Scalable Formal Design Methods for Asynchronous VLSI

Rajit Manohar
Computer Systems Laboratory
Cornell University
http://vlsi.cornell.edu/~rajit/

ABSTRACT

This lecture will provide an overview of the field of asynchronous VLSI, and show how formal methods have played a critical role in the design of complex asynchronous systems. In particular, I will talk about program transformations and their application to asynchronous VLSI, as well as describe a simple language that I developed to describe these circuits and aid in their validation.

Introduction

"It was intended that when Newspeak had been adopted once and for all and Oldspeak forgotten, a heretical thought ... should be literally unthinkable, at least so far as thought is dependent on words. ... Newspeak was designed not to extend but to diminish the range of thought ..."

– George Orwell, "1984"

The design of any modern VLSI system is a daunting task. The chip complexity in terms of the number of devices has been quadrupling every three years for the past three decades. All the devices on a chip operate concurrently; therefore, the design of VLSI systems can be thought of as the design of highly concurrent computations.

Clocked systems use a periodic global clock signal to implement barrier synchronization. These barriers are used to organize the concurrency on a chip. In traditional clocked design, actions that modify variables must complete before a predetermined barrier after which the variable can be read. There is no advantage obtained if some actions complete early unless we can reduce the gap between successive barriers—which requires that *all* actions must complete early, since the barriers are global and periodic.

In asynchronous (clockless) systems, synchronization is implemented by using message-passing primitives or local barriers between concurrent components that communicate with each other. Therefore, if some action finishes early, it is possible for the next action to begin without waiting for a global barrier synchronization.

The number of possible executions of an asynchronous computation can be much larger than the corresponding clocked system with the same functionality, since the absence of global barriers increases the number of possible interleavings of actions. This makes the problem of design verification more challenging. However, the absence of a global synchronization constraint makes asynchronous circuits more amenable to a formal top-down design approach, thereby eliminating the need for logical verification because the circuits are guaranteed to be correct by construction.

While formal top-down design can be used for clocked systems, the resulting circuits are less efficient than those produced by experienced designers. On the other hand, asynchronous circuits produced with this formal approach are as efficient (and sometimes even more efficient) than those produced by experienced designers. This is partly because the absence of the clock makes hand design of such circuits challenging. Thus, formal synthesis becomes the best way to explore the design space and manage design complexity.

To date, formal methods have not proven realistic for building large-scale devices and so practicing engineers shun them. Thus, our focus has been on a practical, *scalable* method that allows us to design extremely complex asynchronous computations using transformations whose correctness can be guaranteed by using local information. To achieve this, we are forced to restrict the class of circuits that we use. I will discuss some of the constraints we have introduced that we believe give us a scalable methodology without sacrificing the efficiency of the final implementation.

Program Transformations

We describe asynchronous computations using CHP (Communicating Hardware Processes), a language derived from Hoare's CSP written using Dijkstra's guarded command notation. At the this level of abstraction, we treat circuits as programs [6].

The introduction of hardware features that improve the final design (either in terms of energy requirements, performance, or other metrics) is treated as an exercise in program transformation. The original description is transformed into a different description in a way that guarantees that the observable features of the two computations are identical [7].

POPL '02 Jan. 16–18, 2002 Portland, OR USA

Once we are satisfied with the concurrent description, it can be transformed into an asynchronous circuit implementation by a mechanical procedure that only uses local information for each concurrent process.

A convincing demonstration of this design methodology was provided by the first asynchronous microprocessor ever designed [8]. This project used program transformations to synthesize asynchronous circuits from a high-level specification. While the methodology used for this project was sound, several parts of the design required an analysis of the complete processor to guarantee that the transformations applied were indeed correct. An important realization after this project was that avoiding shared variables across concurrent components (except those introduced in an extremely disciplined way) was important to avoid synchronization issues that would dominate the design.

If a transformation depends on the syntax of the program, then it is very easy to check whether a particular application of the transformation is valid, or to automate the application of the transformation [1]. Ideally, one should be able to apply transformations to a concurrent component by examining the component itself, not the entire program. Therefore, we preclude the use of shared variables across concurrent components except in the rarest of circumstances [5] to keep the design modular.

The goal of the MiniMIPS project was the design of a high-performance asynchronous processor with an industry-standard instruction set [9]. During this project, we realized that pipelining issues would dominate. In particular, performance optimization of asynchronous circuits requires that the number of buffer stages along various paths is carefully chosen to optimize throughput. Since the actual throughput of various components was not known until very late in the design, we decided to adopt a design style that allowed us to introduce buffer stages at arbitrary points in the communication graph to optimize throughput. This required a global design constraint for every process to guarantee that introducing these buffer stages would not affect the correctness of the design. We call the resulting design slack elastic, because we can adjust the synchronization slack on communication links without affecting correctness [4].

Slack elasticity allows us to introduce pipelining in an asynchronous computation without affecting correctness. In fact, we can obtain even more sophisticated program transformations where we examine the dataflow graph of a program and use projection to break up a sequential computation into concurrent parts [3].

Language Design and Tools

We designed a very simple language (named CAST) to describe asynchronous circuits. While the language itself is simple, it was sufficiently expressive for us to be able to describe a two million transistor asynchronous processor that was correct on first silicon.

The language has two primitive constructs: creation of an instance of a type, and connection between two types. In addition, it has meta language constructs that support component definitions, conditional instances, and loop constructs to permit the creation of regular structures. A connection between two circuit nodes corresponds to the two nodes being connected by a wire. Because connections in the language correspond to two components being permanently aliased, CAST is essentially a linking language.

In addition to supporting the description of circuit components, the language also contains primitive support for specifying simple invariants. The presence of these invariants permits other tools to perform their checks more efficiently. In some cases the invariants are necessary for a particular circuit implementation to be correct.

Exploiting the design hierarchy is critical in the implementation of any tool. One of the recurring themes we encountered while we were developing design tools is that the designer has a tremendous amount of information that is not expressed in the circuit description. Expressing some of this information in the circuit description not only allows the tools to check more properties of the system, but allows them to do so more efficiently [2].

I will conclude by discussing some open problems in the design of tools and languages for asynchronous systems.

References

[1] S. M. Burns and A. J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In *Proceedings of the Fifth MIT Conference on Advanced Research in VLSI*, pages 35–50, 1988.

[2] R. Manohar. A case for asynchronous computer architecture. In *ISCA Workshop on Complexity-Effective Design*, June 2001.

[3] R. Manohar, T.-K. Lee, and A. J. Martin. Projection: A synthesis technique for concurrent systems. In *Proc. Fifth International Symposium on Asynchronous Circuits and Systems*, pages 125–134, April 1999.

[4] R. Manohar and A. J. Martin. Slack elasticity in concurrent computing. In *Proc. Fourth International Conference on the Mathematics of Program Construction LNCS 1422*, pages 272–285, June 1998.

[5] R. Manohar and A. J. Martin. Pipelined mutual exclusion and the design of an asynchronous microprocessor. Technical Report CSL-TR-2001-1017, Cornell University, 2001.

[6] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[7] A. J. Martin. Formal program transformations for VLSI circuit synthesis. In E. W. Dijkstra, editor, *Formal Development of Programs and Proofs*, UT Year of Programming Series, pages 59–80, 1989.

[8] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus. The design of an asynchronous microprocessor. In *Proc. Decennial Caltech Conference on VLSI*, pages 351–373, 1989.

[9] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T.-K. Lee. The design of an asynchronous MIPS R3000. In *Proceedings of the 17th Conference on Advanced Research in VLSI*, pages 164–181, 1997.