lines. More recently, the tightly coupled GARP processor [4] allows multiple configurations to be cached within the reconfigurable logic memory and supports time slicing on the controlling MIPS processor that is integrated with the reconfigurable logic array. Multiple users and multiple tasks thereby appear to be supported, but in fact only one configuration at a time may execute lest multiple configurations contend for control signals. Brebner described the issues involved in managing a virtual hardware resource [1]. He proposed decomposing reconfigurable computing applications into swappable logic units (SLUs), which describe circuits of fixed area and input/output (I/O) interfaces, so that multiple independent tasks might share a single FPGA. Brebner described two models for allocating the resources of the FPGA. The sea of accelerators model admits arbitrarily sized rectangular tasks and is suited to independent tasks of varying computational needs. On the other hand, the parallel harness model, which partitions the resource into fixed sizes, was thought to be more appropriate for cooperating tasks. We examine the implementation of a parallel harness model that partitions an array of FPGAs at the chip level. Our SLUs occupy an entire chip, and currently operate independently.

Our interest in developing a multiuser operating system for reconfigurable computing is motivated by the desire to increase accessibility to a SPACE.2 machine for CS and EE classes within a university environment. A second thrust of this work is to provide abstractions that simplify the programming and use of reconfigurable computers for novices. To this end, we have designed a web–based interface to SPACE.2 that allows multiple users to execute pre–configured applications at the same time. Users familiar with the FPGA design flow are also able to design and upload their own applications within a PC environment. Both are based on a simple multiuser operating system that partitions the FPGAs on a SPACE.2 board to allow up to 8 simultaneous users. This operating system provides the interface to web–based clients and manages the allocation of FPGAs

within the SPACE.2 array to user tasks. We report on the development to date and the future direction of the project.

The remainder of this paper is organized as follows. Section 2 describes the architectural and applications development features of SPACE.2 that impact on the design of the multiuser operating system. Section 3 presents the design and reports on the development of the multiuser operating system and web–based clients. Our findings so far are reported on in Section 4. Section 4 also proposes remedies for overcoming the limitations experienced with the current design. The conclusions and directions for further work are presented in Section 5.

2 The SPACE.2 architecture

The SPACE.2 system consists of a DEC Alpha host into which one or more SPACE.2 processing boards are installed as 64 bit PCI localbus slaves [3]. The host processor communicates configuration and application data over the PCI bus to individual boards. Processing boards can operate independently of each other, but for particularly large applications they can also be connected together over a secondary backplane to form extensive arrays.

The compute surface of a SPACE.2 processing board consists of a 4×2 array of Xilinx XC6216 FPGAs. In addition, the board provides control logic for interfacing the configurable logic to the PCI bus, on–board RAM, and a clock module.

Limited connectivity to I/O blocks in the XC6216 makes a seamless array of gates spanning multiple FPGAs impossible. The devices are interconnected in a 2-dimensional mesh, with a pair of adjacent chips sharing 32 pads in the east-west direction and a column of 4 chips sharing 41 pads in the north-south direction. With appropriate pad configurations, opposing edges of the mesh may be tied together to create a toroidal structure. Access to the on–board RAM is available from the northern and southern edges of the FPGA array.

Several global signals are distributed throughout the array. Devices are programmed via a global 16 bit data bus. This bus is also available for FPGA register I/O. The appropriate chip is selected by the control logic, while mode, configuration, and state registers are selected through an 18 bit address bus. Three common clock frequencies are set in the clock module and distributed with a global clear signal to the FPGA array.

All software on the host interacts with the SPACE.2 processing boards through a character device driver under UNIX. A board is opened as a single–user file to allow a user to make exclusive use of the reconfigurable hardware. Once opened, a board's FPGA configuration and state can be read or written to by an application program.

2.1 Current design flow

SPACE.2 applications typically consist of two parts: a circuit design for configuring the FPGAs of a SPACE.2 board and a host program that manages I/O

and controls the loading, clocking, and interrupt handling for the circuit. Applications initially therefore present a problem in hardware/software codesign.

When the hardware portion of the design has been identified, its design is elaborated in two stages. In the first stage, the logic of the required circuits is derived. Then, in the second stage, the logic is placed onto the FPGAs and its interconnections are routed.

The front–end or host program is responsible for loading the design onto the board, communicating with the running application, and responding to user interrupts. Configurations cannot be tested by simulation due to a lack of tools for simulating multi–chip designs. Circuits are therefore not tested until they are loaded onto SPACE.2. Debugging is then performed by reading the contents of registers while the circuit is live. This may necessitate modifying the original design to latch signals for debugging purposes.

Apart from the difficulty of recognizing opportunities for exploiting parallelism in problems, the current design flow inhibits experimentation with the SPACE.2 platform in several ways. Due to a lack of adequate abstractions, applications development requires the understanding and management of many low–level details. Moreover, we lack the tools to make this task easier. For example, designers need to partition tasks manually, and there is limited support for testing and verifying designs. In order to debug designs, a detailed knowledge of the system is needed. It should be noted that these problems are not unique to SPACE.2 but are generally accepted as barriers to the wider adoption of reconfigurable computing as a useful paradigm. One of the roles of our operating system is to hide as much detail as possible from the novice user.

3 The multiuser environment

In order to introduce reconfigurable computing to novice users and to develop their interests, we embarked on a project to develop two internet accessible interfaces and a multiuser operating system for the SPACE.2 machine. One of these interfaces provides a simple form for the user to select and run one of several pre–configured applications for the machine. This interface is available from common WWW browsers. As the user's knowledge of the FPGA design flow develops, they may upload their own designs from a network PC–based interface. Managing the requests of these two clients, the multiuser server establishes socket connections using TCP/IP, vets user authorizations, allocates FPGAs to users, loads user tasks, and handles user I/O to tasks.

3.1 Web–based demonstration platform

The web–based client consists of a CGI script and forms–based interface that contains a number of pre–configured designs. Users may choose to execute one of a number of simple applications, supply their own data, and obtain results upon completion of processing. The client thus abstracts the details of designing

and running an application. With this interface we also hope to develop clear demonstrations of what the SPACE.2 board is capable of.

The client attempts to establish a connection with the server after the user selects an application and provides input data. In contrast to the PC–based client, no special authorization is required for web–based users. If an FPGA is available, the server provides a connection, and accepts an application in the form of a Xilinx CAL file one line at a time. This file is translated into global coordinates and loaded by the server. Thereafter the user input values are written to the appropriate registers. The simple applications developed so far return results to the user within a predefined number of clock periods. Event–driven applications are not yet catered for. After the results have been obtained, the FPGA resource is freed by the server, and the socket connection to the client is broken.

3.2 Support for applications design

Loading designs onto the SPACE.2 processing board and interacting with them is tricky since it involves lengthy sequences of control functions. When users begin to design their own applications, they should not be distracted by these details. We therefore developed a PC–based Tcl/Tk client to abstract away the complexity of loading and interacting with tasks. This client presents a mouse–driven graphical user interface that can be used from any PC on the internet that uses the Windows 95 operating system. Anybody may request a copy of the client by filling out a web–based form that is processed by the system's administrator in order to establish an entry for the user in the host's password file. The user is then provided with instructions on how to download and install the client by email.

The Tcl/Tk GUI interacts with a dynamic link library (DLL) that also resides on the PC for parsing requests between the client and the server. Use of the DLL provides communications independence between the client and server, thereby allowing either to be more easily replaced. The client provides an authentication procedure by sending a username and password to the server for checking against the host's password file. After a connection is established, the client sends the user's design to the server for translation and loading and sends the user's data for writing to the input registers. When requested to by the user, the server retrieves the application results from the output registers and passes them to the client for display.

Successful SPACE.2 designs must resolve several complex design issues. To overcome some of these problems we have chosen to constrain user designs in a number of ways: (1) We make use of the XC6200 family's support for bus–based register I/O. A "register" may be viewed as a virtual pins abstraction since the user need not worry about interfacing designs to particular pins. (2) The number of registers is limited to a maximum of eight for input and output respectively. Registers are 8 bits wide. The eight input registers are aligned in the leftmost column of an FPGA, and the eight output registers are aligned in the rightmost column. (3) The user design must be able to be fully loaded onto a single FPGA,

thereby eliminating multi–chip partitioning problems. (4) The application must be designed to a fixed global clock speed.

Adding these constraints to applications limits the range of designs that can be created, but simplifies their complexity considerably.

3.3 Operating system design

Normally only one user at a time can gain access to a SPACE.2 board. However, with the possibility of several people wanting to gain access to the machine at the same time, a multiuser operating system is required. Currently, while a designer is using a SPACE.2 board, all other attempts to open the device driver are blocked by the kernel. The multiuser operating system removes the blocking semantics by adding an additional layer to the device driver. The system is modelled on a multiprocess server that accepts up to eight simultaneous socket connections — one for each chip.

The operating system initializes the socket connection code and establishes a file–based allocation table for the 8 FPGAs on the board. The server then listens on a designated port for connection requests that are initiated by the web or PC–based clients. When a user connects to a socket, the operating system checks whether the SPACE.2 board is available for use. This occurs in two stages. If the server does not yet have control of the SPACE.2 device driver, it attempts to gain control in order to determine whether any other user has dedicated access to the SPACE.2 board. After control of the device driver has been gained, the FPGA allocation table is checked for an available chip. If one is found, the server then forks off a process for the new user, and waits for further socket connection requests. If either stage fails, the request is refused and the client will wait until the user attempts to reconnect.

The system validates PC client users against the host's password file and determines the user's privileges (super or normal) before allocating an FPGA to the user. The user's process ID, name, and priveleges are stored in the allocation table for easy reference and to avoid contention. A super user has similar privileges to those of a UNIX root user. This user has access to all user commands as well as simple "house–keeping" functions such as altering the common system clock speed and removing users from the system. Normal users do not have access to these global commands. The list of super users is maintained in a separate file.

The multiuser operating system manages FPGA allocation and deallocation on a first come, first served (FCFS) basis. In the conventional single user operating environment, this function is explicitly handled by the user's front–end program. Once allocated, a chip is not relinquished until the socket connection is broken. The connection to a web–based client is broken after the results from a run have been obtained. A PC–based client is disconnected at the user's request.

The operating system is responsible for loading applications and handling I/O for the applications. A 3 digit code is used for communicating requests between the clients and the server. Requests consist of three parts: the code, followed by an integer representing the number of arguments to follow, and a

list of arguments. The server must acknowledge each part of the request for the communication to proceed.

The application designs are uploaded in the form of a Xilinx CAL file, which can be loaded onto any XC6216 chip. The file is loaded up one line at a time and a special flag is sent to indicate EOF. Row and column addresses in this file are translated into the global SPACE.2 coordinate system so that the design is loaded onto the correct chip. Similarly, I/O requests are parsed so as to address the correct set of cells in the global coordinate system.

4 Performance review

4.1 What sorts of tasks work?

The interface constraints imposed by our clients limit the range of practical tasks that can be performed by the FPGAs. Nevertheless, the current environment does not preclude users from experimenting with reconfigurable computing. Indeed, we see it as an entry point for education, and believe there is ample scope for devising educational projects that fit within the constraints of the interface. Suitable pre–configured designs include simple bit manipulation tasks and more complex neural net classifiers and distributed multipliers. Moreover, it is a matter of designing alternative client interfaces to remove the restrictions on I/O since the multiuser server has the full capability of the device driver at its disposal.

Since all requests go through the original single user device driver, which services them sequentially, it is possible that performance will suffer. The server is therefore not practical for tasks that expect performance guarantees such as real–time tasks.

SPACE.2 is a platform designed for experimentation. The problem of providing a multiuser run–time environment for SPACE.2 requires solutions to numerous issues. These include: the design and compilation of suitable applications, the interplay of time– and space–sharing, what the granularity of swapping should be, and how the device driver and operating system kernel should be designed. We have just begun to explore some of these issues. There is considerable work to go on with.

4.2 Problems and potential solutions

The current implementation of the multiuser operating system suffers from several limitations. In this section we discuss these problems and propose what we perceive to be workable solutions.

The current Tcl/Tk client interface for user designed tasks limits the number of input and output registers to eight of each, and the width of these registers is set to 8 bits. Moreover, input is from the leftmost column and output is to the rightmost column of an FPGA. It may be desirable to have more registers, and that it be possible to interface with these at arbitrary locations, in particular,

as an aid to debugging. A more flexible client interface would overcome this limitation.

Register I/O is acceptable for tasks involving low bandwidth transfers, but for high bandwidth streaming applications it causes overheads that could be avoided. Unfortunately for such applications, use of the on–board RAM is not supported in the current operating system. However, use of the RAM could be supported by applications that make use of the pair of chips at the northern edge of the board, which would leave the rest of the board free for tasks that make use of register I/O.

Another area where more support from the client interface may be necessary is dynamic reconfiguration. While the SPACE.2 board and device driver allow dynamic reconfiguration, the multiuser environment does not currently support it. Tasks are therefore limited in size to a single XC6216 chip. It should be possible to provide hooks and conditions for reconfiguration within the client interface. Alternatively, the operating system could be designed to interface with conventional front–end host programs that control the reconfiguration.

Future enhancements to the operating system should also consider more adaptive space–sharing schemes so as to allow allocating tasks in multiples of a single chip.

At present up to 8 simultaneous users are supported by partitioning the logic resource of a SPACE.2 board at the chip level. Any additional users are blocked from access to the board. To overcome this problem we propose implementing a time–sharing mechanism in the future. Time–sharing would place further constraints on tasks, such as ensuring that all intermediate results are latched before commencing a swap. However, having 8 chips available for swapping ensures a reasonable period over which to amortize the costs of swapping if they are carried out in a round–robin fashion. To reduce the overheads of swapping, it would be desirable to modify the board controller to enable on–board storage of configurations and state.

If time–sharing and swapping is to be supported, it may be necessary to investigate multithreading the device driver to reduce blocking, and to implement a scheduler to reduce the unpredictable delays that can result with the FCFS servicing of requests imposed by the current device driver.

Finally, the fixed global clock speed places an undue constraint on performance. It would be nice to find a solution to the problem of frequency–sharing the board — having multiple tasks of different clock speed sharing the board. To this end, we propose investigating supporting user programmable clock divider circuits within each chip.

5 Conclusions

We have described the initial design of a multiuser server for our SPACE.2 reconfigurable computing platform. This system partitions the array of eight FPGAs available on a SPACE.2 processing board among multiple users, one per user, so as to allow multiple independent tasks to execute simultaneously. This approach

increases utilization and access to the system. However, our solution is rather coarse grained considering that the XC6216 chips used in the array are partially reconfigurable at the cell level. The resource is also under–utilized if less than 8 tasks are running. Moreover, the PCI bus and board device driver form a sequential channel for I/O to the array. Tasks are therefore loaded one at a time, and I/O to tasks is performed sequentially. A further limitation imposed by the board on the initial design is the need to share a common clock. To overcome these problems we intend supporting adaptive partitions and implementing a time–sharing scheme. We will investigate implementing programmable clock divider circuits on each chip to allow the board to be shared in the frequency domain. Further enhancements envisaged for the user interface are intended to facilitate the use of dynamic reconfiguration and support more flexible I/O.

Acknowledgments

The helpful comments and assistance provided by Matthew Altus, Bernard Gunther, Ahsan Hariz, Jan Machotka, Manh Phung, Joseph Pouliotis, and Karl Sellmann are gratefully acknowledged.

URL

Access to the server and documentation is available through links from the URL `http://www.cis.unisa.edu.au/acrc/cs/rc/multios`.

References

1. G. Brebner. A virtual hardware operating system for the Xilinx XC6200. In R. W. Hartenstein and M. Glesner, editors, *Field–Programmable Logic: Smart Applications, New Paradigms and Compilers, 6th International Workshop, FPL'96 Proceedings*, pages 327 – 336, Berlin, Germany, Sept. 1996. Springer–Verlag.
2. D. A. Buell, J. M. Arnold, and W. J. Kleinfelder, editors. *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society, Los Alamitos, CA, 1996.
3. B. K. Gunther. SPACE 2 as a reconfigurable stream processor. In N. Sharda and A. Tam, editors, *Proceedings of PART'97 The 4th Australasian Conference on Parallel and Real–Time Systems*, pages 286 – 297, Singapore, Sept. 1997. Springer–Verlag.
4. J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In K. L. Pocek and J. M. Arnold, editors, *The 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97)*, pages 24 – 33, Los Alamitos, CA, Apr. 1997. IEEE Computer Society.
5. J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. H. Touati, and P. Boucard. Programmable active memories: Reconfigurable systems come of age. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(1):56 – 69, Mar. 1996.
6. M. J. Wirthlin and B. L. Hutchings. Sequencing run–time reconfigured hardware with software. In *FPGA'96 1996 ACM Fourth International Symposium on Field Programmable Gate Arrays*, pages 122 – 128, New York, NY, Feb. 1996. ACM.