An Approach to the Specification and Verification of a Hardware Compilation Scheme

JONATHAN P. BOWEN jonathan.bowen@sbu.ac.uk
South Bank University, Centre for Applied Formal Methods,
School of Computing, Information Systems and Mathematics, Borough Road, London SE1 0AA, UK

HE JIFENG jifeng@iist.unu.edu
The United Nations University, International Institute for Software Technology, PO Box 3058, Macau

Abstract. The use of Field Programmable Gate Arrays (FPGA) to produce custom hardware circuits rapidly using a completely software-based process is becoming increasingly widespread. Specialized Hardware Description Languages (HDL) are used to describe and develop the required circuits. In this paper, we advocate using an even more general purpose programming language, based on Occam, for the automatic compilation of high-level programs to low-level circuits. The parallel constructs of Occam can map directly to hardware as conveniently as to software, with potentially dramatic speed-up of highly parallel algorithms. We demonstrate that the compilation process can be verified using algebraic refinement laws, increasing the confidence in its correctness. Verification is particularly important in high-integrity systems where safety or security is paramount. A prototype compiler has also been produced very directly from the theorems using the logic programming language Prolog.

Keywords: digital systems, formal specification, hardware compilation, parallel programming, programmable hardware, refinement, verification

1. Background

Developments in very large-scale integration (VLSI) technology have made it possible to speed up task-specific software systems by implementing at least some of its modest-size core procedures in hardware circuits. While such practices are increasingly enticing and promising, it remains to be a daunting task to show that a target VLSI device, with size reaching one million transistors, does comply with a given source algorithm. This paper aims to bridge this gap by presenting a selection of transformation rules that converts a program which fulfills the logical specification of a circuit into a digital VLSI device [19]. Field programmable gate arrays (FPGA) which can be dynamically reconfigured by software may be used for implementation. This enables the building of hardware implementations for moderate-sized programs entirely by a software process. A significant feature of such hardware implementation is that a global clock is present to synchronize the activity of sub-components, i.e., update on latches can only take place at the end of each clock cycle.
A high-level parallel programming language (such as Occam [26], for example) can be regarded in this context as a behavioural specification language for hardware devices, capable of being compiled directly into circuits [27]. An Occam-like program, together with its observation semantics, provides the abstract definition of what a hardware circuit should achieve, by means of the proper connection of its components such as latches and gates. Such an approach when automated has been dubbed ‘hardware compilation’ [6, 30]. A major advantage of Occam in this context is the inclusion of high-level parallel constructs that can map naturally to parallel hardware. Occam's well-studied algebraic laws [26] also aid considerably in the verification of such a mapping.

A hardware description language (HDL, e.g., Verilog and VHDL [7, 9]), provides a way to express formally and symbolically the constituent components of a hardware circuit and their interconnections. A hardware description of a VLSI device can be checked against the silicon layout supplied by the designer, and it can also be used as an input to simulators. Hardware description languages are widely used in many computer-aided systems, allowing libraries of standard checked hardware modules to be assembled. The combination of all these techniques removes many errors from a silicon product, once the hardware description of a device has been constructed. In this paper a simple description language for globally clocked circuits will be given an observation-oriented semantics based on the states of the wires of a device.

Algebraic laws [26] based on this semantics permit every circuit description to be expressed in a hardware normal form. This form is designed to guarantee absence of such errors as combinational cycles and conflicts. The necessary link with the higher abstraction level of the programming language is provided by an interpreter written in the programming language itself. A hardware normal form is a correct representation of a source program if its interpretation yields as good a result or a ‘better’ result than that described by the source program itself with respect to some refinement ordering. In this context, ‘better’ means more deterministic or applicable in more situations, for example. This is proved directly for each of the primitive components of the source language; and then a series of theorems show how the hardware form of a composite program can be constructed from the hardware which implements its components. Each theorem has the form of a transformation rule, which can be used directly or indirectly in the design of an automatic compiler. In this way we hope to demonstrate that by its very method of construction the compiler is correct. The hardware normal form is fairly close to the typical notation of a hardware ‘netlist’ language. This second level of translation is not the subject of this paper.

This work builds upon results published in [6, 17, 25]. In particular, there is a strong relationship between our method and that used in software compilation [13]. However, our method handles parallel composition and preserves true concurrency in the implementations. Additionally, a simple FPGA description language is introduced to mimic the behaviour of a synchronous circuit, which can also be defined in the same semantical model used for the source language.

Page has developed a compiler in the functional language SML which converts an Occam-like language, somewhat more expressive than the one presented here, to a