Can Asynchronous Techniques Help the SoC Designer?

Alain J. Martin *
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125, USA

July 7, 2006

Introduction

As technological advances make it possible to integrate an entire system on a single die, the designer of a system-on-chip (SoC) is confronted with increasing difficulties concerning complexity, reliability, energy and power consumption, and clock distribution. All those issues are aggravated by increasing parameters variability as a result of the same technological advances. This paper argues that because of the quasi-independence of asynchronous (QDI) circuits of timing, asynchronous logic alleviates the problems posed by parameter variability, and eliminates the clock distribution problem altogether. Furthermore, as some researchers attempt to turn the liability into an asset by exploiting parameter variability to design truly probabilistic computation, the flexibility and time-independence of asynchronous logic could be a natural match.

*The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, and monitored by the Air Force Office of Scientific Research.

1 Design Variability

Besides the plain complexity of the systems which can be integrated today on a single SoC, parameter variability—so-called PVT variability as a reference to process, supply voltage, and temperature—constitutes one of the main challenges to the designer. Since several papers at this conference already address this problem, there is no need for us to go into much detail here. The important point to make is that any variation that does not cause a circuit to stop functioning altogether will affect the timing behavior of the circuit and possibly its energy consumption. An example is threshold voltage drops caused by doping levels variation, which affect both the switching time and the leakage current.

Even more insidious are the parameter variations over the lifetime of the device, "device aging"as it is called, due to degradation caused by several factors like hot-carrier, electromigration, etc. Now, a well-tuned system may start malfunctioning over time as some part of the system affected by device aging may fail to meet

the clock period requirement.

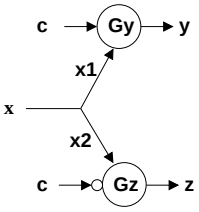
2 Asynchronous and QDI Design

A circuit is called asynchronous when no clock is used to implement sequencing. Being asynchronous does not necessarily mean that the circuit is time-independent (*delay-insensitive* or DI is the term used). In fact, most families of asynchronous logic use delay elements, or knowledge of the relative delays along different paths, to make sure that certain transitions take place before others. A "delay-insensitive" circuit is an asynchronous circuit in which no assumption is made on the delays in operators and wires for the correct operation of the circuit. Delay insensitivity was recognized early on as a desirable attribute of asynchronous circuits, and much effort was spent on designing DI circuits. Without success: in 1990, it was proved that the class of entirely DI circuits is very limited. Practically all circuits of interest fall outside the class. For instance, it is impossible to design even the simplest microcontroller as a DI circuit. Somewhere, some delay assumption has to be introduced. It is admitted (but not proved) that the weakest form of delay assumption with which any Turing-computable function can be built is the *isochronic fork*. Asynchronous circuits that are entirely DI except for isochronic forks are called "quasi-delay-insensitive" or QDI.

3 Isochronic Fork

In order to fully appreciate the quasi-delay-insensitivity of asynchronous circuits (and therefore their robustness to variability), let us go into a little more detail in describing isochronic forks and their timing requirement, as it is often misunderstood as being much more severe than it actually is.

A computation implements a partial order of signal transitions. In an asynchronous circuit without timing assumption, this partial order is based on a causality relation. For example, transition $x\uparrow$ causes transition $y\downarrow$ if x is an input to the gate with y as output, and x becoming true causes y to become false. Transition $y\downarrow$ is said to *acknowledge* transition $x\uparrow$. We do not have to be more specific about the precise ordering in time of transitions $x\uparrow$ and $y\downarrow$. The acknowledgment relation is enough to introduce the desired partial order among transitions, and to conclude that $x\uparrow$ precedes $y\downarrow$. As a consequence, a necessary condition for an asynchronous circuit to be delay-insensitive is that all transitions are acknowledged.



The class of computations in which all transitions are acknowledged is very limited. Consider the example of the figure, which summarizes a common data-dependency case. Signal x is forked to $x1$, an input of gate Gy with output y, and to $x2$, an input of gate Gz with output z. A transition $x\uparrow$ when c (the "data") holds is followed by a transition $y\uparrow$, but not by a transition $z\uparrow$, i.e. transition $x1\uparrow$ is acknowledged but transition $x2\uparrow$ is not, and vice versa when $\neg c$ holds. Hence, in either case, a transition on one output of the fork is not acknowledged.

In order to guarantee that the unacknowledged transition completes without violating the specified order, a timing assumption called the *isochronicity assumption* has to be introduced, and the forks that require that assumption are called *isochronic*. (Not all forks in a QDI cir-

cuit are isochronic.) The timing assumption on isochronic forks is a one-sided inequality that can always be satisfied: It requires that the delay of a single transition be shorter that the sum of the delays on a multi-transition path (typically containing at least 5 transitions).

The interested reader is referred to [1] for an in-depth presentation of asynchronous techniques for SoC design.

4 Robustness to Timing Variation

The weak and one-sided timing requirement of QDI circuits makes them extremely robust to timing variations, whatever their cause. This robustness has been checked by simulation at the electrical level, and also in actual chips. For instance, several layout errors lead to timing variations that would have been fatal in a clocked circuit but were tolerable in a QDI one. We will mention two: in the first asynchronous microprocessor, a student forgot to connect some wells to the proper voltage source and left them floating. In a clocked circuit, the gates inside this well would not have switched at the required speed to meet the clock period requirement. Because the implementation was asynchronous, the slow gates were not noticed. In the QDI MiniMIPS, another student forgot a very long and resistive poly wire in a critical part of the circuit: the component that fetches instructions from the cache. Here, the error was noticed as a significant loss of performance (from 280 MIPS down to 200MIPS) followed.

Which brings us to the main point concerning the timing robustness of QDI circuits. Timing variations in a QDI circuit will never lead to failure, unless they are so extreme as to violate the weak timing requirement of some isochronic fork. But they can affect the performance of the system if the timing variation takes place on a part of the system that is exercised frequently. When the timing variation occurs in a part of the system that is not exercised frequently, the effect on performance will be negligible.

4.1 Clock Trees, GALS, and Metastability

One of the biggest challenges caused by design variation in today's technology is clock tree synthesis, as any variation on the clock tree circuitry may have disastrous effect on the timing of sycnhronous system. Such concerns are leading to extremely complex analysis and designs for clock trees. Obviously, since QDI circuits don't use clocks, this problem is completely eliminated.

This advantage has not been lost on the designers of SoCs, who are currently advocating a globally asynchronous and locally synchronous (GALS) approach that eliminates the global clock. A GALS system is composed of independently clocked synchronous components that are communicating with each-other asynchronously. Several architectures for GALS have been proposed already, and they may be a good stopgap measure for a while. But we believe there is an inherent weakness in the approach that will eventually kill it.

Since the interconnects are asynchronous and each component operates with its own clock, any communication between a component and its environment requires that the asynchronous signal from the environment be synchronized with the local clock through an asynchronous/synchronous interface. All such interfaces use on one of two techniques: the pausable clock or the synchronizer. Whichever technique

is used, they all exhibit *metastability*. At some point, a choice must be made between two unsynchronized signals, and it has been proved that such a choice cannot be made within a given time period: If the two signals arrive within a very small time interval from each other, the device (the "arbiter") that is making the choice may take an arbitrary amount of time to make the choice. (This undecided state is called the metastable state). It is possible to carefully design the arbiter in such a way that the probability that the metastable state lasts longer than the chosen clock period is low enough to be acceptable. But such a design requires that the device behaves according to spec.

Unfortunately, two factors play against the approach. First, as SoCs grow in size and clock distribution issues become more acute, the tendency is towards a larger number of synchronous components, necessitating a larger number of arbiters and synchronizers, hence multiplying the probability of a misbehavior. Secondly, the increasing variability of the devices may cause metastability to occur more frequently than the careful calculations predicted if the parameters of the arbiters have changed. Soon, dealing with metastability will be at least as difficult as building a global clock tree, and therefore GALS will offer no advantage. The only solution left will be a totally asynchronous approach.

5 Energy Efficiency of Asynchronous Circuits

Automatic Shut-off of Idle Parts: An asynchronous system is purely reactive. It is a collection of communicating processes (modules). A process is idle until it receives some messages carrying information about a task to be performed. It then performs the task, usually sends the results of the computation to one or several other processes, and then returns to the idle state until a new message arrives. *When a module is not computing, it doesn't spend any dynamic energy: no transistor is switching.* By gating the clock, some synchronous systems achieve a similar effect. However, the results are far from perfect compared to the perfect shut-off obtained in the asynchronous case. Deciding when a part can be shut off may in itself be a complex task, in particular with data dependencies, that limits the extent to which shut-off can be performed in clocked design.

Automatic Glitch Elimination: Because the signals generated by an asynchronous circuit can be observed at any time, those signals must always have well-defined values: A signal is either stable high, or stable low, or in a quasi-monotonic transition from one stable value to the other one. This monotonicity requirement imposes an entirely glitch-free design style. It so happens that glitches can cause spurious transitions which, in clocked circuits, may account for up to 40 percent of the power consumption in combinational circuits like arithmetic circuits.

Global Clock vs. Local Handshakes: In a synchronous circuit, clock activity may account for up to 50 percent of the power consumption. In an asynchronous circuit, the global sequencing implemented by the clock is replaced by local handshake protocols between adjacent modules. These protocols are complex and contribute to about one third of the transitions in a pipeline like that of the MIPS. But because the transitions are local, the wires are shorter and thus less capacitance is switched. As a result, the energy cost is less than that of the global clock activity. We estimate that the energy saved by replacing a clock distribution tree with an asynchronous

handshake mechanism is between a third and a half of the energy spent in the clock circuitry.

Variable Voltage Supply: Another reason for the energy-efficiency of asynchronous circuits, which is most important to us here, is that asynchronous (in particular QDI) circuits automatically adjust their operating speed to environmental conditions. This is a huge advantage. We don't need to carry out complex trials and statistical guesswork to figure out what voltage and clock speed we should run our chips at in order to leave an appropriate margin. In a clocked system, the concurrent adjustment of the clock together with the supply voltage is a difficult operation that does not produce optimal results.

6 Conclusion

QDI circuits are very robust to variations in timing and therefore immune to most PVT variations. They are also able to adapt automatically to timing variations caused either by accidental factors, or by deliberate introduction of randomness, or by voltage adaptation. Because of the absence of a global clock reference, they are very modular and extensible. For all those reasons, we expect asynchronous and in particular QDI logic to become the technology of choice for SoCs.

References

[1] A.J. Martin, M. Nyström. Asynchronous Techniques for System-on-Chip Design. *Proceedings of the IEEE*, invited paper, to appear 2006.