Asynchronous Logic for High Variability Nano-CMOS

Alain J. Martin
California Institute of Technology
Pasadena, CA 91125, USA

***Abstract*—At the nanoscale level, parameter variations in fabricated devices cause extreme variability in delay. Delay variations are also the main issue in subthreshold operation. Consequently, asynchronous logic seems an ideal, and probably unavoidable choice, for the design of digital circuits in nano CMOS or other emerging technologies. This paper examines the robustness of one particular asynchronous logic: quasi-delay insensitive or QDI. We identify the three components of this logic that can be affected by extreme variability: staticizer, isochronic fork, and rings. We show that staticizers can be eliminated, and isochronic forks and rings can be made arbitrarily robust to timing variations.**

INTRODUCTION

All future technologies, nano-CMOS as well as potential emerging technologies like carbon nanotubes or printed electronics, will face great fabrication challenges that will translate into important parameter variations and decreased reliability [1], [2]. All parameter variations that do not cause the circuit to fail affect the timing. Another challenge to the CMOS designer is subthreshold operation. Operating a circuit at subthreshold voltage is an attractive option to further reduce energy consumption. However it is usually accompanied with large delay fluctuations because of the exponential dependency of the Ids current on Vdd and Vth, combined with Vth variations [3].

Therefore, a design approach that is (mostly) independent of delays for the logical correctness of the circuits produces designs that are much more robust to parameter variations thatn traditional approaches. Because of its weaker dependence on delay, asynchronous logic seems an ideal and perhaps unavoidable choice for digital circuits in the nanoscale era, in particular the *quasi-delay insensitive* (QDI) approach that relies on the weakest possible timing assumption (isochronic fork). In fact, all QDI circuits designed at Caltech have shown remarkable robustness to Vdd, Vth and temperature variations. All (including several microprocessors) could operate over a wide range of voltages, *including subthreshold voltage* without particular precautions. SPICE simulations also show that threshold voltage variations between twice the nominal voltage value and zero volt, are tolerated. The question addressed in this paper is the following. In the presence of extreme PVT (process, voltage, temperature) parameter variations, how will a QDI circuit eventually fail, and what can be done to avoid the failure or at least significantly improve the circuit's robustness?

In order to answer this question, we analyse the different components of a correct QDI system to assess its robustness to parameter variations. We first examine the individual gates (operators) that are the building blocks of a QDI circuit. We then look at the way in which the operators are composed to form a system.

I. IMPLEMENTING QDI OPERATORS

We distinguish two types of operators: combinational and state-holding. A complete logic family for high performance circuits—like, e.g., the circuits of the Caltech MiniMIPS (see [6])—requires only nand, nor and inverter in terms of combinational gates, plus the non-standard write-acknowledge circuit (*wack*).

The implementation of combinational gates presents no problem besides the restriction on the number of transistors in series. The decomposition of a large gate into a network of smaller gates that satisfy the restriction is part of the QDI synthesis procedure.

The number of state-holding elements used in QDI logic is surprisingly small. We can design any digital circuit (complete microprocessor) with just the 2-input C-element, the set-reset latch for memory and registers, and the precharge function. However, the standard CMOS implementation of state-holding operators in this technology does present a serious problem: the weak inverter of the staticizer. Consider the operator defined by the two *production rules*:

$$A \rightarrow x\uparrow$$
$$B \rightarrow x\downarrow$$

(A and B, the "guards" of the production rules, are boolean expressions in terms of the input variables of the operator: When A holds, x is set to true, when B holds, x is set to false.) Without loss of generality, let's assume that it is implemented as

$$A \rightarrow x_{-}\downarrow \qquad x_{-} \rightarrow x\downarrow$$
$$B \rightarrow x_{-}\uparrow \qquad \neg x_{-} \rightarrow x\uparrow$$

By definition of a state-holding element, there exist states in the computation where $\neg A \wedge \neg B$ holds. In those "floating" states, the path to Vdd through the pullup implementing B, and the path to GND through the pulldown A are both cut. The voltage value of the physical node implementing x_{-} has to be maintained in other ways. The general approach is to find another path to Vdd and another path to GND to maintain the current value of x_{-}. The simplest and crudest solution is to add a "staticizer" (or "keeper"). The staticizer implementation consists of *always* maintaining the current value of x_{-} by adding the pullup $\neg x \rightarrow x_{-}\uparrow$ and the pulldown $x \rightarrow x_{-}\downarrow$, giving the gate:

$$A \vee x \rightarrow x_{-}\downarrow$$
$$B \vee \neg x \rightarrow x_{-}\uparrow$$

The added circuitry is a feedback inverter with input x and output x_{-}. The difficulty with this solution is that when the value of x_{-} has to be changed, for instance from true to false, there is a conflict (a "fight") between $A \rightarrow x_{-}\downarrow$ and $\neg x \rightarrow x_{-}\uparrow$. Symmetrically, when the value of x_{-} has to be changed from false to true, there is a "fight" between $B \rightarrow x_{-}\uparrow$ and $x \rightarrow x_{-}\downarrow$. There is no logical resolution to those conflicts; they can only be resolved by physical means by making sure that the current through the pullup implementing B and through the pulldown implementing A is stronger than the current through the transistors of the feedback inverter. This is usually achieved by implementing the two transistors of the feedback inverter as highly resistive ("weak"), making the currents through the pullup and the pulldown of the feedback inverter much smaller than the currents through the pullup B and pulldown A. But the weak currents cannot be too weak since they have to maintain the voltage on node x_{-} in the presence of possibly important leakage.

Hence, the correct behavior of the staticizer relies on a two-sided inequality requirement on the feedback inverter's current. In a technology with high parameter variations, the requirement for the implementation to tolerate a variation of for instance 3σ makes it impossible

978-1-4244-5091-6/09/$25.00 ©2009 IEEE

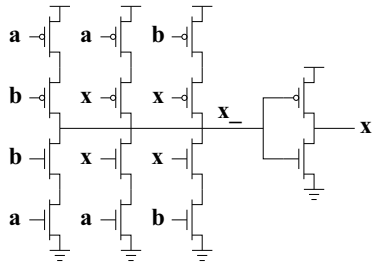

Fig. 1. A two-input C-element implemented as a majority gate.

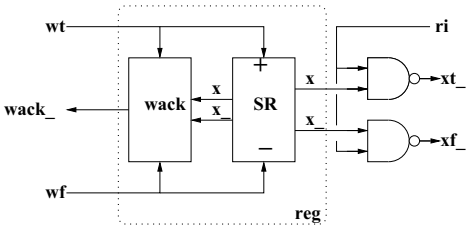

Fig. 2. Dual-rail single-bit register with read and write interfaces

to satisfy the two-sided inequality. *We conclude that it will not be possible to use staticizers in nano-CMOS.*

We have to use another solution to maintain the value of the output nodes in the floating states. The general method consists in identifying the floating states in which the value of the output has to be maintained and using the feedback inverter *only in those states.* Without additional information, the floating states are characterized by $\neg A \wedge \neg B$, leading to the general solution:

$$A \vee \neg B \wedge x \;\rightarrow\; x\_\downarrow$$
$$B \vee \neg A \wedge \neg x \;\rightarrow\; x\_\uparrow$$

A. C-element

The Muller C-element is an essential building block of asynchronous logic. The 2-input C-element with inputs a and b and output x is defined as:

$$a \wedge b \;\rightarrow\; x\uparrow$$
$$\neg a \wedge \neg b \;\rightarrow\; x\downarrow$$

The above transformation applied to this pair of PRs leads to the well-known majority-gate implementation (see Figure 1):

$$a \wedge b \vee a \wedge x \vee b \wedge x \;\rightarrow\; x\_\downarrow$$
$$\neg a \wedge \neg b \vee \neg a \wedge \neg x \vee \neg b \wedge \neg x \;\rightarrow\; x\_\uparrow$$
$$x\_ \;\rightarrow\; x\downarrow$$
$$\neg x\_ \;\rightarrow\; x\uparrow$$

For the three-input C-element, the transformation gives:

$$a \wedge b \wedge c \vee a \wedge x \vee b \wedge x \vee c \wedge x \rightarrow x\_\downarrow$$
$$\neg a \wedge \neg b \wedge \neg c \vee \neg a \wedge \neg x \vee \neg b \wedge \neg x \vee \neg c \wedge \neg x \rightarrow x\_\uparrow$$

B. Read/Write Register

The read/write register is used in a standard pipeline stage of the Caltech Asynchronous Microprocessor (CAM). It is also used in a slightly different form to implement general-purpose asynchronous registers.

As shown in Figure 2, in its simplest form, the register consists of three part: the set/reset latch maintaining the current value of the register bit $(x, x\_)$, the two nand-gates that constitute the read part, and the write-acknowledge (*wack*) cell that generates the acknowledge signal for the write handshake. The implementation of the read part presents no difficulty. The set/reset latch can be implemented as cross-coupled nor-gates which do not need relative sizing unlike the implementation with cross-coupled inverters (see [4]). Only the *wack* needs some attention. Its specification is

$$wt \wedge x \vee wf \wedge x\_ \;\rightarrow\; wack\_\downarrow$$
$$\neg wt \wedge \neg wf \;\rightarrow\; wack\_\uparrow$$

The floating states are $\neg wt \wedge wf \wedge \neg x\_$ and $wt \wedge \neg wf \wedge \neg x$. In both states, the write interface is in the process of changing the values of x and $x\_$ and therefore the output $wack\_$ should be kept high in the floating states. The transformation then gives

$$\neg wt \wedge \neg wf \vee \neg wt \wedge \neg x\_ \vee \neg wf \wedge \neg x \;\rightarrow\; wack\_\uparrow$$

Now the two guards are complementary. *The write-acknowledge is a combinational gate.* Altogether the above design leads to a robust layout for the register. No feedback inverter is needed, the only feedback being that of the cross-coupled nor-gates, and there are at most two transistors in series on all paths. (However, it is difficult to use the cross-coupled nor-gates in an SRAM implementation where compactness is paramount.)

C. Precharge Function

The precharge function is a state-holding cell with two sets of inputs: the input en is a control signal used to precharge the node $z\_$ high in the neutral state ($\neg en$). The other set X of inputs is used to compute the boolean function F when en is high. Function F is necessarily small enough to fit in a single pulldown, and therefore the number of inputs of X is also limited—rarely more than four. The PRS for the precharge function cell is:

$$\neg en \;\rightarrow\; z\_\uparrow \qquad z\_ \;\rightarrow\; z\downarrow$$
$$en \wedge F \;\rightarrow\; z\_\downarrow \qquad \neg z\_ \;\rightarrow\; z\uparrow$$

The floating state is $en \wedge \neg F$. Applying the transformation and after simplification, we get:

$$\neg en \vee \neg F \wedge \neg z \;\rightarrow\; z\_\uparrow$$
$$en \wedge F \vee en \wedge z \;\rightarrow\; z\_\downarrow$$

For example, if F is the dual-rail equality of a and b, i.e. $f = a0 \wedge b0 \vee a1 \wedge b1$, the complement of F is $(\neg a0 \vee \neg b0) \wedge (\neg a1 \vee \neg b1)$. This leads to 5 different pullup paths with 3 transistors in series (including $\neg z$).

The difficulty of this general solution is that the structure of function F may create a complementary function $\neg F$ with too many conjuncts in a term of the DNF representation of the function. But it should be observed that the added pullup $\neg F \wedge \neg z$ is used only to maintain the value true of $z\_$ in the floating state and not to switch the output. Therefore, the possible complexity of $\neg F$, which translates into a weak drive (I_{ds} current), might be acceptable. However, as already argued, the I_{ds} current cannot be too weak, and therefore we need a method to decompose the precharge function implementation when $\neg F$ is too complex. Such a decomposition exists.

D. General Decomposition of the Precharge Function

Without loss of generality, consider a function F with 3 terms in its "complete" DNF (meaning that each minterm contains all literals): $F = f1 \vee f2 \vee f3$. Its complement has 3 conjuncts in each term of its DNF, which is too many since the pullup $\neg F \wedge \neg z$ has a chain of 4 p-transistors. The specification of the PCF(F) is:

$$*[[en \wedge (f1 \vee f2 \vee f3)]; z\uparrow; [\neg en]; z\downarrow]$$

Because at most one term f_i of F is true at any time, the following decomposition is equivalent to the above specification:

$$
\begin{array}{l}
(\ *[[en \wedge f1]; z1\uparrow; [\neg en]; z1\downarrow] \\
\|\ *[[en \wedge f2]; z2\uparrow; [\neg en]; z2\downarrow] \\
\|\ *[[en \wedge f3]; z3\uparrow; [\neg en]; z3\downarrow] \\
\|\ *[[z1 \vee z2 \vee z3]; z\uparrow; [\neg z1 \wedge \neg z2 \wedge \neg z3]; z\downarrow])
\end{array}
$$

where $z1$, $z2$, $z3$ are intermediate variables. Because at most one term f_i of F is true at any time, only one input z_i of the nand-gate is true at any time. Therefore, the nand-gate can be decomposed arbitrarily to satisfy the limitation on transistor-chain length without violation the QDI requirement of stability.

However, the following simplification significantly improves the circuit. It applies whenever the dual-rail version of the function has to be computed, i.e.:

$$
\begin{array}{llll}
\neg en & \rightarrow\ zt\_\uparrow & \neg en & \rightarrow\ zf\_\uparrow \\
en \wedge Ft & \rightarrow\ zt\_\downarrow & en \wedge Ff & \rightarrow\ zf\_\downarrow
\end{array}
$$

If the only floating states are $en \wedge Ft \wedge \neg Ff$ for the Ff block, and $en \wedge Ff \wedge \neg Ft$ for the Ft block, then we can staticize the PRS as:

$$
\begin{array}{l}
\neg en \vee \neg zf\_ \rightarrow\ zt\_\uparrow \\
\neg en \vee \neg zt\_ \rightarrow\ zf\_\uparrow
\end{array}
$$

with the pulldowns unchanged. This simplification requires that $\neg en \vee Ft \vee Ff$ be an invariant of the system. In other words, the condition for applying the transformation is that *the function inputs are not reset when en is set*. Certain QDI templates already satisfy this condition. It is the case for the control/data decomposition scheme of the pipeline stage used in the CAM. (It relies on an easily satisfiable isochronicity assumption.) Other QDI templates can easily be transformed to satisfy the invariant. It is the case for the PCHB ("precharge half-buffer") of the MiniMIPS design style. Since the validity (usually called $v(L)$) of the data input is always computed, it suffices to replace the enable signal en with en' defined as $v(L) \underline{C} en$ (i.e. en' is the output of the C-element with en and $v(L)$ as inputs).

II. Isochronic Fork

Once all QDI building blocks have been defined, the next step is to compose them into a working system. Can we satisfy the timing requirements of isochronic forks? In the past, a simple-to-explain sufficient timing condition has often been used to implement the isochronic fork; but, in the presence of important timing variations, this sufficient timing condition is difficult to implement as it relies on a two-sided timing inequality. The designer should switch to a more complex but easier to satisfy necessary and sufficient condition. This condition relies on the notion of *adversary path*. It is a one-sided timing inequality stating that a single transition should be shorter in time than a sequence of multiple transitions (usually five in CMOS). (For a formal proof of this result, see [7].)

Let us first review how the need for isochronic forks arises. A computation implements a partial order of transitions. In the absence of timing assumptions, this partial order is based on a causality relation. For example, transition $x\uparrow$ causes transition $y\downarrow$ in state S if and only if $x\uparrow$ makes guard By of $y\downarrow$ true in S. 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 $y\downarrow$. In an implementation of the circuit, gate Gx with output x is directly connected to gate Gy with output y, i.e., x is an input of Gy.

Hence, a necessary condition for an asynchronous circuit to be delay-insensitive is that all transitions are acknowledged.

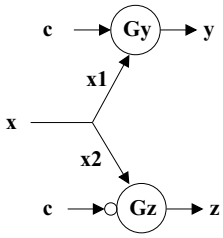

Fig. 3. The fork (x,x1,x2) is isochronic: a transition on x1 causes a transition on y only when c is true, and a transition on x2 causes a transition on z only when c is false. Hence, certain transitions on x1 and on x2 are not acknowledged, and therefore a timing assumption must be used to guarantee the proper completion of those unacknowledged transitions.

Unfortunately, the class of computations in which all transitions are acknowledged is very limited, as we proved in [5]. Consider the following example in which the specification of the computation requires an ordering of transitions on variables x, y, and z defined by the sequence:

$$\ldots\ x\uparrow;\ \ y\downarrow;\ \ \ldots; x\uparrow;\ \ z\downarrow\ldots$$

Implementing this sequence of transitions requires introducing at least one control variable c to distinguish the states in which $x\uparrow$ causes $y\uparrow$ from the states in which $x\uparrow$ causes $z\uparrow$, leading to PRs of the form:

$$
\begin{array}{l}
c \wedge x \ \rightarrow\ y\downarrow \\
\neg c \wedge x \ \rightarrow\ z\downarrow
\end{array}
$$

As shown in Figure 3, x as the output of gate Gx 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 holds is followed by a transition $y\downarrow$, but not by a transition $z\downarrow$, 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 forks*[5]. The branch of the fork with the non-acknowledged transition is called an *isochronic branch.* (Not all forks in a QDI circuit are isochronic.)

A. The Weakest Isochronicity Assumption

First, it is important to realize that the weakest timing assumption associated with an isochronic fork is not a relation between the delays on the different branches of the fork—which would be a very tight assumption indeed. For example, in the case of a fork with two branches, if $\delta(t1)$ and $\delta(t2)$ are the delays of transition $t1$ on one branch and transition $t2$ on the other branch, the timing assumption is NOT that $|\delta(t1) - \delta(t2)| \leq \epsilon$. As already mentioned, such a timing requirement has been used because of its simplicity. It is sufficient but not necessary, and is very difficult to fulfill in the presence of large parameter variations, in particular threshold voltages.

In order to characterize the weakest (necessary and sufficient) timing condition for an isochronic fork to behave properly, let us examine how a circuit will fail when a transition on an isochronic branch fails to complete. In our previous example, assume that $x1\uparrow$ causes $y\downarrow$, but $x2\uparrow$ does not complete. Since $x2\uparrow$ does not change the value of z, initially the failure of $x2\uparrow$ to complete does not cause a failure of gate Gz. But $y\downarrow$ causes a sequence of transitions that will eventually change an other input of Gz, say transition t since we do not know the sense of this transition. At this point, transition $x2\uparrow$ not having completed will cause a misfiring of Gz, since in the partial-order specification of the circuit, $x2\uparrow$ should have completed.

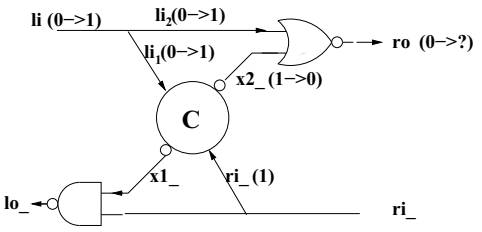

Fig. 4. The fork (li,li1,li2) is isochronic, with li2 as isochronic branch for the transition 0 to 1 on li

Hence, the weakest isochronicity assumption is that transition $x2\uparrow$ completes before the sequence of transitions starting at $x1\uparrow$ and ending at t. This sequence of transitions defines a path in the circuit called the *adversary path* of isochronic branch $x2$. The isochronicity assumption states that the delay $\delta(x2\uparrow)$ for unacknowledged transition $x2\uparrow$ to complete be always smaller than the sum of the delays of the transitions on the adversary path (including acknowledged transition $x1\uparrow$).

This one-sided inequality can always be satisfied by making the adversary path longer. SPICE Monte Carlo simulations of a typical QDI circuit in IBM 65nm CMOS 10 LPe show that an adversary path of length 5 will fail once in 10,000 at Vdd = 400mV. But an adversary path of length 7 will fail only once in 2 million trials at Vdd = 300mV.

B. A Worst-case Example

A worst-case example, where the adversary path contains just one gate, is the Caltech "Q-element" (see [4]) shown in Figure 4. Its specification in terms of boolean transitions on its external variables is: $*[lo\_\downarrow; li\uparrow; lo\_\uparrow; li\downarrow; ro\uparrow; ri\_\downarrow; ro\downarrow; ri\_\uparrow]$ (where $*[...]$ means "repeat forever.")

We analyze the isochronic fork $(li, li1, li2)$. Initially, $x2\_$ is true and ro is false. In order to satisfy the specification, transition $li\uparrow$ should not change the value of ro. But since the inverted output of the C-element is the input $x2\_$ of the nor-gate the adversary path ($li1$, C-element, $x2\_$) causes transition $x2\_\downarrow$, which could cause transition $ro\uparrow$, unless the isochronic-branch transition $li2\uparrow$ terminates before $x2\_\downarrow$. This is one of the tightest isochronicity requirements since the adversary path contains only one gate, the C-element. In an implementation of the C-element as a majority gate, the inverted output has to be taken after two inverters, making the adversary path safer (3 transitions).

III. Oscillating Rings

A QDI system is nothing but an interlocking of oscillating rings. Therefore, a correct implementation of a QDI system requires to guarantee that the rings of restoring gates keep oscillating, in the same way that an odd number of inverters connected in a ring will keep oscillating if properly designed. Two conditions, called *stability* and *non-interference* are necessary and sufficient to guarantee that a QDI circuit is free of hazards (incomplete or erroneous transitions). Rule $Au \rightarrow x\uparrow$ is stable in a computation if Au can be invalidated only in the states where x is true. (In other words, once Au becomes true it remains true until $x\uparrow$ fires.) Non-interference states that $Au \rightarrow x\uparrow$and $Ad \rightarrow x\downarrow$ cannot fire in the same state, i.e. Au and Ad are never true in the same state.

Since hardware computations are non-terminating, each transition $z\uparrow$ is followed, after a number of other transitions, by transition $z\downarrow$, and vice versa. Since the guards Bu and Bd of those transitions are mutually exclusive, the chain of transitions between $z\uparrow$ and $z\downarrow$ must contain a transition that invalidates Bu. Hence, transition $z\uparrow$ invalidates its guard Bu through a sequence of intermediate transitions. Can we still say that Bu is stable? Is it possible that the effect of $z\uparrow$ propagates through the cycle of gates fast enough to invalidate itself? At the electrical level, could the voltage $V(z)$ stabilize at an intermediate value close to $Vdd/2$?

Arguing that such a ring of operators is not self-invalidating is equivalent to arguing that the ring oscillates. This is an electrical property of the circuit that relates the slew rates of transitions, the gain of the operators, and the number of operators on the ring. We must require that any cycle of operators be implemented with a number of stages at least equal to a chosen minimum to guarantee that the cycle is not self-invalidating. In a CMOS circuit operating at nominal Vdd, three restoring operators with good gain are usually sufficient, although we usually require five to be safe. In other nano-technologies, the minimal number of operators on a ring will have to be determined based on the electrical properties of the technology.

IV. Conclusion

The goal of this paper was to establish the necessary and sufficient conditions to make QDI circuits practically immune to the large parameter variations that will affect the design of nano-CMOS systems and subthreshold operation. First, we have shown that there exist efficient static implementations of state-holding operators without weak feedback and with no more than two transistors in series in the pullups. Those implementations do not rely on specific current ratios or transistor sizing.

Second, the only necessary timing assumption, the isochronic fork assumption, has been refined, and we showed that it is a one-sided timing assumption that can always be satisfied. Third, the issue of oscillating rings of operators has been analyzed. It was established that in order to keep a ring oscillating, it has to contain a minimal number of restoring operators. This number is technology dependent.

The same results can be applied to making SRAM robust to parameter variations at the cost of an area overhead. If the designer insists on a 6T SRAM cell design, then the usual issues of read robustness and relative transistor sizing have to be dealt with, with increasing difficulty as the technology advances.

In conclusion, the reader should observe that the conditions we have formulated for the correct operation of a QDI system are different from what is required in a clocked system. In particular, the timing requirements in a clocked system are two-sided inequalities that are difficult to satisfy over a wide range of parameter values. This is where the robustness advantage of QDI lies, with the absence of clock.

Acknowledgments

The research described in this paper was supported by a grant from the National Science Foundation.

References

[1] Shekhar Borkar *et al.* Parameter Variations and Impact on Circuits and Microarchitecture. *DAC 2003*, pp338-342.

[2] K.Bowman *et al.* Impact of Die-to-Die and Within-Die Parameter Fluctuations on Maximum Clock Frequency Distribution for Gigascale Integration. *IEEE JSSC,* vol.37, no.2, 2002.

[3] S. Hanson *et al.* Ultralow-voltage, minimum-energy CMOS. *IBM J. Res & Dev.* vol 50, No 4/5. 2006.

[4] A.J. Martin, M. Nyström. Asynchronous Techniques for System-on-Chip Design. *Proc. of the IEEE*, Special Issue on Systems-on-Chip, 94, 6, 1089-1120. 2006.

[5] Alain J. Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. *Sixth MIT Conf. on ARVLSI,* MIT Press, 1990.

[6] Alain J. Martin *et al.* The Design of an Asynchronous MIPS R3000 Microprocessor. *Proc. 17th Conf. on ARVLSI*. IEEE Computer Society Press, pp. 164–181, 1997.

[7] Sean Keller, Michael Katelman, Alain J. Martin. A Necessary and Sufficient Timing Assumption for Speed-Independent Circuits. *Proc. 15th IEEE Int. Symp. on Asynchronous Circuits & Systems*, 2009.