5 AUTOMATIC FORMAL DERIVATION APPLIED TO HIGH-LEVEL SYNTHESIS

José Manuel Mendías
Román Hermida

5.1 Introduction

Since the early 80's, when first high-level synthesis (HLS) algorithms appeared, behavioral synthesis tools have had to evolve quickly. From day to day new fields of application are being discovered that require to review the old techniques. Continuously, the market is requiring new algorithms able to perform more efficient search of solutions. Everyday more abstract specification methods are required, which enlarge the semantic gap between specification and circuit. And all this development has got a price to be paid: the complexity of algorithms grows and the supporting data structures become more sophisticated. As a consequence, the bugs in the tools (or even in the algorithms) proliferate. The effect is that reliance in synthesis tools decreases and nowadays no sensible designer takes the risk to accept a circuit automatically generated without a later validation step; either using simulation or verification techniques. This means, in practice, that the well-known paradigm of correctness by construction, widely publicized formerly, has to be handled very carefully.

To address this problem the so-called formal or transformational synthesis systems appeared recently. Their aim is to perform all the design steps within a purely mathematical framework, where the synthesis process itself becomes the proof of soundness of the implementation. There are three common characteristics of these systems: i) they use a single mathematical formalism that is versatile enough as to represent the specification, the implementation, and the set of possible intermediate states of a design process; ii) they synthesize by sequentially applying a set of behavior preserving transformations, which have been proved to be correct;
iii) they are not automatic: although any transformation can be done automatically, the sequence of transformations is decided by the designer. From a theoretical point of view, the designs generated by these systems are necessarily correct by construction, given that the transformations are sound. However, from a practical point of view, the reliability of the tool depends both on the complexity of the mathematical support and the reliability of the implementation of the kernel of transformations.

Although a good survey on formal systems can be found in [Kuma96], some systems can be highlighted. The DDD system [JoBo91], which being based on a simple functional language, is focused to manually transform datapaths under sequential control. It uses the so-called factorizations [John89] as basic transformations to modify the design structure by applying distributivity among functions. Another interesting system is T-Ruby [ShRa95] which works with very abstract specifications described in terms of input/output relations at both logic and RT-level [JoSh90]. It allows to obtain a great deal of implementations with systolic or iterative style. In the commercial side, the LAMBDA/DIALOG [FiFM91] tool should be mentioned as an example of the application of higher order theorem provers to design problems. And more recently, we could cite HASH [BIEi97] that is specialized in the verification of schedules of linear graphs, by using as a kernel a modified version of the higher order theorem prover HOL. There are two important aspects to be considered in HASH. Firstly, it does not use a standard implementation of HOL, because in standard implementations the complexity of the proof grows exponentially with the length of the critical path. Secondly, the proposed representation, unfortunately, does not admit feedback loops, which means a severe limitation on input graphs and, even worse, the inability to represent real circuits, that must be fed back to allow component reuse.

In this chapter we will present the main features of a research project that led to the design of the formal tool FRESH (FRom Equations to Hardware) that covers the whole HLS process. Like previous systems, the tool is not self-contained (it does not make any design decisions), but besides the possibility of being operated by a designer, it can be driven by a conventional HSL tool, thus creating a framework where the correctness of the design can be ensured automatically. The main advantages of this system are easiness, reliability, applicability and efficiency. Easiness means not having to modify conventional tools to adapt them to the mathematical formalism; they simply must deliver the results of their search: what cycle each operation has been scheduled in, what operations share the same module, etc. Reliability means selecting a kernel of few simple transformations (which minimizes the number of error sources) and adopting a declarative representation with first-order formal semantics (which simplifies both designer interpretation and tool processing). Applicability is obtained, differing from other systems, neither by constraining the kind of accepted behaviors nor the kind of reachable designs. Efficiency is obtained by specializing the system in order to perform complete formal synthesis processes (from specs to datapath+controller) with quadratic complexity.