Available online at www.sciencedirect.com
SCIENCE DIRECT
Microprocessors and Microsystems 29 (2005) 435–450
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Programming paradigms for reconfigurable computing

Gareth Lee, George Milne*[1]

School of Computer Science and Software Engineering, The University of Western Australia, Perth, WA 6009, Australia

Accepted 21 January 2005
Available online 22 February 2005

Abstract

High-level programming paradigms are examined to determine their appropriateness for describing systems, which are amenable to automated compilation onto a reconfigurable computing platform. We aim to find a set of language features to act as a basis for future language development which: provide a concise description of the system to be realised; provide clear and intuitive semantics; abstract the underlying technology and are appropriate to the underlying technology.

Clearly language design is a subjective process, but we have adopted a systematic approach by assessing the efficacy of existing programming and hardware description languages. We examine the languages Java, VHDL, Standard ML and Circal. Our method also bases the comparison on a set of properties drawn from an archetypal example which we know maps well onto reconfigurable computing platforms. We conclude that none of these established languages has all the properties required for describing reconfigurable computation.

This approach leads to insight into the capabilities of existing languages and allows us to determine the essential characteristics of future languages oriented to reconfigurable computing.

Keywords: Circal; Field-programmable gate array; Hardware description language; Java; Reconfigurable computing; Standard ML; VHDL

1. Introduction

Continued increases in circuit density have elevated the field-programmable gate array (FPGA) from a utility device with which electronic engineers implement glue logic, into a fully-fledged computer architecture. Applications for FPGAs are still rather specialised and they are mostly used, much as embedded DSP devices, in high performance signal processing systems. However, given their ever decreasing cost, high performance and low power requirements, FPGAs are becoming more attractive in a wide range of mobile computing and embedded applications, either as stand alone devices or in concert with general purpose processors.

Reconfigurable computers are general-purpose computer devices built from FPGAs and composed of a large matrix of configurable logic blocks (CLBs). A simplified FPGA schematic is shown in Fig. 1: section (a) shows a matrix of CLBs with each connected to its immediate neighbours and (b) the three distinct elements which make up each CLB:

- One or more functional units, programmable to implement Boolean functions;
- One or more latches which allow state to persist over time;
- A programmable crossbar router which allows each CLB to connect its inputs and outputs to its neighbours (or via its neighbours to CLBs further afield).

For example, grouping together a number of functional units and configuring them to behave as full-adders allows an ALU to be constructed; similarly grouping together latches allows arbitrary registers to be allocated. Each of the CLBs operate concurrently as do all the communication signals between them.

The behaviour of the functional units and the connections between CLBs can be configured by setting bits in a global random access memory, hence the configuration of any

* Corresponding author. Tel.: +61 8 6488 2717; fax: +61 8 6488 1089.
E-mail address: george@csse.uwa.edu.au (G. Milne).
[1] This work was funded by a grant from the Australian Research Council.

0141-9331/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2005.01.002

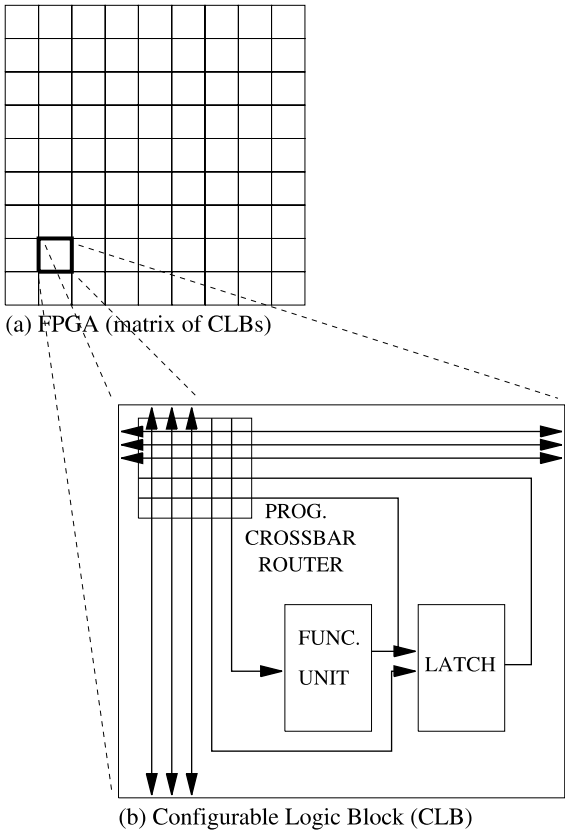

Fig. 1. The key elements of a field-programmable gate array.

CLB can be changed at any time by an external controller.[2] Thus, the reconfigurable computer can be programmed to form arbitrary circuits, rather than following a sequence of instructions, as does a general purpose computer.[3] Reconfigurable computers can also be programmed to form a circuit specialised to solving a particular problem in a highly concurrent manner. Typically pipelined circuits which implement systolic arrays or cellular automata are created; pipelines which are wide (acting on each of the elements of a vector) and deep (concurrently performing a long sequence of steps) provide the greatest concurrency.

With the widespread use of reconfigurable computers based on FPGAs, will come additional demands on the tools used to program them and validate the systems that result. This paper focuses on the use of high-level languages (HLLs) in the design of systems for reconfigurable computers. It has been widely demonstrated in the software development community that the use of HLLs, and the design abstraction which results, offer substantial productivity gains. This is also true of reconfigurable computing which delivers computational throughput by orchestrating massive concurrency, rather than relying on high clock rates.

Most current languages for reconfigurable computing have one of two origins:

- They are hardware description languages (HDLs) designed for describing static architectures in ASIC devices which have been adapted for use with FPGAs: for example, VHDL [1] and Verilog [2]; or
- They are adapted from systems programming languages, such as Handel-C [3] (based on C), SystemC [4] (based on C++) or Java [5] (by using Xilinx's Forge software).

In both cases, the choice of language is seen as being of secondary importance and most of the effort goes into developing sophisticated compilers. It is assumed that designers familiarity with the base language will be sufficient to overcome any mismatch between the language paradigm and characteristics of the underlying hardware.

Our approach is to consider the language issue afresh; to avoid subjectivity we focus on language paradigms and capabilities rather than syntactic issues. We consider *object orientation*, *functional programming*, *direct hardware description* and *process algebras*. We have deliberately selected a wide range, including approaches not traditionally associated with reconfigurable computing.

In this paper, we compare and contrast these distinct language paradigms to assess their efficacy for programming reconfigurable computers. In each case, a representative language has been chosen to allow us to make concrete comparisons. We take an experimental approach by encoding a test example using each of the chosen languages to allow a comparison of the paradigms using a number of essential attributes and capabilities.

Our approach also differs from many previous authors who have used languages, such as those under consideration here, as 'circuit generators' [6,7]. In this case, the languages are used to generate low-level structure; it is assumed the designer has already decided on the best circuit to solve a problem and needs to decide how to physically lay out the circuit using available FPGA resources. In contrast, we have adopted a high-level approach, using the languages to describe the system in the most literal and transparent fashion possible. We assume compiler and synthesis tools are available to map the design onto FPGA resources.

The paper is structured as follows: In Section 2, we introduce the test example, a regular expression matcher, which we previously demonstrated is suited to implementation using reconfigurable computers [8,9]. In Section 2.2, we discuss the properties of the example and how these are representative of a much larger category of systems, which are amenable for implementation on reconfigurable computing platforms. In Section 3, we introduce the target languages, which act as representatives of the paradigms

[2] Typically this configuration controller will be external to the FPGA logic but some manufacturers are now integrating a general purpose processor within the die and even allowing self reconfiguration from within the FPGAs logic.

[3] Such stored-programs computers are normally based on the von Neumann architecture where data and instructions are transferred in and out of a processor core across a single bus from external memory (or the related Harvard architecture, which uses separate busses for instructions and data).