

On Nanotechnology's Fundamental Scaling Limits: A Study On The Impact Of Robustness To Delay Uncertainty On A Design's Performance And Area

Stephen Bijansky, Andrey Zykov, and Margarida Jacome

Abstract—Nanoelectronic devices and interconnects will most likely exhibit substantial performance variability. Delivering high performance under such uncertainty conditions will require implementing more complex control and communication mechanisms between system components, in comparison to those used state of the art synchronous design. A key issue becomes thus to assess how the increased densities afforded by nanotechnologies will be countered by design overheads associated with achieving designs that are robust to performance variability. Towards this end, we consider two representative asynchronous design styles, one using asynchronous single rail channels (more suitable for low to moderate delay variability conditions), and another using asynchronous dual rail channels (more suitable for high delay variability), and assess their relative merits, for representative benchmarks. The experimental results of our study clearly exhibit the very substantial overheads/costs that will necessarily be incurred to achieve high performance under substantial delay uncertainty conditions, suggesting that such uncertainty may indeed be one of the fundamental scaling limits of nanotechnologies.

I. INTRODUCTION

Fast paced progress in fabrication and assembly of nanoelectronic devices suggests that it will be possible to manufacture large-scale computation nanofabrics within 10-15 years [1]–[4]. . Aside from greatly increasing concerns with complexity and scalability, nanoelectronic fabrics share two characteristics that will impact the entire design hierarchy. First, irrespective of the winning technologies, e.g., based on semiconductor nanowires and/or carbon nanotubes, it is widely recognized that devices and interconnects at the nanoscale will exhibit (1) a density of defects at least an order of magnitude greater than state-of-the-art silicon technologies [2], [3], [5], [6]; and (2) increased susceptibility to transient faults, due to reduced noise margins and several wearout mechanisms [2], [7]. Second, the timing and performance of devices and interconnects at the nanoscale are subject to significant variability due to (1) physical phenomena at the nanoscale, e.g., a single charge or defect may significantly impact the structural stability and response time of a nanodevice [8], [9]; (2) parametric variations introduced by limitations in the molecular techniques and manufacturing processes used for fabrication of future nanoelectronic devices [10], [11]; and (3) the possible use of reconfigurable fabrics to achieve defect-tolerance [12]. As a result, system designs based on nanoelectronic devices are likely to exhibit a substantial amount of performance variability. Thus, low reliability and performance variability are

intrinsic to nanoscale regimes and are here to stay, and loom large as potential showstoppers to enabling nanotechnologies.

Designs that overcome these challenges may incur substantial overheads in performance, power and area. For example, defect and fault tolerance can be achieved via redundancy, e.g., in data representation (coding), computation, communication, and storage. Similarly, as discussed in the sequel, the impact of performance variability (at the device-level and circuit-level) on a systems throughput may be smoothed by moving to an asynchronous design paradigm, yet this will require more complex control and communication mechanisms between system elements, as compared to traditional synchronous design. These considerations point to fundamental scaling limits, where the increased densities are countered by design overheads associated with achieving defect/fault tolerant designs that are robust to performance variability. In this paper, we will focus on one of the dimensions of the problem - namely, on assessing the overheads incurred by designs that must operate correctly under high delay uncertainty conditions.

Synchronous circuit design has been one of the key enablers underlying the phenomenal success of the semiconductor industry for the last 50 years. Its simplicity and scalability allow design complexity to be controlled via powerful abstractions exploited by top-down design methodologies and tools capable of efficiently synthesizing and testing increasingly complex systems. However, in moving towards deep submicron and nanotechnologies, several effects have started to erode the efficiency of synchronous design, namely, increased delay uncertainty and variability (see above), and prohibitive levels of power consumption. Indeed, the area and power expended to drive the clock signal across a chip with an acceptable skew is becoming overwhelming - clock distribution networks can be responsible for as much as 40% of the total power dissipation of a chip [13]. Furthermore, enforcing all elements of a design to work in lockstep, even when they have no useful work to do, may be very wasteful. Recently, power saving techniques have been incorporated to mitigate this problem, e.g., clock gating whereby a section of the design is disabled, but this incurs additional overhead and adds to the complexity of the designs. These trends call in to question the effectiveness of synchronous design, and have spurred renewed interest in asynchronous design.

Indeed, while the performance of a synchronous design is defined by the slowest component/path operating under worst case conditions, asynchronous designs have potential to deliver average case performance. Specifically, the operation of an

asynchronous circuit can immediately start when all of its inputs are available (rather than when the global clock signal arrives), thus its operational speed does not depend on the maximal delay of an entire circuit, but only the delays of active paths. This permits asynchronous circuits to operate as fast as possible in each instance, and to take advantage of delay variations across activated paths - this is particularly attractive when delay uncertainty/variability is high. Additionally, asynchronous designs are also attractive from the standpoint of power dissipation. Indeed, circuits are activated only when they have useful work to do, and power hungry global clock networks are eliminated. Unfortunately, asynchronous circuit design also poses substantial challenges. Namely, they require additional circuitry to implement explicit handshaking between system components. These may in turn have a substantial impact on average delay and power. Also, since there is no global clock to assert when signals are stable, hazards and glitches must be explicitly addressed in such designs.

So far, in state-of-the-art technologies, a combination of both styles seems to yield the best solutions. Namely, Globally Asynchronous Locally Synchronous (GALS) design styles have become increasingly popular in the context of large SoCs and microprocessors, whereby subsystems or system components run at their own clock frequency (forming clock domains) which communicate with each other asynchronously [14]. The advantages of GALS are a simplified clock distribution network with reduced clock power dissipation, the ability to run each domain at its optimal frequency, while individually performing dynamic voltage and frequency scaling, and the ability to still use conventional design and testing methods for most of the design. Indeed, if such clock domains are appropriately defined, the impact of inter-domain synchronization on system performance, and the area and power cost of the required synchronization circuitry, can be fully amortized by the above mentioned benefits. Fully asynchronous circuit or component designs have also been demonstrated in industry and academia, see e.g. [15], [16], yet the benefits relative to synchronous solutions have not yet been substantive enough to justify a costly change in paradigm.

As mentioned above, the key question framing the content of this paper is one of assessing fundamental scaling limits, where the increased densities afforded by nanotechnologies are countered by design overheads associated with achieving designs that are robust to performance variability. Specifically, when the degree of performance uncertainty/variability reaches a level such that building a reliable clock is no longer practical/viable, and thus more complex control and communication mechanisms between system elements are required (as compared to traditional synchronous design), what will be the overhead incurred by such designs and, given those, would the increased densities still pay-off?

Towards a first assessment of such 'costs', we consider two asynchronous design styles. One such style incurs a relatively small area overhead with respect to the communication and control circuitry required by 'traditional' synchronous design, and yet exhibits a severe performance degradation, as delay variability/uncertainty increases, since it relies substantially/heavily on timing assumptions. In contrast, the second

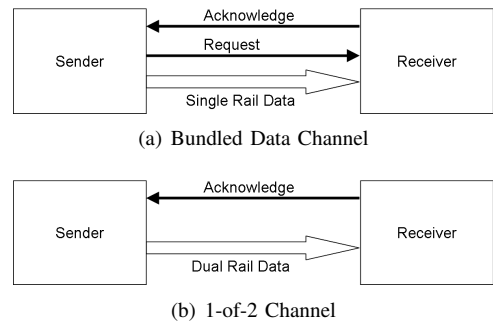


Fig. 1. Asynchronous Data Channels

design style can very effectively smooth out the deleterious effects of performance variability on overall system performance yet, unfortunately, at a very high cost in terms of extra control and communication circuitry. Our results will clearly show that, when technologies characterized by relatively low delay variability are targeted, the first (or 'lighter') design style (relying heavily on timing assumptions) exhibits by far the best performance and yet, as we increase delay variability, its performance rapidly degrades, and the area/cost of operating 'as fast as possible' in such circumstances becomes exceedingly high - captured in our second design style. In other words, achieving 'high performance' becomes exceedingly 'costly' when robustness to high delay variability is required. Since fault and defect tolerance have not yet been included in the 'cost/overhead' equation, the results of our study are definitely sobering in terms of the monumental challenges that lie ahead towards making designs targeted at nanotechnologies attractive/scalable.

The remaining of this paper is as follows. We start by providing some background on asynchronous design. Then, in Section III we discuss the two asynchronous design styles considered in our study. The experimental methodology adopted in the paper is presented in Section IV and an analysis of experimental results is given in Section V. Finally, some conclusions are given in Section VI.

II. BACKGROUND

In asynchronous design, processing blocks communicate data through asynchronous channels - Fig. 1 shows two such channels, namely, a *bundled-data* channel and a *1-of-2* channel. [17] [18]

When communication is performed through a bundled-data channel (see Fig. 1(a)), the sender (or producer) block asserts a request signal to tell the receiver (or consumer) block that a new batch of valid data (i.e., a new data 'token') is 'ready'. Subsequently, the consumer block asserts an acknowledge signal to tell the producer that the data has been received, and is thus no longer needed. The bundled data approach uses, thus, one data line per data bit (just like synchronous designs), plus a request line to signal that all data lines/bits are valid, and an acknowledge line to signal that the current data/token has been consumed. The request signal is usually generated using a 'matched' delay line - the line is dimensioned such that its delay is equal to that of the producer block plus

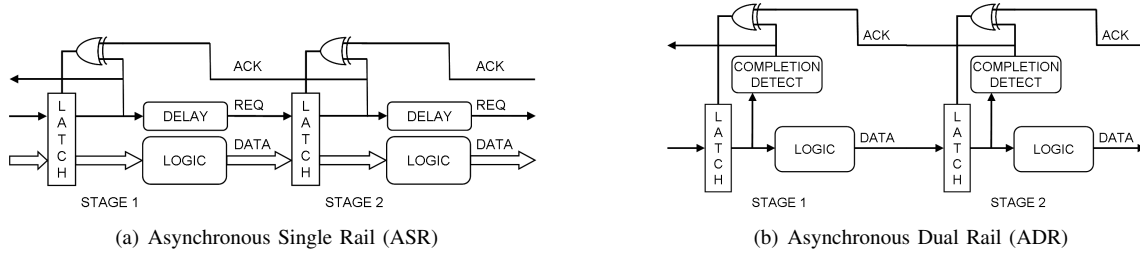


Fig. 2. Design styles exposing different area vs. performance trade-offs

some appropriate tolerance margin [19]. Such circuits thus incorporate a key timing assumption – the delay line must take longer than the worst case delay of the producer block, considering potential statistical variability in, both, the delay line and the producer block.

In contrast, when designs using 1-of-2 or dual rail encoding channels are considered (see Fig. 1(b)), the receiver block can detect the presence of a new token from the data itself, and thus no explicit request signal from the sender is needed. Note that 1-of-2 encoding uses 2 data lines per bit – one of these data lines represents logic 1 and the other line represents logic 0. So, when one of the two data lines are asserted, the receiving stage knows that the data is valid, and when both data lines are de-asserted, then it knows that no data is present.¹ Circuits based on dual rail encoding are sometimes called *speed independent*, since (differently from the bundled-data case) they make no timing assumptions with respect to the delay of the computation/functional blocks which produce data tokens (labeled at “logic” in Fig. 2). Note further that, when a token comprises more than 1-bit of data, a *completion detection* circuit is required at the receiving stage, in order to detect when all of the data token’s lines/bits are valid. As in the previous case, as soon as the receiving stage is finished with the data, it asserts the acknowledge signal. Then, proceeding with the 4-phase handshaking protocol typical of those designs, upon receiving the acknowledge signal, the sender resets the data lines (in order for the completion detection circuit to also reset, in preparation for the next data token), and then waits for the acknowledge signal to be de-asserted by the receiver, before sending a new token.

Thus, in simple terms, dual rail channel based asynchronous designs rely on a data encoding scheme to propagate a request along with the data, while bundled-data channel based designs rely on timing assumptions to ensure the validity of request signals – specifically, correct circuit operation requires such request signals to be asserted only after the corresponding data lines have become ‘valid’.

III. ASYNCHRONOUS DESIGN STYLES

In order to study the ‘overhead’ incurred by achieving robustness to delay variability, i.e., its impact on the performance and area of a design, we consider two asynchronous design styles – namely, one using bundled data channels, which we denote *Asynchronous Single Rail* (see Fig. 2(a)), and another

using 1-of-2 data channels, which we denote *Asynchronous Dual Rail* (see Fig. 2(b)). As will be seen, these two design styles expose different trade-offs between area and performance, being representative of key different strategies that can be pursued towards achieving such robustness, namely, via different degrees of reliance on timing assumptions vs. explicit handshaking. Since each of the design styles is capable of delivering maximum performance for a particular target ‘degree’ of uncertainty, both are relevant. Furthermore, as will be seen, the design style that performs by far the best under high delay uncertainty (*Asynchronous Dual Rail*) requires an extremely large area overhead (with respect to an ‘idealized’ synchronous design), revealing potential technology scaling limits due to delay uncertainty.

A. Asynchronous Single Rail (ASR)

Fig. 2(a) shows two pipeline stages of an asynchronous single rail (ASR) circuit. [19] As indicated above, ASR circuits use bundled data channels, with the request signal being generated by a delay line (see Section II). In an ASR circuit, the delay line is essentially used as a ‘local clock’ – that is, when the request signal reaches the end of the delay line, the latch at the beginning of the next stage knows that all the data is valid. Ideally, one wants to have the delay lines matching closely the delay of the logic blocks, so that the ASR pipeline operates as fast as possible. Unfortunately, when there is high statistical delay variability, the ASR delay lines need to be made correspondingly longer, so as to safely accommodate worst case logic delays.

1) *Pipeline*: Let us describe the operation of an ASR pipeline in some more detail – to illustrate the discussion, consider pipeline stages 1 and 2 in Fig. 2(a). A new processing cycle for stage 1 starts with the arrival of a request signal, indicating that all of the incoming data lines are valid. This causes the latch to change its state from transparent to closed, while concurrently a request signal is launched through stage 1’s delay line (as the new data token is also going through stage 1’s logic block). When the request signal reaches stage 2’s latch, the latter knows that all of its current input data is valid, and thus latches it. It then sends an acknowledge signal back to stage 1 (i.e., to its latch controller), and stage 1’s latch becomes again transparent. This concludes the processing cycle, since stage 1 is now ready to process a new data token (as soon as it receives a new request token). Our ASR circuits use simple D-latches, and the request line is comprised of a series of inverters.

¹Both lines asserted represents an error or invalid state.

2) *Cycle Time*: Based on the previous discussion, cycle time is given by:²

$$\text{cycle time} = 2L + Dl + X \quad (1)$$

where L denotes a latch delay, Dl denotes the actual delay of a stage's delay line, and X denotes an XOR gate delay (used to control the latches).

Additionally, the ASR circuit must verify that the propagation time through the delay line is slower than the worst case critical path delay of the logic elements. This timing constraint can be expressed as follows:

$$Dl > E \quad (2)$$

where E denotes the data 'evaluation' delay, that is, the delay of a pipeline stage's logic block.

3) *Logic Style*: Since ASR circuits rely on single rail data, they use logic elements/gates identical to those adopted in traditional synchronous design.

B. Asynchronous Dual Rail (ADR)

Fig. 2(b) shows two pipeline stages of an asynchronous dual rail (ADR) circuit. In contrast to the previous case, ADR circuits are insensitive to data evaluation delays (that is, to the delay of a pipeline stage's logic blocks), since they explicitly encode the completion of such an evaluation (and thus data validity) in the dual rail data itself. This has many potential advantages in terms of enhancing average performance, since such evaluation delays may vary substantially, not just due to actual wire and gate delay uncertainty, but also because the computation itself may exhibit substantial data driven variability – for example, the delay of ripple carry adder may vary significantly depending on the length of the longest carry propagation chain.

1) *Pipeline*: Since the request signal is encoded explicitly in the dual rail data, only the acknowledge signal has a designated wire (see Fig. 2(b)). As discussed in Section II, the stages in an ADR pipeline interact through a 4-phase handshaking protocol. Assertion of a data token from stage 1 is followed by a positive acknowledge edge from stage 2 (indicating that the new data token has been 'consumed'). After that, all data lines are de-asserted back to zero (representing a reset control token sent from stage 1). This is followed by a negative acknowledge edge from stage 2, indicating that it is now ready to receive the next data token, and thus concluding the cycle. In simple terms, each stage thus passes through two sequential 'phases': *evaluation* and *reset*. During evaluation, the stage is initially at the reset state (i.e., with all nodes at logical '0'), and then it evaluates a new data token. Reset happens after completion of the evaluation phase, in order to bring all nodes back to logical '0'.

2) *Cycle Time*: Cycle time can be determined as follows. Consider first the pipeline's forward propagation delay, i.e., the time it takes for a data token to move from one stage to next stage, in an empty pipeline. This time is given by the sum of an evaluation delay (over a stage's logic block) and a

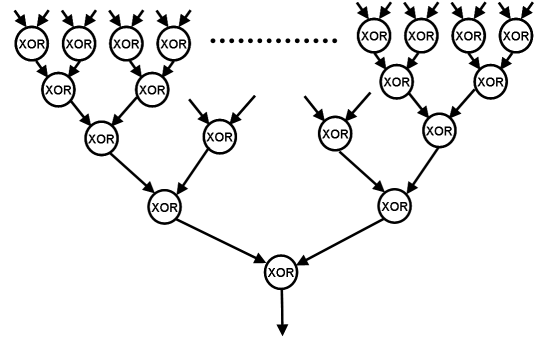


Fig. 3. A partial view of a 32-bit Parity Checker

latch delay, that is, $(E+L)$. Similarly, we may also consider the pipeline's forward propagation time for reset tokens, that is, the time taken to reset the entire logic block, yet typically such reset time is less than the evaluation time. For this reason, reset tokens will be typically stalled by previous data tokens in the next stage, and thus this contribution ends up being also $(E+L)$. The other key component of the cycle time is the backward propagation delay, that is, the time it takes for a stage to send back an acknowledge signal, in response to the receipt of an acknowledge from the next stage. This time consists of an XOR gate delay, a latch delay and a completion detection delay $(X+L+D)$. Yet, in this case we may have different delays for data and reset tokens, since completion detection times may be different for both.

Accordingly, cycle time is given by the sum of two forward propagation times and two backward propagation times:

$$\text{cycle time} = 4L + 2X + 2E + Dd + Dr \quad (3)$$

where L denotes average latch delay with respect to data and control input changes, X is an XOR gate delay, E denotes a data token evaluation delay, and Dd and Dr are detection delays for data and reset tokens respectively.

3) *Logic Style*: As mentioned above, for dual rail logic we have three logical states, each represented by two binary values. Namely, 00 corresponds to 'no data', 01 corresponds to logic 0, 10 corresponds to logic 1, and 11 is not used. Thus, dual rail logic must correctly calculate outputs considering this encoding. Note that sometimes the output may be calculated even if not all inputs are known. For example, the output of an AND gate is known if one of its inputs is logic 0, even if the other input is still unknown – since the validity of the output data is encoded in the data itself, dual rail logic can take advantage of such situations for performance enhancement.

C. ASR vs. ADR

In summary, in ASR designs, stages interact through an handshaking protocol, and thus there are no global timing constraints, nor common constraints between different pipeline stages. However, inside a stage we have a rigid control scheme, implemented via delay lines – namely, the latter must guarantee sufficient timing margins to account for the worst case statistical variability. So, when such variability is significant, the performance of ASR designs is obviously quite

²For simplicity, we assume interconnect delays to be negligible.

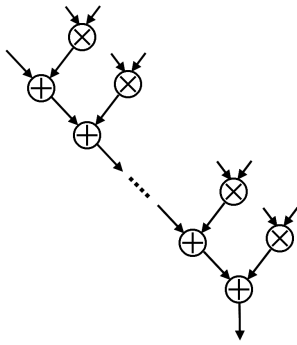


Fig. 4. A partial view of an 8-element DCT

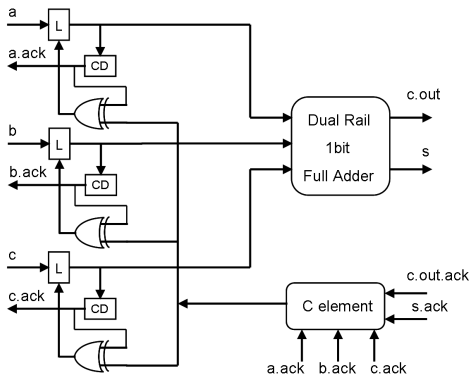


Fig. 5. Adder

affected. In the case of ADR designs, we further relax timing constraints even inside stages, at the cost of much heavier control logic. Specifically, dual rail encoding enables correct operation for any interstage communication delay and logic block delay (or function evaluation time), by using an explicit completion detection circuit directly applied to the data itself. Though ADR designs make no timing assumptions inside a stage in terms of data evaluation delay, it incorporates some timing assumptions in the control logic (mostly related to the operation and control of latches). Yet, if these circuits are designed properly, these timing assumptions may actually negatively impact performance only at very high levels of delay variability.

IV. EXPERIMENTAL METHODOLOGY

A. Benchmark Circuits

Two benchmark circuits were used to assess the performance vs. area trade-offs exposed by ASR vs. ADR circuits. Our first benchmark is a simple 32-bit parity checker. As shown in Fig. 3, the parity checker consists of a tree of XOR gates, thus providing no opportunities for the ADR circuits to explore data dependent performance variability, as discussed in Section 3.B(3). Our second benchmark is much larger – namely, it is an 8-element Discrete Cosine Transform (DCT) kernel. As shown in Fig. 4, each DCT stage is comprised of a 16bit by 12bit multiplier followed by an accumulate stage.

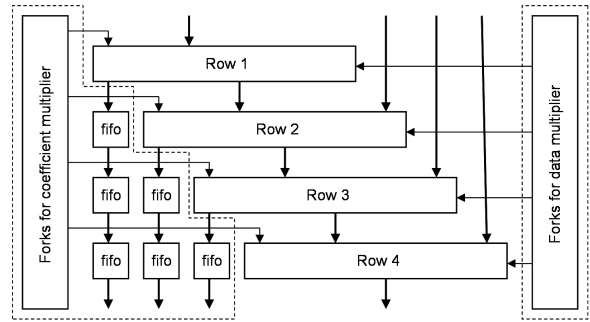


Fig. 6. Multiplier

B. Implementation

1) *Functional Components*: Both ASR and ADR circuit styles use the same XOR logic and topology for the parity checker. Also, both styles use an array multiplier for the DCT. The only key difference in this case is that the ASR circuit uses a parallel prefix adder while the ADR circuit uses a ripple carry adder. A parallel prefix adder was adopted for ASR circuit because it is substantially faster than the ripple carry adder.

Both the parity checker and the DCT circuits in the ADR style employ a deep single bit pipelining, implementing a single bit dataflow. Such deep pipelining leads to significant control logic overhead, yet this extra control logic enables also the free flow of bits, constrained only by real data dependencies. Specifically, the DCT circuit (Fig. 4) is designed using a two dimensional(2D) grid of single bit adders, or sequence of ripple carry adders, exposing plenty of opportunities for such performance enhancing "free flow" of bits.

The ADR DCT 2D grid of adders is built out of single bit adder stages – Fig. 5 shows one such single bit full adder, with 3 inputs and 2 outputs. This stage consists of a dual rail combinatorial block, to compute the full adder function, three latch blocks for each of the three dual rail inputs, and corresponding completion detection circuits, XOR control circuits for the latches, and one Muller C-element. The Muller C-element is used to coalesce the five acknowledge signals into a single signal, which is then used to control the latches [17].

2) *Special Considerations for ADR Circuits*: Fig. 6 shows the grid multiplier used for DCT circuit. Each row represents a ripple carry adder. The dashed lines represent added elements in the ADR circuit that are used to enhance performance, as discussed below.

During multiplication, each input bit has a very high fanout within the grid multiplier, ie. each input bit needs to be replicated to several stages. In the ADR circuit, each of these destinations must send an acknowledge signal back to the input bit – this acknowledgment is done through the input fork stage shown in Fig. 6. The simplest possible fork stage is a buffer stage with one input and several outputs. This fork stage waits for acknowledges from all receivers. Using this simple fork stage approach introduces additional unwanted synchronization between all receivers, which may significantly increase the cycle time of the circuit.

One way to reduce such penalties associated with the fork

stage is to use multiple fork stages built in the form of a tree with FIFOs suitably placed at the various tree levels. The impact of such organizations in circuit performance will be shown in Section V.

C. Simulation Methodology

In this work, we consider only static delay variations. The performance of our various circuits was determined using Monte Carlo simulation. For each run and for each gate in the critical path, a Gaussian random delay number is assigned. The nominal value for each gate was determined using Synopsys HSPICE simulations with the Berkeley BTPM 65nm transistor model. The 1-sigma standard deviation was varied from 0 to 40% of the nominal gate value, to simulate target nanotechnologies with different degrees of parameter variability, with 50% of the total variation considered to be global, and 50% considered to be local. Global and local variations are considered to be not correlated, so the sum of the variance for local and global parts is equal to total variance. In order not to be overly optimistic, we ensure that the delay of each gate would never be less than 50% of its nominal value. According, if a random number was less than 50% of the gate nominal value, a new random number was chosen. The Gaussian distribution parameters were adjusted such that the desired mean and standard deviation would remain the same after this adjustment.

A few additional details on the results generated from each run. For the ASR circuits, 100,000 chips were simulated using the static critical path as the overall delay. From these 100,000 chips, the fastest 10 chips and the slowest 10 chips were discarded. The slowest chip of the 99,980 remaining chips was selected. Using the same methodology, 100,000 chips with only inverter chains in the request line were simulated to determine the delay for various lengths of inverter chains. Using the fastest chip in a particular lot of chips with the same number of inverters, we then determined the number of inverter that would guarantee the timing constraint for the delay line. Lastly, the lot of inverter chains was analyzed to determine what the mean delay of the lot and 95% delay of the lot would be. These final mean and 95% delays are show in the figures in Section V.

For the ADR circuits, the Synopsys VCS verilog compiler and simulator was used to generate the cycle delays. The cycle time is computed from one rising edge of an acknowledge signal to the next rising edge of an acknowledge signal. The average cycle time is computed using a large number cycles.

V. ANALYSIS OF EXPERIMENTAL RESULTS

In this section, we present and discuss our experimental results – namely, for each benchmark and sigma value (that is, range of delay variability exhibited by some hypothetical target nanotechnology), we identify which design style achieves the best throughput, i.e., which style handles such uncertainty in a more performance efficient manner, and we then do a rough assessment on the relative cost of such efficiency, i.e., on incurred area overhead in terms of number of transistors.

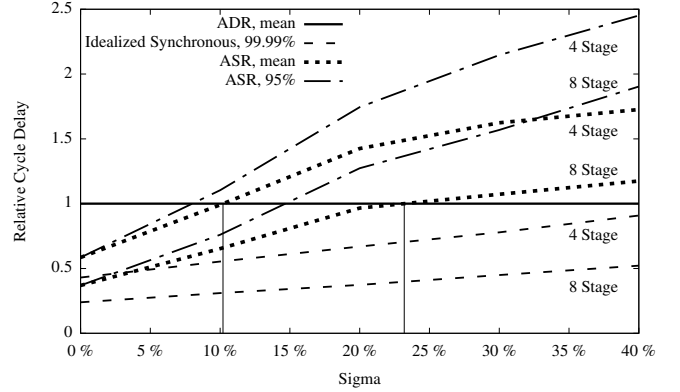
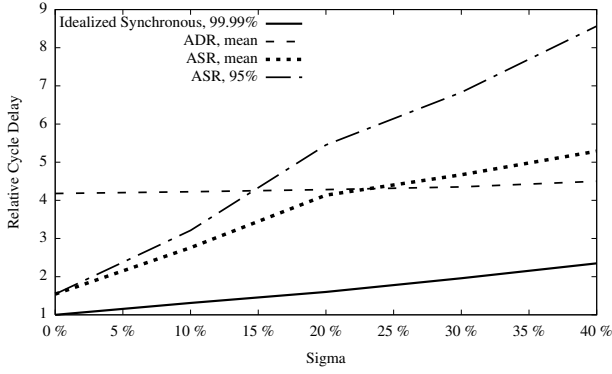


Fig. 7. 4 and 8 stage DCT Pipeline: Cycle Time Normalized to ADR

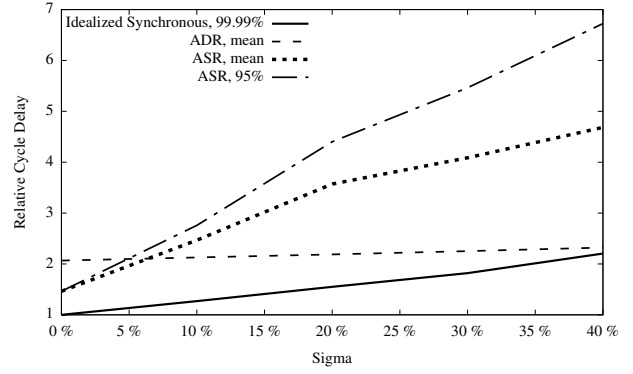
A. Performance Analysis

As discussed in Section II, the so called 'speed independence' of ADR designs (that is, the absence in such designs of timing assumptions with respect to the delay of computation/functional blocks, as well as interstage communication) has a major impact on the style's basic cycle delay – this can be easily seen, by comparing the cycle delay for ADR designs, see equation (3), to the cycle delay for ASR designs (which do incorporate such timing assumptions), see equation (1). Since performance/throughput was the main optimization criteria in our study, it was critical to design the two benchmark ADR circuits for 'maximum' performance (see discussion on single bit dataflow and frontend fork stage in Section IV.B), so as to identify when would the delay variability 'smoothing' capabilities (inherent to their speed independent mode of operation) overcome the baseline or nominal cycle time benefits of the much 'lighter' ASR design style (that is, the superior performance of the latter, when no delay uncertainty is observed). Note that, as sigma increases, the ASR design changes, i.e., the delay line is obvious dimensioned to take correspondingly longer, while the ADR (or speed independent) design remains exactly the same. Our experiments will thus allow us to identify the range of delay variability for which the worst-case based dimensioning of the ASR delay line will lead to a system throughput actually worst than that achieved by the speed independence ADR style.

Note further that, as mentioned in Section IV.B-2, the ASR cycle time can be varied arbitrarily, by varying the logic depth of the design's pipeline stages. In what follows, we use Fig. 7 to support the discussion on the relative implications of such choice, using the DCT benchmark. Specifically, Fig. 7 shows the cycle times obtained for two ASR implementations, one with eight pipeline stages and another with four stages, both normalized to the cycle time obtained for the ADR design, for each sigma value. Specifically, we report: (1) the mean and 95% normalized cycle delay for both ASR designs; and (2) the 99.99% normalized delay for the pipeline stages used on each one of the two ASR designs (which we denote 'idealized synchronous design'). Note that the values reported in (2) represent the actual delays taken by the computation itself, with no overheads due to the tolerance margins required by the ASR delay lines. In other words, by comparing the values



(a) DCT: Cycle Time Normalized to Nominal Idealized Synchronous



(b) Parity Checker: Cycle Time Normalized to Nominal Idealized Synchronous

Fig. 8. Performance Results

TABLE I
TRANSISTOR COUNT COMPARISON

(a) Parity Checker						(b) DCT					
	ADR		ASR		ADR/ASR overhead		ADR		ASR		ADR/ASR overhead
	# Trans. ($\times 10^3$)	%	# Trans. ($\times 10^3$)	%			# Trans. ($\times 10^3$)	%	# Trans. ($\times 10^3$)	%	
functional	0.7	10.6	0.5	70.5	1.5	190.1	7.9	109.4	87.7	1.7	
control	4.3	61.1	0.2	26.7	22.8	704.7	29.5	10.2	8.2	68.8	
latches	2.0	28.3	0.02	2.8	99.2	149.8	62.6	5.1	4.1	292.6	
total	7.0	100	0.7	100	10.0	2392.8	100	124.7	100	19.2	

TABLE II
DCT: TRANSISTOR BREAKDOWN

	Logic Core		Front End Stages					
	# Trans. ($\times 10^3$)	% total	Fork tree with Fifo		Fork tree without Fifo		Basic Fork	
			# Trans. ($\times 10^3$)	% Core	# Trans. ($\times 10^3$)	% Core	# Trans. ($\times 10^3$)	% Core
function	190	16.7	0	0	0	0	0	0
control	338	29.7	366	32.1	121	10.6	27	2.4
latches	611	53.6	886	77.8	174	15.3	14	1.2
total	1139	100	1252	109.9	295	26.9	41	3.6
cycle time	(770ps)		886ps		1364ps		1872ps	

in (2) to the values in (1), one can see the slow down incurred by the delay line control scheme, with respect to the maximum possible speed at which the computation could ideally proceed, for each sigma value. As it can be seen, that performance gap (or loss) increases substantially with sigma, that is, with increases in delay uncertainty, due to the need for worst case based dimensioning of the delay line.

The straight line shown in Fig. 7 (with value 1 for all sigmas) represents the ADR mean cycle time used for normalization. Perhaps the most critical information conveyed in this figure is the sigma values at which the ADR line crosses the two curves representing the mean cycle times for the 4 and 8 pipeline stage ASR designs - highlighted with two vertical lines. These two points denote the degree of variability beyond which the 'speed independence' of the ADR design starts to pay-off with respect to the corresponding ASR design, in terms of actual system performance.

Specifically, our results show that the performance of the ASR design with 8 pipeline stages is superior to that of the

maximum performance ADR design up to a sigma of approximately 24%, which represents quite a substantial degree of uncertainty. However, as one would expect, when a much slower ASR design with only 4 pipeline stages is considered, its cycle time is superior to that of the maximum performance ADR design only for sigma values up to approximately 10%. In short, in order to improve the performance/throughput of the preferred/lighter ASR designs, one should try to pipeline them as much as possible - exactly what is typically done also in contemporaneous synchronous design.

Fig. 8 shows relative cycle delays for our two benchmarks, i.e., the Parity Checker and DCT, but now normalized to the what we denote "nominal idealized synchronous delay", that is, the actual delay required by the computation itself, with no safety time margins of any type, and assuming sigma equal to zero. Note first that both ADR designs show almost no cycle time increases with sigma (typically under 10%) - this illustrates the power of 'speed independence' at

work. In contrast, the performance of both ASR designs is extremely sensitive to σ (i.e., to the assumed degree of delay variability), as expected. Note finally, the ASR DCT design reported in Fig. 8 has 8 pipeline stages (and a critical path of 3 INV + 3 NAND3 + 2 NAND4, with a nominal delay of 170ps), while the ASR Parity Checker design is comprised of a single stage (and a critical path of 5 INV + 10 NAND2, with a nominal delay of 257ps). Thus, the crossing points with respect to the corresponding ADR designs reflect both such baseline throughputs. Naturally, if we had pipelined more the ASR designs, the crossing points would move to the right, and vice-versa. Perhaps more informative than the actual crossing points is the actual slopes of both curves.

B. Area Analysis

Tables I(a) and Table I(b) show the number of transistors used by the Parity Checker and the DCT designs, partitioned by functional logic, control logic, and storage/latches. The ASR DCT design considered here has again 8 pipeline stages. The numbers shown on the third row of each of tables speak for themselves – the ADR Parity Checker design uses ten times more transistors than the corresponding ASR design. For the DCT design, the numbers are even worse, in that the ADR design uses 20 times more transistors than its ASR counterpart.

Table II shows the rationale of such an increase – namely, the DCT ADR design requires a huge frontend stage aimed at feeding the various input bits of the logic core at a speed as fast as the core can sustain (see details in Section IV.B). Namely, as it can be seen in the last line of the table, with a perfect front-end, the DCT core delivers an average cycle time of 770ps. Yet, to sustain a throughput close to that one, the costly frontend shown in the leftmost column of Table II is needed.

Note finally that, as expected, the percentage of transistors spent on actual computation, i.e., on functional logic, (as opposed to control and storage), is in the order of only 10% for the ADR designs, that is, the designs that perform the best under high uncertainty conditions, as opposed to 70% or more for the ASR designs. Thus, a huge area overhead (i.e., extra control and storage) is indeed required to effectively smooth out performance variability under such conditions.

VI. CONCLUSION

We have considered two representative asynchronous design styles, ASR and ADR, and assess their relative merits for representative benchmarks. Our goal was to study how the increased densities afforded by nanotechnologies will be countered by design overheads associated with achieving robustness to performance variability. The experimental results of our study clearly indicate that very substantial overheads will be incurred in delivering high performance under substantial delay uncertainty, possibly suggesting that this may indeed be one of the fundamental scaling limits of nanotechnologies. We are currently incorporating power considerations in our study.

REFERENCES

- [1] T. Rueckes, K. Kim, E. Jose-Levich, G. Tseng, C.-L. Cheung, and C. Lieber, "Carbon nanotube based non-volatile random access memory for molecular computing," *Science*, vol. 289, pp. 94–97, 2000.
- [2] T. I. Kamins and R. S. Williams, "Trends in nanotechnology: Self-assembly and defect tolerance," in *Proc. NSF Partnership in Nanotechnology Conf.*, Jan. 2001.
- [3] G. Bourianoff, "The future of nanocomputing," *Computer Magazine*, pp. 44–49, Aug. 2003.
- [4] A. DeHon, "Array-based architecture for FET-based nanoscale electronics," *IEEE Trans. Nanotechnology*, vol. 2, no. 1, pp. 23–32, 2003.
- [5] J. R. H. et. al., "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716–1721, Jun. 1998.
- [6] S. C. Goldstein and M. Badiu, "Nanofabrics: Spatial computing using molecular electronics," in *Proc. International Symposium on Computer Architecture (ISCA)*, Jul. 2001, pp. 178–191.
- [7] N. C. et al., "Soft error considerations for deep-submicron CMOS circuit applications," in *IEEE International Electron Devices Meeting: Technical Digest*, 1999, pp. 315–318.
- [8] R. W. Keyes, "Fundamental limits of silicon technology," *Proc. IEEE*, vol. 89, no. 3, pp. 227–239, Mar. 2001.
- [9] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to binary logic switch scaling - a Gedanken model," *Proc. IEEE*, vol. 91, no. 11, pp. 1934–1939, November 2003.
- [10] D. S. Boning and J. Chung, "Statistical metrology – measurement and modeling of variation for advanced process development and design rule generation," in *Proc. Intl. Conference on Characterization and Metrology for ULSI Technology*, 1998.
- [11] M. A. Breuer, S. K. Gupta, and T. M. Mak, "Defect and error tolerance in the presence of massive numbers of defects," *IEEE Des. Test. Comput.*, vol. 21, no. 3, pp. 216–227, May-June 2004.
- [12] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies," in *Proc. Design Automation Conference (DAC'04)*, 2004.
- [13] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proc. Intl. Workshop on System-level Interconnect Prediction*, 2004, pp. 7–13.
- [14] A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2002.
- [15] A. Martin, M. Nystrom, and C. Wong, "Three generations of asynchronous microprocessors," *IEEE Design & Test of Computers, special issue on Clockless VLSI Design*, November/December 2003.
- [16] G. K. et al., "Implementation of a third-generation 1.1-ghz 64-bit microprocessor," *IEEE Journal of Solid-State Circuits and Systems*, vol. 17, no. 11, pp. 1461–1469, Nov. 2002.
- [17] A. Davis and S. M. Nowick, "An introduction to asynchronous circuit design," University of Utah, Salt Lake City, Utah, Tech. Rep. UUCS-97-013, Sept. 1997.
- [18] P. A. Beerel, "Asynchronous circuits: an increasingly practical design solution," in *Proc. International Symposium on Quality Electronic Design (OSQED'02)*, Mar. 2002, pp. 367–372.
- [19] M. Singh and S. Nowick, "Mousetrap: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. IEEE International Conference on Computer Design (ICCD'01)*, Austin, Texas, Sept. 2001, pp. 9–17.