Reassessing the causes of asynchronous systems performance

Alain J. Martin
California Institute of Technology

Abstract. In the course of the past 25 years of research in asynchronous VLSI, several complex systems, mainly microprocessors, have been designed and successfully fabricated to demonstrate the proposed technology.

Most prototypes have displayed a set of unique advantages. Some advantages, such as improved robustness to parameter variations, can be directly attributed to the quasi delay-insensitivity of circuits without clocks. But the remarkable performance of some large asynchronous chips in the combined space of speed and power has been offset by the mediocre behavior of some other comparable asynchronous prototypes. This suggests that some other factor may be at play.

The question raised here is the following. What if the good performance of complex asynchronous systems were not due to the absence of a clock?

Having been for many years an ardent proponent of asynchronous logic, I will happily slip into the role of the iconoclast and propose that the excellent performance of several asynchronous systems may not be (mainly) due to asynchrony (the absence of clock).

My argument concerns the design of at least moderately complex systems, typically a microprocessor. (I am not talking about single components such as a FIFO or an adder.) A sufficient number of such complete asynchronous systems have now been produced, and we can draw some conclusions. I will use as examples the projects I know best: the asynchronous systems developed at Caltech between 1988 and today. Other labs have produced asynchronous prototypes following an essentially similar approach: the Cornell asynchronous microcontrollers, the Grenoble ASPRO series, and the Fulcrum switches are the closest in design style.

What characterizes those approaches is a systematic top-down synthesis starting from a simple and, in principle, easily verifiable version of the device. I will call this approach “ab initio”, because no attempt should be made at the start to guess the final structure of the design. Rather, starting from this simple initial version, the final structure is reached through a sequence of semantic-preserving transformations. The goal of the transformations is usually to increase concurrency (pipeline depth, for example) and at the same time to replace large monolithic components with a number of smaller and smaller ones. In the end, each component has a standard, easily implementable structure. This first set of transformations is not part of the compiling phase. The source and result of a transformation are in the same formalism. At Caltech, this formalism is the programming language CHP. It is only at the end of this first phase, when all components are of the appropriate size and the decomposition is expected to deliver the target performance, that the second phase, which can be called compilation, transforms each CHP component into an asynchronous circuit.

The quality of the formalism (CHP, for example) is crucial to the method, as it must allow an algebraic style of manipulations of the representation of the system, and it requires us to look at a programming language not so much as a "description language," but rather as a formal notation.

Other formalisms, such as Petri Nets or STG ("signal transition graphs") have been used with success as well. Their graph-base model gives them a flexibility that language-based models sometimes miss. My personal reticence with such tools is that they do not scale well with the size of the systems to be represented. Who can give me the Petri Net or STG describing a complete microprocessor?

All systems engineered following this general approach have produced excellent results in terms of energy efficiency, robustness to parameter variations, combined energy and delay performance. All were functioning correctly on first silicon, often over a wide range of power supplies, including sub-threshold. Other groups using a similar approach have reported similarly good results.

But several asynchronous systems developments have followed a different approach. Whereas the final implementation of the components as asynchronous circuits followed a method essentially similar to the first one, the overall system decomposition was quite different. The starting point was usually a pre-existing synchronous architecture – or at least a standard, generic, synchronous pipeline. Without mentioning specific cases, the general observation is that asynchronous systems developed along this path often produced unremarkable results, not better and sometimes worse than their synchronous counterpart – except where it concerns attributes specific to asynchrony, such as robustness to parameter variations.

The discrepancy between the two methods has been a surprise to me, as it should be to all those interested in the technology. Once, a proponent of the second approach was so surprised by the results claimed for the Caltech MiniMIPS (an asynchronous version of a MIPS R3000 microprocessor) that he suggested at an NSF meeting that the performance figures we reported could absolutely not be true, and must be fraudulent…

So what may be causing the performance gap between the two approaches? Both use asynchronous logic as circuit implementation, albeit with some significant differences: the circuits of the ab-initio approach are usually of the QDI flavor, whereas the circuits of the second approach are usually not. But that difference is not significant enough to explain the discrepancy. So if the excellent performances of the ab-initio approach are not due to asynchrony, they may be the result of the drastically different system-level design.

At the system level, the main design issue is complexity – and, in particular, in the case of a modern microprocessor, complexity due to intricate concurrency and communication, which are crucial to achieving any kind of performance. In a traditional approach, the engineer confronted with this complexity has only one safe way out: to stick as closely as is possible to what he or she already knows might work. In the absence of a correct-by-construction method, drastically new explorations of alternative solutions are too risky and too time consuming.

In the ab-initio approach, many (possibly all) alternative solutions can be generated with the emboldening knowledge that they are all correct. A thorough exploration of the solution space is achievable by the algebraic manipulation of an abstract object (a CHP program) that is a correct representation of the complete design. Certainly, asynchrony greatly facilitates the application of this synthesis method by postponing dealing with the physical issues as long as possible.

Another argument is that, by positioning the high-level design squarely within the realm of distributed computing, all the tools (program constructs, proof methods) and techniques of concurrency can be brought to bear on the problem with much better results than those achieved by traditional HDLs.

In the end, it is conceivable that asynchrony is mainly a powerful enabler for a concurrency-based approach to digital system design – and that this approach itself represents a fundamental paradigm shift from traditional methods.