Towards High-Level Specification, Synthesis, and Virtualization of Programmable Logic Designs

Oliver Diessel, Usama Malik and Keith So

School of Computer Science & Engineering
University of New South Wales
UNSW Sydney NSW 2052
Australia

Abstract. Current FPGA design flows do not readily support high-level, behavioural design or the use of run-time reconfiguration. Designers are thus discouraged from taking a high-level view of their systems and cannot fully exploit the benefits of programmable hardware. This paper reports on our advances towards the development of design technology that supports behavioural specification and compilation of FPGA designs and automatically manages FPGA chip virtualization.

1 Introduction

A significant barrier to the wider adoption of reconfigurable computing is the problem of mapping applications into circuit structures that can easily be implemented on a given FPGA device. Ideally we should be able to describe the desired functionality of hardware or its components, and have a compiler map these specifications into effective logic allocations and reconfiguration schedules. Being oriented towards static hardware configurations, current FPGA design flows do not support such dynamic circuit design and configuration.

Our research aims to identify the key techniques and principles that underlie the design of suitable languages and compilers for the high-level design and implementation of run-time reconfigurable applications. We are currently investigating how to model and express the parallelism inherent in FPGA circuits and how to implicitly control run-time reconfiguration in an abstract and machine independent manner.

For a number of reasons, we have so far focused on the use of a process algebra as the high-level specification language [7]. Process algebras (PAs) are simple yet powerful formalisms in which it is easier to explore fundamental language issues than with hardware description languages and programming languages. PAs are well-suited to the behavioural description of interacting finite state machines, and as such can be used to model control-parallel and systolic FPGA applications. Furthermore, there is the hope that a top-down, hierarchical, and modular focus, as emphasized by a PA such as Circal, will aid logic synthesis because ever more complex structures may be built through assembly, while the effort required to design each module remains relatively constant.

Our approach differs from other efforts to develop high-level programming languages for reconfigurable computing, which commonly augment sequential languages such as C, C++, and Java with support for data-parallel operations and hardware layouts [1, 6, 2, 9]. While such languages have been sucessfully used to design efficient applications, they emphasize a signal-oriented view of computation, do not allow concurrency to be expressed naturally, and they do not attempt to target the run-time reconfigurable capabilities of the hardware. Our goal is to model the capabilities of the hardware in order to have a well-understood target for compilers that can exploit those capabilities and languages that allow them to be expressed.

2 An FPGA interpreter for Circal

In [3] we described a compiler that derives and implements a digital logic representation of high-level behavioural descriptions of systems specified using Circal. At the topmost design level, the circuit is clustered into blocks of logic that correspond to the processes (individual finite state machines) of a system. The process logic blocks implement circuits with behaviours corresponding to the component processes of the specification. Below the process level in the hierarchy, the circuits are partitioned into component circuit modules that implement logic functions of minor complexity. The circuit modules are rectangular in shape and are laid out onto abutting regions of the array surface, allowing signals to flow from one module to another via aligned ports.

The current implementation of the compiler targets the Virtex chipset [10], which is substantially more coarse-grained, operates at considerably higher frequencies, and is availble in much greater logic densities than the original Xilinx XC6200 target. Module placement follows a decomposition approach similar to that of the XC6200-based compiler, albeit with an arrangement that utilizes the fast logic cascade chains and the enriched routing fabric available on Virtex, while taking into account that partial reconfiguration is column-oriented.

2.1 Developing support for run-time deployment of Circal models

The compiler may produce circuits that cannot be implemented because they are too large for the available FPGA resource. We have therefore developed support for automatically partitioning the circuit and swapping the resulting partitions with those on chip so as to give the effect of having enough FPGA area to implement circuits of any size.

The approach we are exploring is to insert a virtual hardware manager (VHM) between the front-end of the compiler, which derives a hardware independent representation of the circuit in a modularized form, and the back-end, which maps and places these modules onto a particular FPGA. In doing so, we preserve some off-line features of a compiler, and incorporate some of the on-line functions of an interpreter. The VHM stores the circuits off-chip in a state transition graph form. When a new partition is needed, the module parameters for the corresponding sub-graph are passed to the back-end for bitstream generation and loading. The back-end makes use of the JBits API for configuring the device [4].

The FPGA area is partitioned into regions that are reserved for holding the logic for a single porcess. The area provided is large enough to implement the logic for a single state in the worst case. We attempt to provide sufficient space to map a sizable portion of the state transition graph by expanding the area allocated to each process. At initialization, the VHM selects a sub-graph rooted at the initial state for each process. The graph is traversed in a breadth-first manner in order to identify the amount of logic that can be accommodated on chip. To avoid backtracking, the search is guided by circuit size estimates based on the number of transitions each included state adds to the sub-graph. The identified sub-graph is then given to the back-end which maps it to the FPGA. When a state transition leads to a state that lies on the boundary of the implemented sub-graph, an exception is generated and the VHM selects a new sub-graph rooted at the boundary state.

2.2 Example

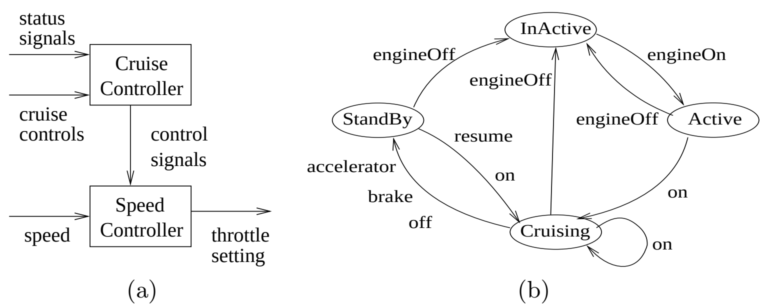

Fig. 1. (a) A cruise control system; (b) Cruise Controller graph [5]

Figure 1(a) depicts a cruise control system for an automobile. The system consists of a Cruise Controller and a Speed Controller. Inputs are provided by sensors not shown in the diagram. In Figure 1(b) we model the Cruise Controller process. Suppose the area avilable to this process suffices to implement the complete transition logic for 2 states together with any outedges from these leading to a boundary state. In that case, the initial partition includes the *Inactive* and *Active* states, as well as the state flip-flop for the *Cruising* state. When the *Cruising* state is entered, an exception is triggered, and a new sub-graph consisting of the *Cruising* and *Standby* states is configured, together with a state flip-flop for the *Inactive* state.

The Virtex compiler needs 7 columns and 5 rows of configurable logic blocks (CLBs) to implement the complete state graph. The interpreter allows the first partition to be implemented in 4 columns and 4 rows and the second partition to be implemented in 5 columns and 4 rows. Reconfiguring the 5 columns takes about 1.25ms on an XCV1000 chip, which contains 64 rows and 96 columns of CLBs altogether. The time to generate the partition is likely to be larger.

Reconfiguration delays are expected to be amortized in practice by aligning processes on top of each other. The stack of processes could thus be reconfigured with a single frame update. In order to reduce the cost of generating the partitions, they might be cached once generated in case they are to be used once more.

3 Future Work

In its current form, Circal is suited to the specification of control-flow applications like logic controllers, protocols checkers, and cellular automata. In order to fully exploit the capabilities of modern high density FPGAs, support for datapath designs must be provided by Circal. One approach we intend to pursue is to consider building a domain specific language into which the control features developed with Circal are embedded.

References

1. M. Aubury, I. Page, G. Randall, J. Saul, and R. Watts. Handel-C *Language Reference Guide*. Oxford Univerity Computing Laboratory, Oxford, UK, Aug. 1996.
2. P. Bellows and B. Hutchings. JHDL — An HDL for reconfigurable systems. In Pocek and Arnold [8], pages 175 – 184.
3. O. Diessel and G. Milne. A hardware compiler realizing concurrent processes in reconfigurable logic. *IEE Proceedings — Computers and Digital Techniques*, 148(4):152 – 162, Sept. 2001.
4. S. A. Guccione and D. Levi. XBI: A java-based interface to FPGA hardware. In J. Schewel, editor, *Configurable Computing: Technology and Applications, Proc. SPIE 3526*, pages 97–102. SPIE – The International Society for Optical Engineering, Nov. 1998.
5. J. Magee and J. Kramer. *Concurrency: State Models & Java Programs*. Worldwide Series in Computer Science. John Wiley & Sons, New York, NY, 1999.
6. O. Mencer, M. Morf, and M. J. Flynn. PAM–Blox: High performance FPGA design for adaptive computing. In Pocek and Arnold [8], pages 167 – 174.
7. G. J. Milne. CIRCAL and the representation of communication, concurrency, and time. *ACM Transactions on Programming Languages and Systems*, 7(2):270 – 298, Apr. 1995.
8. K. L. Pocek and J. M. Arnold, editors. *The 6th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, Los Alamitos, CA, Apr. 1998. IEEE Computer Society Press.
9. G. Snider, B. Shackleford, and R. J. Carter. Attacking the semantic gap between application programming languages and configurable hardware. In *FPGA'01 Ninth International Symposium on Field Programmable Gate Arrays*, pages 115 – 124, New York, NY, Feb. 2001. ACM Press.
10. Xilinx. *Virtex 2.5V Field Programmable Gate Arrays*. Xilinx, Inc., Oct. 2000. Version 1.3.