A Hardware/Software Co-design Environment for Reconfigurable Logic Systems

Gordon M^{c}Gregor, David Robinson and Patrick Lysaght

Dept. of Electronic and Electrical Engineering,
University of Strathclyde,
204 George Street,
Glasgow, G1 1XW
United Kingdom

FAX: +44 (0)141 552 4968
email: g.mcgregor@eee.strath.ac.uk

Abstract. The design of reconfigurable systems is currently an area of intense interest within the programmable logic community. One of the main obstacles to the widespread adoption of this technology is the relative immaturity of the associated design methodologies and tools. This paper reports on a development environment and associated tools for the implementation and debugging of reconfigurable applications. These tools are part of a larger program of work to improve the construction of reconfigurable systems. The current environment is specific to a custom Xilinx XC6200 development board, but the principles are generic and portable to other reconfigurable systems and architectures.

1. Introduction

The development of reconfigurable applications typically proceeds in an ad-hoc manner, as there are very few CAD tools to accelerate or simplify the design process. If reconfigurable logic techniques are to be more widely accepted, application development procedures have to be better understood and become more structured. The current lack of CAD tools results in most examples being handcrafted [1], [2]. The consequent disadvantages are limited re-usability, restricted testability and the re-invention of design approaches. To address these issues new tools and methodologies are being researched to reduce the design effort involved in developing reconfigurable designs.

A design environment has been developed to compliment research into the simulation of reconfigurable systems. The Dynamic Circuit Switching (DCS) simulation tool [3] allows the simulation and specification of a reconfigurable design in its entirety. The interactions between reconfiguring and static circuits are modeled, along with normal

circuit operation. With DCS, new designs can be investigated and solutions iterated upon much earlier in the design cycle than has been possible with other methods [4], [5], [6]. The amount of effort expended in evaluating candidate designs for implementation is thus greatly reduced. The original DCS tools are in the process of being enhanced and expanded to aid in several new areas of the design flow. Work has proceeded on methods to simplify and accelerate the co-design of software and hardware that interact with and control the reconfigurable application. The integration of these various tools provides a framework for the development of complete reconfigurable logic solutions from specification through to final implementation.

The paper is organised into five further sections. The next section presents an overview of the development environment and the design flow. The middle sections describe in more detail the component parts of our current system and the important features of the software API (Application Programming Interface). Section five presents an Interactive Design Tool (IDT) that builds upon the API to allow prototyping and debugging of the software and hardware routines. Conclusions and future work are discussed in the final section.

2. Overview of Development Environment

The underlying goal of the research was to provide a flexible and re-useable environment for the rapid investigation of dynamically reconfigurable applications. The development environment has been designed as a research tool, to allow the rapid investigation and production of proof-of-concept reconfigurable applications. The main objective was to raise the level of the development process above the level of the communication, configuration and management tasks that are inherent in reconfigurable systems. These goals have been achieved, allowing many more designs to be investigated more rapidly then was previously the case.

There are three main components to the development environment. These are the target reconfigurable architecture, a collection of software classes for programmatic control, and an interactive design tool. The target device is hosted on a custom PCI (Peripheral Component Interconnect) plug-in card, developed in the University of Strathclyde [8]. Software control of the card is encapsulated in C++ classes for ease of re-use and to allow abstraction from the implementation detail. The IDT provides the designer with access to the methods (C++ functions) of these classes, allowing the debugging of hardware operation and the development of reconfiguration algorithms.

3. Hardware Sub-System

The target system, shown in **Fig 1**, interfaces to the host via a Xilinx XC4013 FPGA, implementing a LogiCore PCI interface. A Xilinx XC6216 dynamically

reconfigurable FPGA is the target device for reconfigurable applications. There is 1Mbyte of general purpose SRAM on the board, controlled via the XC6216 device. The card has two EEPROMs to allow serial configuration of either FPGA and the XC6216 can be (re)configured via the PCI bus, from configurations stored in the SRAM, or generated locally on either the XC6200 or XC4013 devices. The card is hosted in the Windows 95 operating system.

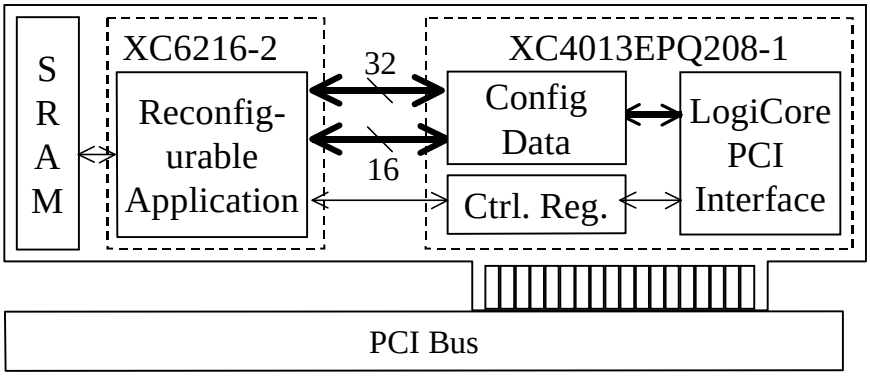

Fig 1 PCI plug-in card

The card communicates using PCI target mode for single data bursts between the card and host. Although initiator mode operation and interrupt services have been developed [7], they are not provided in the current revision of the system. Complete control of the XC6216 device operation is available via the XC4013 FPGA.

The design flow proceeds as follows.

- Designs are initially simulated using VHDL models that have been post-processed using the DCS simulation tools. VHDL simulation is then used to verify the operation of the reconfigurable design.

- When simulation is complete, each reconfigurable task is isolated from the design by hand. Methods to automatically extract reconfigurable tasks from the design files and then synthesise these tasks are being investigated, enhancing the DCS technique [7].

- The Velab VHDL elaborator tools are then used to convert the designs to EDIF netlists for placement and routing using the Xilinx XACT6000 design tools. The layout of reconfigurable tasks typically requires manual intervention, to ensure the consistent placement of logic across configurations. Current APR tools have no concept of layout across configurations so this task is the responsibility of the designer. Manual placement of logic blocks is specified using attributes.

Sections of the array can also be reserved, forcing logic to be routed around these areas. Logic may then be configured into these spaces at a later time. The non-reconfigurable portions of the design are automatically placed and routed as normal.

- The final stage of the design process is the generation of the bitstream files, using the Xilinx design tools.

For some designs, the bitstream generation tools produce very large, redundant configuration files and manual intervention is required to extract the relevant configuration information. The redundancy can be attributed to the APR tools being unaware of the reconfigurable nature of the design. Future tools will have to address these issues.

Improved estimates of removal and loading time [9] can be made from the configuration bitstream sizes and annotated into the DCS simulation. Work is currently ongoing to provide automated methods to generate accurate back-annotated timing simulation, through extensions to DCS.

4. Software API

An object-orientated API has been defined to simplify the control and management of reconfigurable applications. The interface encapsulates and hides the details of accessing the card, allowing work to proceed at a more abstract level when managing reconfigurable applications. The principal contribution of the API is support for event-driven reconfiguration of the target hardware.

The API is composed of three distinct layers. At the lowest level, a VxD (virtual device driver) controls access to the physical hardware, over the PCI bus. The next layer is the Hardware Abstraction Layer (HAL) that provides simple access functions. The Hardware Interaction Layer (HIL) layer builds upon the first two layers and provides more complex functionality. The designer uses a combination of these layers to interact with the underlying hardware. Applications typically interact directly with the HAL for data passing and use the HIL for configuration and reconfiguration management. An overview of the interaction between layers is shown in Fig 2. The IDT is also shown to indicate its relative location in the hierarchy.

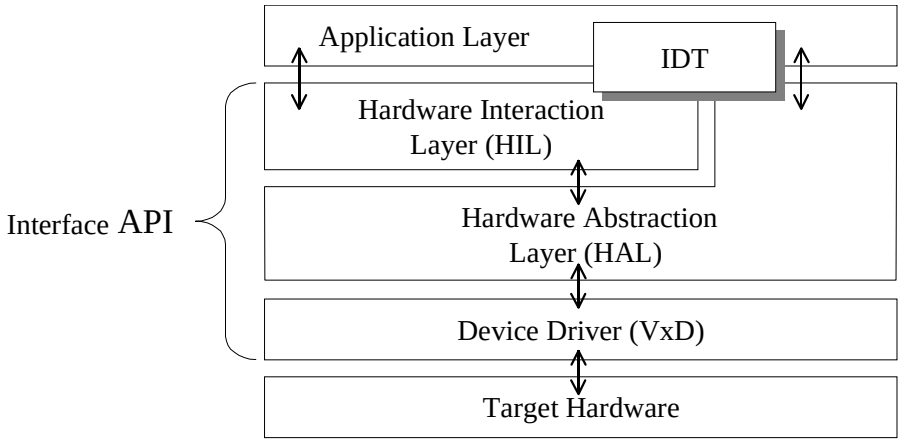

Fig 2 API Overview

The API has been designed to be easily re-targeted to other platforms. For example, support for other XC6200 prototyping boards would require changes to only the VxD layer. Targeting other FPGAs would require modification at the HAL layer but the methods in the HIL should remain substantially unaffected.

4.1. Device Driver / VxD

This layer of the API manages the initialisation and control of the hardware sub-system with the Windows 95 operating system. It obtains and stores configuration information, provided by the operating system, about the resources allocated to the hardware. The information is used by the VxD to mediate during data exchange. Application software cannot communicate directly with the underlying hardware. The VxD provides four hardware-specific methods to the HAL layer, to read and write an arbitrary location on the XC6216, and to read and write to the control register implemented in the XC4013. Although applications can communicate directly with the VxD, the access functions are greatly simplified in the HAL.

4.2. Hardware Abstraction Layer

The Hardware Abstraction Layer extends the basic functionality of the VxD layer. It provides more abstract access to the control register, implementing functions such as device reset, clock single stepping and data transfer. It reduces communications to the underlying hardware by maintaining copies of information on the current state of the device, e.g., map register contents or control register values. The HAL also performs address translation and validation.

Applications interact directly with this layer to pass single data words between an application and arbitrary locations on the target device. Address and data information

are combined within a single object, simplifying the interfaces between methods. The address format is stored as either a numerical value or a symbolic name. The information in the symbol table produced by the XC6000 tools is used to locate named registers on the device. Registers are identified by the name used to instantiate them within the VHDL source code.

The ability to use symbolic register access greatly increases the readability and maintainability of source code. Relocation of interfaces in the hardware has no impact on the supporting application software. Removing the details of the hardware location from the application simplifies the interaction between the software and hardware portions of the design. The HAL maintains a database of the location of registers within the design, which is updated for each new configuration. Development time is also reduced, as the application software does not need to be recompiled as frequently to reflect hardware changes.

4.3. Hardware Interaction Layer

The Hardware Interaction Layer (HIL) extends the functionality of the previous two API layers, to provide configuration control and scheduling services. The main responsibilities of this layer include; configuration management, block data passing, advanced clock control, and reconfiguration control and scheduling.

Configuration management is encapsulated within three classes in the HIL. One of these classes encapsulates the format-dependent details for loading configuration files. Once loaded, the data is stored in an internal format that is used by the remainder of the API. A configuration can be pre-loaded for later use or immediately downloaded to the device. Using this feature, all configurations for an application can be pre-cached from disk to enhance performance. Methods also allow for downloaded configurations to be read-back and verified, taking into account device features such as wild-carding that allow multiple parts of the device to be configured with the same data, in a single access cycle.

Configuration objects are stored within managed list structures. Names are associated with each configuration to simplify retrieval. A configuration management class stores the list of configuration objects in the application and is used to reduce the amount of data transfer to the target device. If a request for a configuration that is currently active on the device is made, the management class suppresses the data transfer. If this feature is not suitable it can be disabled. Suppressing the configurations can only safely be done when it is known that configurations do not overlap, e.g., in applications where the entire device is being reconfigured on each configuration.

The HIL provides methods to enhance the transfer of blocks of data between software and hardware. Data objects are instantiated and associated with a location on the

array. A data object can be either a disk file or a memory block. These are treated identically, exploiting the polymorphism of C++. Blocks of data can be passed between software and the reconfigurable hardware by calling methods in the data objects. The application software does not have to maintain the data structures or track the currently active location.

Support is provided within the HIL to setup and manage event-driven dynamic reconfiguration of the target reconfigurable subsystem. An event is defined by providing a location to inspect on the array and a trigger value. The event is triggered when the value on the array is equivalent to the assigned value. Each event has a vector table of configurations associated with it. When the event is triggered the first configuration in the list is downloaded to the array. The next time the event occurs, the next configuration in the table is used, and so on. The vector tables can be optionally set to loop back to their beginning when exhausted, or simply suspend when the end of the table is reached.

Managed lists of events are stored within the HIL. Event handling within the current system is non-optimal as the trigger values are polled from the array. A list is updated when called by the application software and all events within it are checked. Events are processed in the order in which they occur in the list, allowing priorities to be attached to reconfigurations on a first come, first served basis. More complex scheduling and prioritisation can be developed using multiple lists. The limitations introduced by polling do not represent a significant overhead in the interactive debugging tools, but an interrupt scheme is obviously more desirable for application development. Support for interrupt driven events would require explicit support in the reconfigurable hardware. The polled method places no onus on the designer to use special components, as the content of any arbitrary register is accessible by default.

The software API has been used in the development of several reconfigurable applications. Image processing and filtering applications have been produced, as well as examples of pattern matchers using data folding [10] principles. The experience of developing these applications prompted the creation of an interactive tool to accelerate the debugging and prototyping of applications.

5. Interactive Design Tool

An interactive design tool has been developed, building upon and extending the features of the software API. The tool's GUI provides the user with extensive control over all of the device functionality. The techniques used within this design tool are drawn from a combination of hardware emulation and software debugging methods. The user interface for the tool is shown in **Fig 3**.

To aid hardware debugging, the outputs of all cells in the array are observable. These values represent the combinatorial or sequential output, depending on the device configuration. Any arbitrary register can be set from software, with the proviso that the underlying logic has to be configured to accept data over the configuration bus. It is a feature of the XC6200 that any register can be configured to be accessible in this manner, without requiring routing to I/O pads.

Several flexible clocking methods are supported, allowing an arbitrary number of clock pulses to be sent to the array. Access to marked registers on the array can be configured to generate a notification pulse in the control logic, which can be routed into the application logic and used as a clock. This novel feature of XC6200 FPGAs is exploited to provide clock stepping during the transmission of data to the array. With slight modification, any synchronous design can be converted to allow single step control over the clock input.

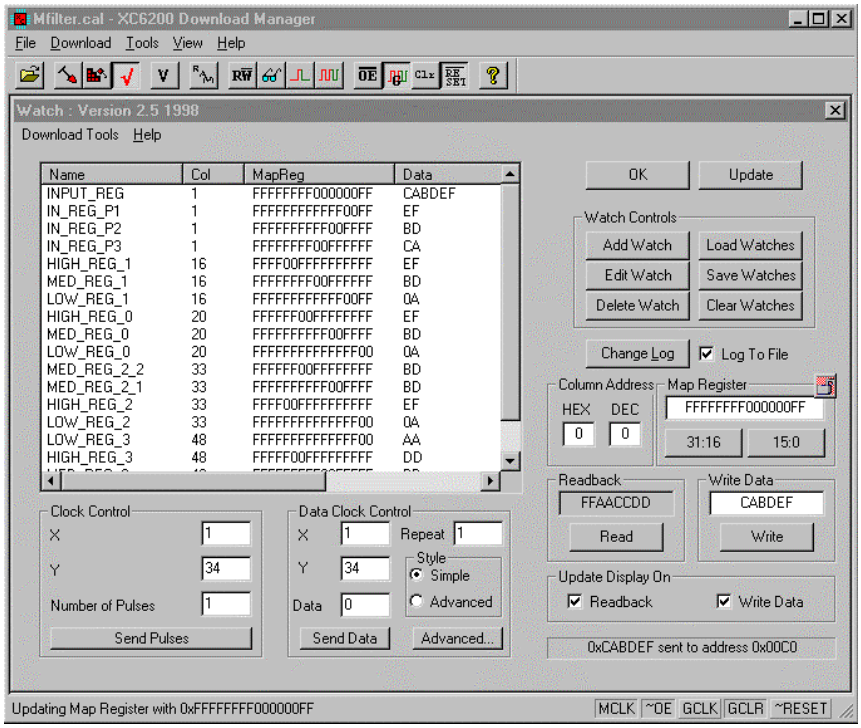

Fig 3. Debugging Interface

Inspection points or watches can be set on areas of the array. The values of these watches are updated on each clock transition. Breakpoint values can be associated with watches, which are analogous to the events described in the previous section. Using breakpoints, the system clock can be advanced until an event of interest occurs, then operation is suspended, allowing the designer to inspect the state of the target

sub-system. At this point, new configurations can be loaded into the array, or logical values changed.

To aid in the prototyping of configuration algorithms, watches also support reconfiguration vectoring, as discussed previously. The order in which watches are entered in the debug tool affects the order in which they are updated, allowing the events to be prioritised. The interactive nature of the tool allows the rapid investigation of a variety of what-if scenarios for the control algorithms. When the algorithm is finalised, it can be quickly converted to a programmatic form, as the interface and methods are consistent between the API and the debugging tool. Application algorithms can also be prototyped in much the same manner, either by single stepping through the operation, or by transferring blocks of data until breakpoints are encountered. Full logging is provided of the selected watches, both at clock step points and during reconfiguration intervals.

6. Conclusions

The development environment has provided a flexible platform for research into reconfigurable systems. Many of the more novel features in this work are currently only supported by the XC6200 family of devices but it is anticipated that similar features will appear in next-generation FPGAs and that the techniques discussed can be migrated onto these devices.

The environment has proven a success for two main reasons. The more advanced features have allowed for rapid prototyping of complex systems. In particular, the integration of the application software with the underlying hardware has proven a simple task when using the API. The second main success has been the use of the IDT to simplify the introduction of relatively inexperienced users into the field of reconfigurable logic.

Closer integration of processor and FPGA would alleviate many of the speed penalties currently incurred by communicating over the relatively slow PCI bus [8]. The co-design environment has the potential to be of particular relevance for prototyping designs for some of the recently announced systems-level FPGA / microprocessor hybrid devices [6], [11]. As a result, complete software control and management of reconfiguration may prove a feasible solution for an increasing number of reconfigurable designs.

References

[1] Hadley, J. D., & Hutchings, B. L.: Design Methodologies for Partially Reconfigured Systems, In Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines, pp. 99-107, Apr 95

[2] Eggers, H., Lysaght., P., Dick, H., McGregor, G.: Fast Reconfigurable Crossbar

Switching in FPGAs In R. W. Hartenstein, M. Glesner (Eds.) Field-Programmable Logic - Smart Applications, New Paradigms and Compilers, Springer-Verlag, Germany, 1996, pp. 297-306

[3] Lysaght, P., Stockwood, J.: A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays In IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 4, No. 3, pp. 381-390, Sept. 96

[4] MacKinlay, P.I., Cheung, P.Y.K., Luk, W., Sandiford, R.L.: Riley-2: A Flexible Platform For Co-design And Dynamic Reconfigurable Computing Research In Field Programmable Logic and Applications, Proceedings of FPL'97, pp. 91-100, W. Luk & P. Cheung Eds., Springer-Verlag, 1997

[5] Kwiat, K., Debany, W.: Reconfigurable Logic Modelling, In Integrated System Design, December 1996 (www.isdmag.com)

[6] Faura, J., Moreno, J. M., Madrenas, J., Insenser, J. M.: VHDL Modeling of Fast Dynamic Reconfiguration on Novel Multicontext RAM-based Field Programmable Devices In VHDL User's Forum in Europe 1997

[7] Robinson, D., Lysaght, P., McGregor, G.: New CAD Tools and Framework for Detailed Timing Simulation of Dynamically Reconfigurable Logic, ibid.

[8] Robinson, D., Lysaght, P., McGregor, G., Dick, H.: Performance Evaluation of a Full Speed PCI Initiator and Target Subsystem using FPGAs In Field Programmable Logic and Applications, Proceedings of FPL'97, pp. 41-50, W. Luk & P. Cheung Eds., Springer-Verlag, 1997

[9] Lysaght, P.: Toward an Expert System for Apriori Estimation of Reconfiguration Latency in Dynamically Reconfigurable Logic In Field Programmable Logic and Applications, Proceedings of FPL'97, pp. 183-192, W. Luk & P. Cheung Eds., Springer-Verlag, 1997

[10] Foulk, P. W.: Data Folding in SRAM Configurable FPGAs In IEEE Workshop on FPGAs for Custom Computing Machines, pp. 163-171, Napa, CA, Apr. 1993

[11] Motorola: MPACF250 - MPA's CORE+ Reconfigurable System : Product Brief, Ref. MPACF250PB/D, Motorola, Inc., 1998