Preface

In the mid 1960s, when a single chip contained an average of 50 transistors, Gordon Moore observed that integrated circuits were doubling in complexity every year. In an influential article published by Electronics Magazine in 1965, Moore predicted that this trend would continue for the next 10 years. Despite being criticized for its "unrealistic optimism," Moore's prediction has remained valid for far longer than even he imagined: today, chips built using state-of-the-art techniques typically contain several million transistors. The advances in fabrication technology that have supported Moore's law for four decades have fuelled the computer revolution. However, this exponential increase in transistor density poses new design challenges to engineers and computer scientists alike. New techniques for managing complexity must be developed if circuits are to take full advantage of the vast numbers of transistors available.

In this monograph we investigate both (i) the design of high-level languages for hardware description, and (ii) techniques involved in translating these high-level languages to silicon. We propose SAFL, a first-order functional language designed specifically for behavioral hardware description, and describe the implementation of its associated silicon compiler. We show that the high-level properties of SAFL allow one to exploit program analyses and optimizations that are not employed in existing synthesis systems. Furthermore, since SAFL fully abstracts the low-level details of the implementation technology, we show how it can be compiled to a range of different design styles including fully synchronous design and globally asynchronous locally synchronous (GALS) circuits.

We argue that one of the problems with existing high-level hardware synthesis systems is their "black-box approach": high-level specifications are translated into circuits without any human guidance. As a result, if a synthesis tool generates unsuitable designs there is very little a designer can do to improve the situation. To address this problem we show how source-to-source transformation of SAFL programs "opens the black-box," providing a common language in which users can interact with synthesis tools whilst exploring the different architectural tradeoffs arising from a single SAFL specification. We demonstrate this design methodology by presenting a number of transformations that facili-

tate resource-duplication/sharing and hardware/software co-design as well as a number of scheduling and pipelining tradeoffs.

Finally, we extend the SAFL language with (i) π-calculus style channels and channel-passing, and (ii) primitives for *structural*-level circuit description. We formalize the semantics of these languages and present results arising from the generation of real hardware using these techniques.

This monograph is a revised version of my Ph.D. thesis which was submitted to the University of Cambridge Computer Laboratory and accepted in 2003. I would like to thank my supervisor, Alan Mycroft, who provided insight and direction throughout, making many valuable contributions to the research described here. I am also grateful to the referees of my thesis, Tom Melham and David Greaves, for their useful comments and suggestions. The work presented in this monograph was supported by (UK) EPSRC grant GR/N64256 "A Resource-Aware Functional Language for Hardware Synthesis" and AT&T Research Laboratories Cambridge.

December 2003 Richard Sharp