Computer Architecture Modeling, Synthesis, and Verification

Arvind, Daniel L. Rosenband, Jacob B. Schwartz

Introduction: Past work in our group has shown that atomic actions formulated as a Term Rewriting System (TRS) are a good way to describe hardware. We have shown that advanced architectures can be described precisely using sets of atomic actions[1, 2] and formal methods can be applied to these descriptions to reason about their properties[3]. Furthermore, we have shown that such atomic descriptions are also amenable to efficient hardware synthesis [4, 5]. The goal of our current efforts are to create a unified framework in which three aspects of design – modeling, verification, and hardware synthesis – can be explored synergistically. This work combines expertise in architectures, compilers, formal methods, and hardware descriptions. Our expectation is that it will lead to an advanced chip design flow that provides the designer with a much higher level of abstraction than what is currently used and will lead to far quicker design cycles from early design concept to verified layout.

Approach: We use Bluespec[6] as the base language for hardware modeling, synthesis and verification. Bluespec is a high-level object-oriented language which has been designed to embody (hardware-oriented) atomic actions as its execution model, and whose syntax and type system is based on the functional language Haskell. In Bluespec an object represents a hardware module with internal state, rules to manipulate the state, and an interface (a set of methods) through which other modules can observe and manipulate this state. The Bluespec compiler flattens each module and performs extensive partial evaluation in order to arrive at an intermediate form (ATS) which essentially represents a set of atomic actions. Synthesis and scheduling algorithms are applied to the ATS to derive either an RTL Verilog description for synthesis or a C executable for simulation.

The goals of our modeling efforts are to create quickly very precise models of processors, systems, and application specific hardware (for example, a router or a cache-protocol engine). These models need to be executable to discover inconsistencies and incompleteness of specifications, and to allow experimentation for performance evaluation. The high-level features of Bluespec, especially its module system which permits putting Bluespec wrappers around modules written in Verilog and C, make it particularly suitable for modeling using a step-wise refinement design methodology. Bluespec's type system also makes it easy to create highly parameterized design libraries. For example, we have already created highly parameterized versions of FIFO's, register files, reorder and completion buffers. These modules can be used across different projects. Bluespec's execution model, which is based on serializability of atomic actions, makes the concurrency in a system easy to express. This combination of high-level language features and concurrency makes Bluespec a powerful modeling language.

However, what makes Bluespec unique is that we can also generate hardware circuits from these descriptions. Even if not refined to the level one would expect of an actual hardware implementation, we expect to generate hardware circuits from these behavioral descriptions to map them on FPGA's. Using the FPGA's we can then execute the models far more quickly than can be done in software simulation. This should allow for rapid experimentation with a variety of general purpose and special purpose architectures.

Our approach to improving hardware synthesis results is taking several directions. One effort involves improving the algorithms that generate hardware from sets of atomic actions. Another effort involves creating an even higher level of abstraction for the designer. We are working on automatic synthesis of pipelined circuits, examining real-time constraints, etc. The third effort in the area of synthesis is examining how user annotations can help guide the synthesis process. The designer is often aware of design properties that the compiler cannot derive, and we would like to introduce these properties as annotations that carry proof obligations.

One of the driving forces behind using Term Rewriting Systems (TRS) to describe hardware was that it provided a precise definition of hardware behavior. At the same time, it is a formalism that is suitable for formal verification methods. Since Bluespec was inspired by these TRS descriptions, we would like to combine the modeling and synthesis capabilities with formal verification methods. We would like to apply model checking and automatic theorem provers to Bluespec descriptions (or to its intermediate form) so that we can reason about their properties. For example, we would like to be able to prove that a FIFO description behaves as we would expect a FIFO to behave. Or, we would like to show that a pipelined processor description has the same behavior as a non-pipelined version.

Progress: Several two-stage and five-stage processors with a very simple ISA have been created. They allow us to experiment with basic building blocks such as bypass circuits, reorder buffers, and register files. Using these building

blocks we have gone on to create a fully bypassed five-stage MIPS processor. In simulation we have executed MIPS programs on this processor. Synthesis results of the MIPS processor are also encouraging. The clock cycle time is comparable to that of a hand coded verilog implementation and the gate counts are slightly higher. However, so far no serious effort has been made to reduce the amount of logic generated.

Our main effort in the area of synthesis has been to improve compile times. In [7] we developed an algorithm for modular scheduling of atomic actions. Using this algorithm we have been able to dramatically reduce compile times of large models. For example, we were able to reduce the compile time of the MIPS processor from over 15 minutes with optimizations turned off, to a little over 30 seconds with optimizations turned on. This algorithm also presents a method to introduce scheduling annotations (at module boundaries), that result in more realistic and efficient circuits.

Future: We will continue research in all three areas: modeling, synthesis, and verification using Bluespec as the base description language. For modeling purposes we plan to create a robust infrastructure to take behavioral hardware descriptions and map them to FPGAs. This will allow for rapid prototyping and exploration of processors and other hardware architectures. We also plan to create more complex processor models which incorporate caches, branch prediction, superscalar functional units, and other features found in modern processors. The possibility of collaborating with several companies on system level modeling of their next generation high-performance computers also exists.

Synthesis efforts will continue to improve the hardware and atomic action schedules. We will extend our modular scheduling algorithms to support more complicated module topologies. We also want to explore what user annotations can be used to assist the compiler in meeting performance / real-time requirements. Another area that we are investigating is extensions to the type of atomic actions that we allow. For example, atomic actions that take multiple cycles can be used to more easily create pipelined circuits.

Our automated formal verification effort is just beginning and we are exploring the possibility of combining Bluespec with some assertional language. We plan to explore the invocation of model checkers and theorem provers automatically from a Bluespec program. Our goal is to create a unique hardware design environment which will facilitate the creation of robust, verifiable and efficient chips as quickly as possible.

Research Support: This work has been supported by Intel, IBM, and NSF.

References:

[1] Arvind and X. Shen, "Using term rewriting systems to design and verify processors," *IEEE Micro Special Issue on Modelling and Validation of Microprocessors*, vol. 19, no. 3, pp. 36–46, May/June 1999.

[2] X. Shen, *Design and Verification of Adaptive Cache Coherence Protocols*, PhD Thesis, Massachusetts Institute of Technology, February 2000.

[3] J. Stoy, X. Shen, and Arvind, "Proofs of correctness of cache-coherence protocols," in *Formal Methods Europe*, J. N. Oliveira and P. Zave, Eds. 2001, vol. 2021 of *Lecture Notes in Computer Science*, pp. 43–71, Springer-Verlag.

[4] J. C. Hoe, *Operation-Centric Hardware Description and Synthesis*, PhD Thesis, Massachusetts Institute of Technology, June 2000.

[5] J. C. Hoe and Arvind, "Synthesis of Operation-Centric hardware descriptions," in *International Conference on Computer Aided Design*. IEEE/ACM, 2000, pp. 511–518.

[6] Lennart Augustsson et al, "Bluespec: Language definition," 2001, `http://bluespec.org`.

[7] D. L. Rosenband, J. B. Schwartz, and Arvind, "Modular scheduling of atomic actions," in *CSG MEMO-463*, 2003.