Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications

Jordi Cortadella, *Member, IEEE*, Alex Kondratyev, *Senior Member, IEEE*,
Luciano Lavagno, *Member, IEEE*, and Christos P. Sotiriou, *Member, IEEE*

***Abstract*—Asynchronous implementation techniques, which measure logic delays at runtime and activate registers accordingly, are inherently more robust than their synchronous counterparts, which estimate worst case delays at design time and constrain the clock cycle accordingly. Desynchronization is a new paradigm to automate the design of asynchronous circuits from synchronous specifications, thus, permitting widespread adoption of asynchronicity without requiring special design skills or tools. In this paper, different protocols for desynchronization are first studied, and their correctness is formally proven using techniques originally developed for distributed deployment of synchronous language specifications. A taxonomy of existing protocols for asynchronous latch controllers, covering, in particular, the four-phase handshake protocols devised in the literature for micropipelines, is also provided. A new controller that exhibits provably maximal concurrency is then proposed, and the performance of desynchronized circuits is analyzed with respect to the original synchronous optimized implementation. Finally, this paper proves the feasibility and effectiveness of the proposed approach by showing its application to a set of real designs, including a complete implementation of the DLX microprocessor architecture.**

***Index Terms*—Asynchronous circuits, concurrent systems, desynchronization, electronic design automation, handshake protocols, synthesis.**

I. Introduction

FEEDBACK closed-loop control is a classical engineering technique used to improve the performance of a design in the presence of manufacturing uncertainty. In traditional digital design, synchronization control is performed in an open-loop fashion. That is, all synchronization mechanisms, including clock distribution, clock gating, and so on, are based on a feedforward network, from the oscillator to one or more phase-locked loops to a clock buffering tree and routing network. All delay uncertainties in both the clock tree and the combinational logic must be designed out, i.e., taken care of by means of appropriate worst case margins.

Manuscript received June 15, 2005. This work was supported by the Working Group on Asynchronous Circuit Design (ACID-WG, IST-1999-29119), the ASPIDA under Project (IST-2002-37796), and CICYT under TIN2004-07925. This paper was recommended by Associate Editor R. Camposano.

J. Cortadella is with the Software Department, Universitat Politécnica de Catalunya, Barcelona 08034, Spain (e-mail: jordi.cortadella@upc.edu).

A. Kondratyev is with Cadence Berkeley Laboratories, Berkeley, CA 94704 USA (e-mail: kalex@cadence.com).

L. Lavagno is with Politecnico di Torino, Torino 10132, Italy (e-mail: lavagno@polito.it).

C. P. Sotiriou is with ICS-FORTH, Crete 711 10, Greece (e-mail: sotiriou@ics.forth.gr).

Digital Object Identifier 10.1109/TCAD.2005.860958

This approach has worked very well in the past, but, currently, it shows several signs of weakness. A designer, helped by classical electronic design automation (EDA) tools, must estimate at every design stage (floor planning, logic synthesis, placement, routing, mask preparation) the effect that uncertainties about the following design and fabrication steps will have on geometry, performance, and power (or energy) of the circuit. In the case of delay and power, these uncertainties add up to huge margins that must be taken in order to ensure that a sufficiently large number of manufactured chips work correctly, i.e., within specifications. Statistical static timing analysis (SSTA, see, e.g., [1] and [26]) partially deals with the problem, by separating uncorrelated variations, whose effect is reduced because they quickly average out, and correlated variations, which must still be taken care of by margins.

This paper focuses on reducing the effect of correlated variability sources such as supply voltage, operating temperature, and large-scale process variations (e.g., optical imperfections). Such sources of power and performance variation cannot be taken into account purely by SSTA.

In addition to variability effects induced by process and operating conditions, people now use circuit-level power minimization and equalization techniques, such as dynamic voltage scaling and adaptive body biasing, that have very significant effects in terms of performance. Unfortunately, operating very close to the transistor threshold voltage increases the significance of nonlinearities and second-order effects, thus, making the *a priori* prediction of delays across a broad range of operating voltages very problematic.

Changing the clock frequency in order to match performance with scaled supply voltage is already quite difficult, since it multiplies the complexity of timing analysis by the number of voltage steps, and variability impact at low voltages is much more significant. Performing frequency scaling in the presence of adaptive body biasing, and hence, variable threshold voltage, is even more complex. Moreover, clocks generated by phase-locked loops cannot be used during frequency change transients.

The techniques described in this paper make voltage/frequency-based power optimization and control much easier, since they are inherently more tolerant of delay variations.

Fortunately, several kinds of applications, and, in particular, those using complex processor architectures for part of the computation (e.g., general purpose computing and multimedia), and several others that are tolerant to environmental variations (e.g., wireless communications) do not have to obey strict timing constraints at all times. Due to the widespread use of caches,

irregular processing speeds, and multitasking kernels, all these application areas inherently require algorithms that are tolerant to internal performance variations and offer only average case guarantees. For example, a digital camera takes about 1 s to process four or five million pixels. In all these cases, a design style in which the device provides average case guarantee, but may occasionally go slower (e.g., when used in the desert) or faster (e.g., when used on the snow) is quite acceptable. If the performance of that device, on average, is double that of a traditionally designed one, then there is a significant motivation to use the robust techniques described in this paper.

It is widely reported that, as technology progresses, the distance between the "official performance" and the "actual performance" of a chip is continuously broadening, and 100% margins (meaning that an integrated circuit can work twice as fast as it is officially rated) are not uncommon even today. This motivates us to look into the issue of measuring circuit delay at runtime, after fabrication, rather than estimating it during design, before fabrication. Unfortunately, this requires us to consider asynchronous design techniques, since they are inherently closed loop, and hence, more robust in the presence of variation, as discussed above. This is enough to make most designers nervous, since asynchronous design has traditionally been considered dangerous. We believe that there are two major reasons for this fact.

1) There are no good computer-aided design (CAD) tools that completely cover the design flow.
2) Asynchrony involves changing most of the designers' mentality when devising the synchronization among different components in a system.

This paper is a first step in the direction of automatically introducing, based only on standard EDA tools and flows, asynchronous feedback control of latches and flip-flops in a digital design.

We propose a methodology that deviates from normal application-specified integrated circuit (ASIC) design only when it deals with the clock tree at the logical level. That is, specification using synthesizable hardware description languages (HDLs), logic synthesis, layout, verification, extraction, automated test pattern generation, and so on all remain the same.

This is only a first step, because we only consider the synchronization level and not the actual measurement of logic and wire delays. In this paper, delays are bounded by using matched delay lines, which must be longer than the longest path in the combinational logic. Ongoing research devoted to automated conversion of a datapath to dual rail, in order to measure actual delays, is discussed in [12].

A. Desynchronization

Desynchronization incorporates asynchrony in a conventional EDA flow, without changing the "synchronous mentality" or requiring new tools. Both aspects are quite advantageous from several standpoints. First, the notion of operation cycle lives in the subconscious of most circuit designers. Finite state machines, pipelined microprocessors, multicycle arithmetic operations, etc., are typically studied with the underlying idea of operation cycle, which is inherently assumed to be defined by a clock. As an example, one can think about the traditional lecture on computer architecture explaining the DLX pipeline. One immediately imagines the students looking at the classical timing diagram showing the overlapped IF–ID–EX–MEM–WB stages, synchronized at the level of a cycle. It would be very difficult to persuade the lecturer to explain the same ideas without that notion. Secondly, most EDA tools, from logic synthesis to verification, assume a cycle-based paradigm for computation (between clock edges) and memorization (at clock edges), which is very useful to separate functionality (Boolean logic) from performance (timing of longest and shortest paths).

Operation cycles are useful for reasoning and designing. On the other hand, an underlying asynchronous implementation is extremely valuable for the reasons described above. Both these apparently conflicting requirements can be reconciled by using the concept of desynchronization. The essential idea is to start from a synchronous synthesized (or manually designed) circuit and replace directly the global clock network with a set of local handshaking circuits. The circuit is then implemented with standard tools, using the flows originally developed for synchronous circuits. The only modification is the clock tree generation algorithm. With this approach, we provide a design methodology that can be picked up almost instantaneously and without risk by an experienced team.

B. Contributions

This work gets its inspiration from a number of contributions from past work, each providing a key element to a unique novel methodology. Many of the concepts that appear in this paper have been around for a long time, such as handshake protocols, asynchronous pipelines, local controllers, etc.

The essential novelty of our contribution is that it provides a fully automated synthesis flow based on a sound theory that guarantees correctness, does not require any knowledge of asynchronous design by the designer, and does not change at all the structure of synchronous datapath and controller implementation, but only affects the synchronization network.

In particular, our design flow starts from a standard synthesizable HDL specification or gate-level netlist, yet, it provides several key advantages of asynchronicity, such as low electromagnetic interference (EMI), global idling, and modularity.

To show that the suggested methodology is sound, we provide formal proofs of correctness based on the theory of Petri nets. We study different handshake protocols for latch controllers and present a taxonomy determined by the degree of concurrency of each protocol. A controller that preserves the maximum concurrency for desynchronization is also presented.

We validated our approach by comparing synchronous and desynchronized designs of large examples, including an implementation of the data encryption standard (DES) and one of the DLX microprocessor [19], since we did not want to rely on small artificial logic synthesis benchmarks. Both design styles were implemented using the same set of commercial EDA tools for synthesis, placement, and routing. To the best of our knowledge, this is the first time an asynchronous

design obtained through a conventional EDA flow does not show any penalty (in terms of area, power, and performance) with respect to its synchronous counterpart. Initial measurements from a fabricated version of the DLX, with both synchronous and desynchronized clock trees, further confirms simulation results.

II. Previous Work

Sutherland, in his Turing award lecture, proposed a scheme to generate local clocks for a synchronous latch-based datapath. His theory for asynchronous designs has been exploited successfully by both manual designs [16] and CAD tools [2], [5], [7]. That methodology is very efficient for dataflow type of applications but is less suitable to emulate the behavior of synchronous system by firing of local clocks in a sort of "asynchronous simultaneity."

In a different research area, Linder and Harden started from a synchronous synthesized circuit and replaced each logic gate with a small sequential handshaking asynchronous circuit, where each signal was encoded together with synchronization information using a level-encoded dual-rail (LEDR) delay-insensitive code [24]. That approach bears many similarities with ours, in particular, because it generates an asynchronous circuit from a synchronous specification, but in our opinion, it attempts to go too far, because it transforms each combinational gate into a sequential block, which must locally keep track of the odd/even phases. Thus, it may have an excessive overhead, even when used for large-granularity gates such as in FPGAs. To alleviate this overhead, a coarse-grain approach was used in [29], but no direct apples-to-apples comparison with a synchronous design was presented there.

Similarly, Theseus Logic proposed a design flow [23] that uses traditional combinational logic synthesis to optimize the datapath and uses direct translation and special registers to generate automatically a delay-insensitive circuit from a synchronous specification. That approach also has a high overhead and requires designers to use a nonstandard HDL specification style, different from the synchronous synthesizable subset.

Kessels *et al.* also suggested generating the local clocks of synchronous datapath blocks using handshake circuits [20], but used Tangram as a specification language. This has some advantages, in that synchronous block activation can be controlled at a fine granularity level as in clock gating, but does not use a standard synchronous register transfer level (RTL) specification.

The generation of local clocks from the handshaking circuitry while ensuring the global "synchronicity" was first suggested in [30]. That was the first work suggesting a conversion of synchronous circuits into asynchronous ones through replacement of flip-flops by master–slave latches with corresponding controllers for local clocking. Similar ideas were exploited in a doubly latched asynchronous pipeline suggested in [21]. Our paper extends the results from [21] and [30] by using more general synchronization schemes and provides a theoretical foundation for the desynchronization approach by proving a behavioral and temporal equivalence between a synchronous circuit and its desynchronized counterpart.

We also extend with respect to our own previous work in [10] and [11], because we use a maximally concurrent synchronization mechanism, show how previously published handshake controllers can be derived from this maximally concurrent model by concurrency reduction, and, finally, prove its equivalence to the synchronous version.

A related research area, albeit in a totally different application domain, is desynchronization of synchronous language specifications for deployment on distributed loosely synchronized platforms. In that case, the problem arises from the need to use synchronous languages [18] for embedded software modeling. These languages offer the same zero-delay abstraction as combinational logic and, thus, ensure easy specification of composable deterministic reactive software modules. However, compilation techniques into machine code for these languages traditionally assume implementation on a single processor, while application areas (e.g., automotive and aerospace) generally assume distributed implementation onto loosely coupled control units, which do not share a dependable common clock.

Benveniste *et al.* [3], [4] devised conditions under which a synchronous specification can be deployed on an architecture that does not ensure in-order reception of events on different physical signals, which is very similar to the assumption made in asynchronous hardware design. We use some of their definitions in order to formally prove the equivalence between our desynchronized circuits and the original synchronous specification. Note, however, that Benveniste's original results require the synchronous modules to satisfy a pair of conditions that are not true in general of any synchronous design.

1) Endochrony requires every module distributed asynchronously (in our case, every group of logically related registers) to have a way to tell when its inputs are ready and which ones are irrelevant in a given operation cycle, based only on their values.
2) Isochrony requires two modules who share a signal to agree always on its value (i.e., they cannot assign concurrently conflicting values to it).

In our case, we simplify such conditions, so that they are met by every possible input synchronous circuit. In particular, we assume that all inputs to a combinational block are required to compute its output. While conservative, our condition is easier to satisfy than Benveniste's ones. It potentially loses performance and power, with respect to a solution implemented using Benveniste's approach, because it neglects to consider "sequential don't cares" when determining synchronization conditions. However, it is automatable, and, hence, we chose it for our work.

III. Marked Graphs

A marked graph (MG) is the formalism used in this paper to model desynchronization. They are a subclass of Petri nets [25] that can model decision-free concurrent systems.

Definition 3.1 (MG): An MG is a triple $(\Sigma, \rightarrow, M_0)$, where Σ is a set of events, $\rightarrow \subseteq (\Sigma \times \Sigma)$ is the set of arcs (precedence relation) between events, and $M_0 : \rightarrow \longrightarrow \mathbb{N}$ is an initial marking that assigns a number of tokens to the arcs of the MG.

An event is enabled when all its direct predecessor arcs have a token. An enabled event can occur (fire), thus, removing one token from each predecessor arc and adding one token to each successor arc. A sequence of events σ is feasible if it can fire from M_0, denoted by $M_0 \xrightarrow{\sigma}$. A marking M' is reachable from M if there exists a σ such that $M \xrightarrow{\sigma} M'$. The set of reachable markings from M_0 is denoted by $[M_0\rangle$.

An example of an MG is shown in Fig. 3(b), where the events $A+$ and $A-$ represent the rising and falling transitions of signal A, respectively. In the initial marking (denoted by solid dots at arcs), two events are enabled, i.e., $B+$ and $D+$. The sequence of events $\langle D+\ D-\ C+\ B+\ B-\ A+\ C-\rangle$ is an example of a feasible sequence of the MG.

Definition 3.2 (Liveness): An MG is live if for any $M \in [M_0\rangle$ and for any event $e \in \Sigma$, there is a sequence fireable from M that enables e.

Liveness ensures that any event can be fired infinitely often from any reachable marking.

Definition 3.3 (Safeness): An MG is safe if no reachable marking from M_0 can assign more than one token to any arc.

Definition 3.4 (Event Count in a Sequence): Given a firing sequence σ and an event $e \in \Sigma$, $\overline{\sigma}(e)$ denotes the number of times that event e fires in σ.

The following results were proven in [9] for strongly connected MGs.

Theorem 3.1 (Liveness): An MG is live if and only if M_0 assigns at least one token on each directed circuit.

Theorem 3.2 (Invariance of Tokens in Circuits): The token count in a directed circuit is invariant under any firing, i.e., $M(C) = M_0(C)$ for each directed circuit C and for any M in $[M_0\rangle$, where $M(C)$ denotes the total number of tokens on C.

Theorem 3.3 (Safeness): An MG is safe if and only if every arc belongs to a directed circuit C with $M_0(C) = 1$.

In the rest of the paper, we only deal with strongly connected MGs.

IV. Zero-Delay Desynchronization Model

The desynchronization model presented in this section aims at the substitution of the global clock by a set of asynchronous controllers that guarantee an equivalent behavior. The model assumes that the circuit has combinational blocks (CL) and registers implemented with D flip-flops (FF), all of them working with the same clock edge [e.g., rising in Fig. 1(a)].

A. Steps in Desynchronization Method

The desynchronization method proceeds in three steps.

1) Conversion of the flip-flop-based synchronous circuit into a latch-based one [M and S latches in Fig. 1(b)]. D flip-flops are conceptually composed of master–slave latches. To perform desynchronization, this internal structure is explicitly revealed [see Fig. 1(b)] to:
   a) decouplelocal clocks for master and slave latches (in a D flip-flop, they are both derived from the same clock);
   b) optionally improve performance through retiming, i.e., by moving latches across combinational logic.

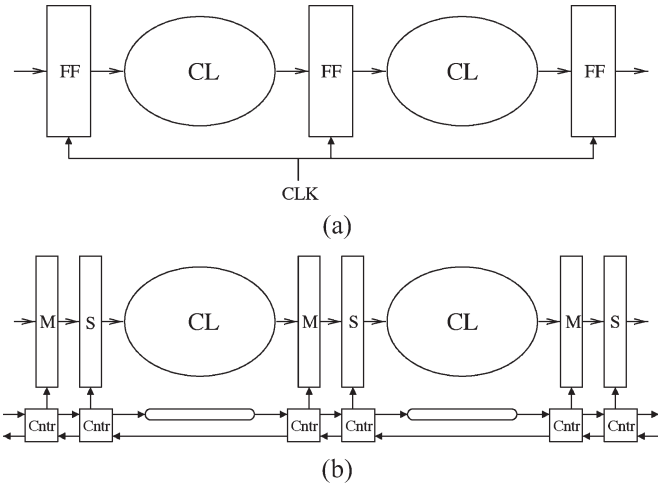


Fig. 1. (a) Synchronous circuit. (b) Desynchronized circuit.

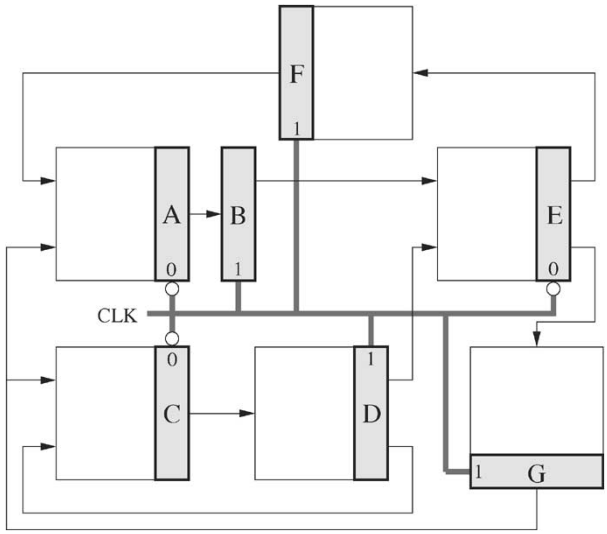


Fig. 2. Synchronous circuit with single global clock.

   The conversion of a flip-flop-based circuit into a latch-based one is not specific to the desynchronization framework only. It is known to give an improvement in performance for synchronous systems [8] and, for this reason, it has a value by itself.
2) Generation of matched delays for the combinational logic [denoted by rounded rectangles in Fig. 1(b)]. Each matched delay must be greater than or equal to the delay of the critical path of the corresponding combinational block. Each matched delay serves as a completion detector for the corresponding combinational block.
3) Implementation of the local controllers. This is the main topic of this section.

Fig. 2 depicts a synchronous netlist after the conversion into latch-based design, possibly after applying retiming. The shadowed boxes represent latches, whereas the white boxes represent combinational logic. Latches must alternate their phases. Those with a label 0 (1) at the clock input represent the even (odd) latches. All latches are transparent when the control signal is high (CLK $= 0$ for even and CLK $= 1$ for odd). Data transfers must always occur from even (master) to odd (slave)

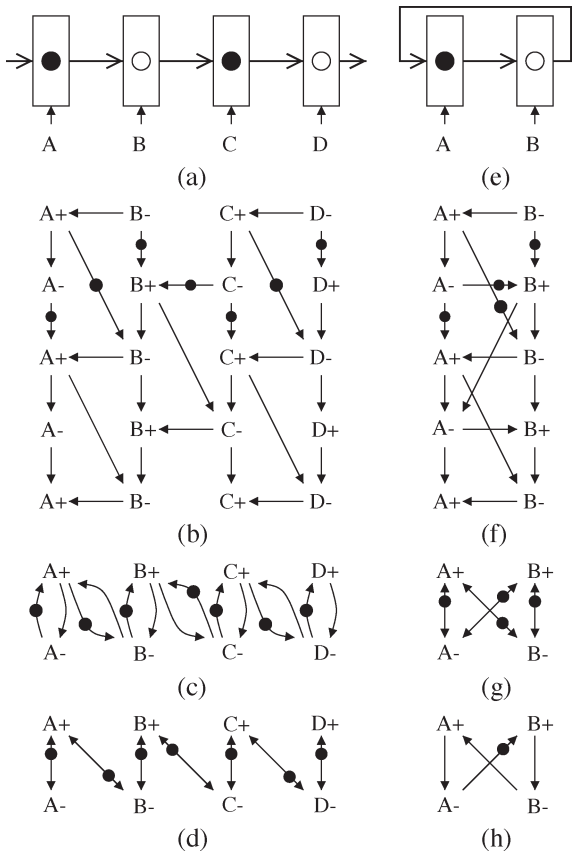


Fig. 3. Desynchronization model for linear pipeline and ring.

latches and vice versa. Usually, this latch-based scheme is implemented with two nonoverlapping phases generated from the same clock.

Initially, only the latches corresponding to one of the phases store valid data. Without loss of generality, we assume that these are the even latches. The odd latches store bubbles, in the argot of asynchronous circuits.

B. Zero-Delay Model

This section presents a formal model for desynchronization. The aim is to produce a set of distributed controllers that communicate locally with their neighbors and generate the control signals for the latches in such a way that the behavior of the system is preserved. For simplicity, we assume that all combinational blocks and latches have zero delay. Thus, the only important thing about the model is the sequence of events of the latch control signals. The impact of the datapath delays on the model will be discussed during the implementation of the model (Section VI).

For simplicity, we start by analyzing the behavior of a linear pipeline [see Fig. 3(a)]. The generalization for any arbitrary circuit will be discussed later. Black dots represent data tokens, whereas white dots represent bubbles. In the model, we assume that all latches become transparent when the control signal is high. The events $A+$ and $A-$ represent rising and falling transitions of the control signal A, respectively.

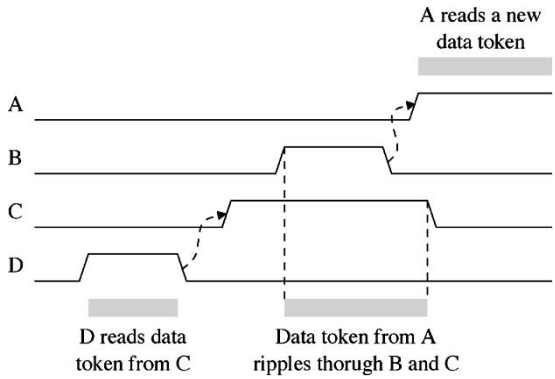


Fig. 4. Timing diagram of linear pipeline in Fig. 3(a)–(d).

Fig. 3(b) depicts a fragment of the unfolded MG representing the behavior of the latches. There are three types of arcs in this model (we only refer to those in the first stage of the pipeline).

1) $A+ \rightarrow A- \rightarrow A+$, which simply denotes that the rising and falling transitions of each signal must alternate.
2) $B- \rightarrow A+$, which denotes the fact that for latch A to read a new data token, B must have completed the reading of the previous token coming from A. If this arc is not present, data overwriting can occur, or in other words, hold constraints can be violated.
3) $A+ \rightarrow B-$, which denotes the fact that for latch B to complete the reading of a data token coming from A, it must first wait for the data token to be stored in A. If this arc is not present, B can "read a bubble" and a data token can be lost, or in other words, setup constraints can be violated.

The marking in Fig. 3(b) represents a state in which all latch control signals are low and the events $B+$ and $D+$ are enabled, i.e., the latches B and D are ready to read the data tokens from A and C, respectively.

Fig. 3(c) shows the MG that derives from the unfolded graph in Fig. 3(b). A simplified notation is used in Fig. 3(d) to represent the same graph, substituting each cycle $x \rightleftarrows y$ by a double arc $x \bullet\!\!-\!\!\bullet y$, where the token is located close to the enabled event in the cycle (y in this example).

It is interesting to notice that the previous model is more aggressive than the classical one generating nonoverlapping phases for latch-based designs. As an example, the following sequence can be fired in the model of Fig. 3(a)–(d)

$$D+\ D-\ C+\ B+\ B-\ A+\ C- \cdots.$$

After the events $\langle D+\ D-\ C+\ B+ \rangle$, a state in which $B = C = 1$ and $A = D = 0$ is reached, where the data token stored in A is rippling through the latches B and C. A timing diagram illustrating this sequence is shown in Fig. 4.

But can this model be generalized beyond linear pipelines? Is it valid for any arbitrary netlist? Which properties does it have? We now show that this model can be extended to any arbitrary netlist while preserving a property that makes the circuits observationally equivalent to their synchronous versions: flow equivalence [17].

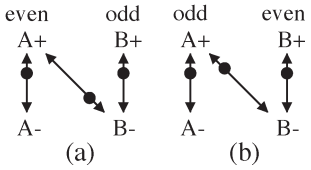


Fig. 5. Synchronization between latches: $A \rightarrow B$.

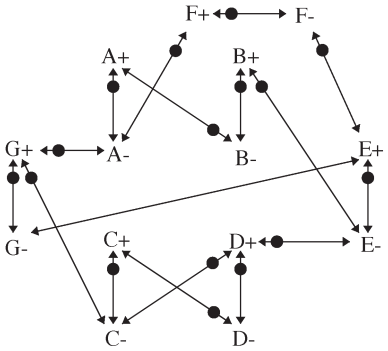


Fig. 6. Desynchronization model for circuit in Fig. 2.

C. General Desynchronization Model

The general desynchronization model is shown in Fig. 5. For each communication between an even latch and an odd latch, the synchronization depicted in Fig. 5(a) must be defined. If the communication is between odd and even, the one in Fig. 5(b) must be defined. Note that the only difference is the initialization. The odd latches are always enabled in the initial state to read the data tokens from the even latches.

By abutting the previous synchronization models, it is possible to build the model for any arbitrary netlist, as shown in Fig. 6. The MGs obtained by properly abutting the models in Fig. 5 are called circuit MGs (CMGs).

We now show that a desynchronized circuit mimics the behavior of its synchronous counterpart. For that, it must be proven that:

- a desynchronized circuit never halts (liveness);
- all computations performed by a desynchronized circuit are the same as the ones performed by the synchronous counterpart (flow equivalence).

The remainder of this section is devoted to prove these two statements.

D. Liveness

For the proof of liveness, the reader must bear in mind the meaning of the double arcs $x \longleftrightarrow y$, which represent $x \underset{\longleftarrow}{\overset{\bullet}{\rightarrow}} y$

Theorem 4.1: Any CMG is live.

Proof: By Theorem 3.1, it is enough to prove that there is no directed circuit in the CMG without any token. Rather than giving a formal proof, we merely give hints that can easily lead the reader to a complete proof. It is easy to see that there is no way to build an unmarked path longer than three arcs. As an example, let us try to find the longest unmarked path from $D+$ in the CMG of Fig. 3(c). After building the path $D+ \rightarrow D- \rightarrow$

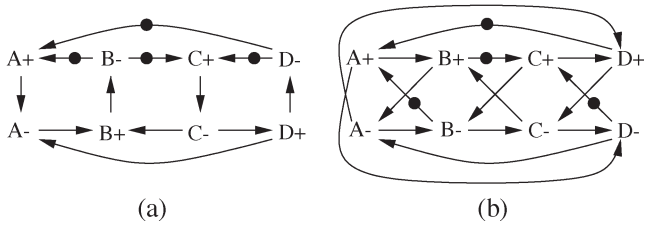


Fig. 7. Synchronization of ring. (a) Live model. (b) Nonlive model.

$C+ \rightarrow C-$, it is not possible to extend it unless a marked arc is included, either $C - \bullet\!\!\rightarrow C+$ or $C - \bullet\!\!\rightarrow B+$. A case-by-case study leads to a complete proof. ■

Liveness guarantees something crucial for the model: absence of deadlocks. This property does not hold automatically for every "reasonable" model. Fig. 7 depicts two different desynchronization models for a ring, which can be obtained by connecting the output of latch D with the input of latch A in Fig. 3(a). Fig. 7(a) depicts a nonoverlapping model between adjacent latches, whereas Fig. 7(b) uses a four-phase handshake with the sequence $A+\ B+\ A-\ B-$ for each pair of adjacent latches.

When building the protocol for a ring, the second model is not live due to the unmarked cycle

$$A- \rightarrow B- \rightarrow C- \rightarrow D- \rightarrow A- .$$

One can easily understand that after firing events $A+$ and $C+$, the system enters a deadlock state. It is also easy to prove that this model is live for acyclic netlists.

The acid test of liveness for a handshake protocol consists of connecting two controllers back to back for a two-stage ring [see Fig. 3(e)]. Fig. 3(f) depicts the unfolded behavior after including all causality constraints for the communications $A \rightarrow B$ and $B \rightarrow A$. The folded behavior is shown in Fig. 3(g), which can also be obtained by combining the synchronization models of Fig. 5(a) and (b). Several arcs become redundant, thus, deriving the simplified model shown in Fig. 5(h).

Interestingly, the resulting protocol derived from the "aggressive" concurrent model is "naturally" transformed into one that is nonoverlapping, live, and safe. Note that a two-stage ring is typically derived from the implementation of a finite-state machine, in which the current state stored in a register is fed back to the same register after going through the combinational logic that calculates the next state. As an example, the handshake protocol between latches C and D in Fig. 2 (see Fig. 6 also) becomes nonoverlapping.

E. Flow Equivalence

In this section, we prove that a desynchronized circuit mimics its synchronous counterpart. We show that, for each latch, the value stored at the ith pulse of the control signal is the same as the value stored at the ith cycle of the synchronous circuit.

We first present some definitions that are relevant for synchronous circuits.

Definition 4.1 (Synchronous Behavior): Given a block A (combinational logic and latch), we call F_A the logic function

TABLE I
OBSERVABLE TRACES AT LATCHES OF SYNCHRONOUS CIRCUIT

cycle	*clk*	trace
initial	0	$E_0^1 \ldots E_0^n$ $O_0^1 \ldots O_0^m$
1	1	$E_0^1 \ldots E_0^n$ $O_0^1 \ldots O_0^m$
	0	$E_1^1 \ldots E_1^n$ $O_1^1 \ldots O_1^m$
2	1	$E_1^1 \ldots E_1^n$ $O_2^1 \ldots O_2^m$
$\vdots$	$\vdots$	$\vdots$
i	1	$E_{i-1}^1 \ldots E_{i-1}^n$ $O_i^1 \ldots O_i^m$
	0	$E_i^1 \ldots E_i^n$ $O_i^1 \ldots O_i^m$
$i+1$	1	$E_i^1 \ldots E_i^n$ $O_{i+1}^1 \ldots O_{i+1}^m$

calculated by the combinational logic. We call A_i the value stored in A's latch after the ith clock cycle. Let us call $E^1 \ldots E^p$ the (even) predecessor latches of an odd latch O, and $O^1 \ldots O^p$ the (odd) predecessor latches of an even latch E. Then

1) $O_i = F_O(E_{i-1}^1, \ldots, E_{i-1}^p)$;
2) $E_i = F_E(O_i^1, \ldots, O_i^p)$;

where all even blocks store a known initial value at cycle zero.

For the sake of simplicity here, we model a closed circuit, i.e., one without primary inputs from the environment. The environment can be considered explicitly either by slightly changing the proofs, or by modeling it as a nondeterministic function. The latter mechanism also allows us to show how a desynchronized circuit can be interfaced with a synchronous one (the environment), namely by driving its input handshake signals with the global clock and ignoring its output handshake signals. The latter must be shown to follow the correct protocol by means of appropriate timing assumptions.

The behavior of a synchronous circuit can be defined as the set of traces observable at the latches. If we call $E^1 \ldots E^n$ and $O^1 \ldots O^m$ the set of even and odd latches, respectively, the behavior of the circuit can be modeled by an infinite trace in which each element of the alphabet is an $(n+m)$-tuple of Table I.

If we project the trace onto one of the latches, say A, we obtain a trace $A_0 A_1 \ldots A_i \ldots$, i.e., the sequence of values stored in latch A at each cycle.

We now present a lemma that guarantees a good alternation of pulses between adjacent latches.

Lemma 4.1 (Synchronic Distance): Let $(\Sigma, \rightarrow, M_0)$ be a CMG, E and O two adjacent blocks such that E is even and O is odd, and σ a sequence fireable from M_0.

1) If E transfers data to O, then

$$\overline{\sigma}(E+) \leq \overline{\sigma}(O-) \leq \overline{\sigma}(E+) + 1.$$

2) If O transfers data to E, then

$$\overline{\sigma}(E-) \leq \overline{\sigma}(O+) \leq \overline{\sigma}(E-) + 1.$$

Proof: Both inequalities hold by the existence of the double arcs $E+ \leftrightarrow O-$ or $O+ \leftrightarrow E-$ that guarantee the alternation between both events. The initial marking is the one that makes the difference between the even-to-odd and odd-to-even connections. ■

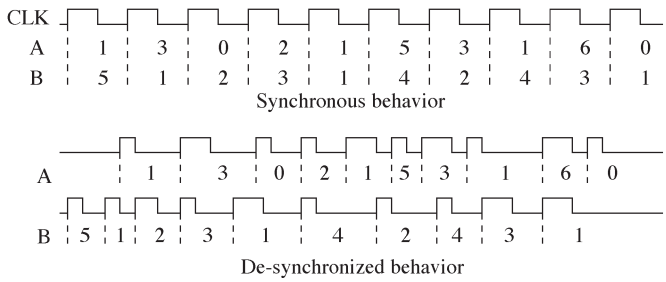


Fig. 8. Flow equivalence.

This lemma states that adjacent latches alternate their pulses correctly, which is crucial to preserve flow equivalence.[1]

We now present the notion of flow equivalence [17], which is related to that of the synchronous behavior in [24], in terms of the projection of traces onto the latches of the circuit.

Definition 4.2 (Flow Equivalence): Two circuits are flow equivalent if:

1) they have the same set of latches;
2) for each latch A, the projections of the traces onto A are the same in both circuits.

Intuitively, two circuits are flow equivalent if their behavior cannot be distinguished by observing the sequence of values stored at each latch. This observation is done individually for each latch and, thus, the relative order with which values are stored in different latches can change, as illustrated in Fig. 8. The top diagram depicts the behavior of a synchronous system by showing the values stored in two latches, A and B, at each clock cycle. The diagram at the bottom shows a possible desynchronization. From the diagram, one can deduce that latches A and B cannot be adjacent (see Lemma 4.1), since the synchronic distance of their pulses is sometimes greater than 1 (e.g., B has received five pulses after having stored the values $\langle 5, 1, 2, 3, 1 \rangle$, while A has only received two pulses storing $\langle 1, 3 \rangle$).

The following theorem is the main theoretical result of this paper.

Theorem 4.2: The desynchronization model preserves flow equivalence.

Proof: By induction on the length of the trace.

Induction hypothesis: For any latch A, flow equivalence is preserved for the first $i-1$ occurrences of $A-$ and until a marking is reached with the ith occurrence of $A-$ enabled (see Fig. 9). The marking of the arcs $E^k+ \rightarrow E^k- \rightarrow E^k+$ or $O^k+ \rightarrow O^k- \rightarrow O^k+$ is irrelevant for the hypothesis.

Basis: The induction hypothesis immediately holds for odd latches in the initial state [Fig. 9(a)]. For even latches [see Fig. 9(b)], it holds after having fired $O^1 + \ldots O^p+$ once from

[1]A similar result was derived in [24], also based on the Marked Graph Theory, using, however, a very different circuit structure and implementation philosophy.

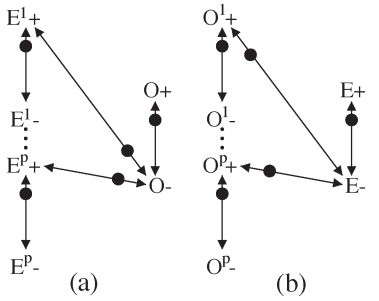


Fig. 9. Illustration of Theorem 4.2.

the initial state. This single firing preserves flow equivalence since each latch O^k receives the value

$$O_1^k = F_{O^k}\left(E_0^1, \ldots, E_0^m\right)$$

obtained from the initial value of $E^1, \ldots, E^m$, the (even) predecessor latches of O^k.

Induction step (case O odd): Since the ith firing of $O-$ is enabled, we know that each E^k+ transition has fired $i-1$ times (see Lemma 4.1) and, by the induction hypothesis, stores the value E_{i-1}^k. Therefore, the next firing of $O-$ will store the value

$$O_i = F_O\left(E_{i-1}^1, \ldots, E_{i-1}^p\right)$$

which preserves flow equivalence. Moreover, the ith firing of E^k+ will occur after O has been closed, since the arc $O- \rightarrow E^k+$ forces that ordering. This guarantees that no data overwriting occurs on latch O.

Induction step (case E even): Since $E-$ has fired $i-1$ times, then O^k+ has fired i times, according to Lemma 4.1. Since the O^k latches are odd, they store the values O_i^k, by the induction hypothesis and the previous induction step for odd latches. The proof now is reduced to case of O being even, in which

$$O_i = F_O\left(E_i^1, \ldots, E_i^p\right).$$

This concludes the proof, since induction guarantees flow equivalence for any latch A and for any number firings of $A-$. ■

V. Handshake Protocols for Desynchronization

Section IV presented a model for desynchronization that defines the causality relations among the latch control signals for a correct flow of data in the datapath. Now, it is time to design the controllers that implement that behavior.

Several handshake protocols have been proposed in the literature for such purpose. The questions are as follows: Are they suitable for a fully automatic desynchronization approach? Is there any controller that manifests the concurrency of the desynchronization model proposed in this paper?

We now review the classical four-phase micropipeline latch control circuits presented in [15]. For that, the specification of each controller [15, Figs. 5, 7, and 11] has been projected onto the handshake signals (Ri, Ai, Ro, Ao) and the latch control signal (A), thus, abstracting away the behavior of the internal state signals.[2] The projection has been performed by preserving observational equivalence.[3]

Fig. 10(a)–(c) shows the projections of the controllers from [15]. The leftmost part of the figure depicts the connection between an even and an odd controller generating the latch control signals A and B, respectively. The rightmost part depicts only the projection on the latch control signals when three controllers are connected in a row.

The controllers from [15] show less concurrency than the desynchronization model. For this reason, we also propose a new controller implementing the protocol with maximum concurrency proposed in this paper [Fig. 10(e)]. For completeness, a handshake decoupling the falling events of the control signals (fall decoupled) is also described in Fig. 10(d).

In all cases, it is crucial to properly define the initial state of each controller, which turns out to be different for the even and odd controllers. This is an important detail often missed in many papers describing asynchronous controllers.

The question now is: Which ones of these controllers are suitable for desynchronization? Instead of studying them one by one, we present a general study of four-phase protocols, illustrated in Fig. 11. The figure describes a partial order defined by the degree of concurrency of different protocols. Each protocol has been annotated with the number of states of the corresponding state graph. The MGs in the figure do not contain redundant arcs.

An arc in the partial order indicates that one protocol can be obtained from the other by a local transformation (i.e., by moving the target of one of the arcs of the model). The arcs $A+ \leftrightarrow A-$ and $B+ \leftrightarrow B-$ cannot be removed for obvious reasons (they can only become redundant). For example, the semi-decoupled protocol (five states) can be obtained from the rise-decoupled protocol (six states) by changing the arc[4] $A+ \multimap B-$ to the arc $A+ \multimap B+$, thus, reducing concurrency.

The model with eight states, labeled as "desynchronization model," corresponds to the most concurrent model presented earlier in this paper, for which liveness and flow equivalence have been proven in Section IV. The other models are obtained by successive reductions or increases of concurrency.

The nomenclature rise and fall decoupled has been introduced to designate the protocols in which the rising or falling edges of the pulses have been decoupled, respectively. The rise-decoupled protocol corresponds to the fully decoupled one proposed in [15].

In [6], the following results were proven for the models shown in Fig 11.

1) All models except the simple four-phase protocol (top left corner) are live.
2) All models except the two models at the bottom are flow equivalent.

[2]In fact, A is the signal preceding the buffer that feeds the latch control signal. The polarity of the signal has been changed to make the latch transparent when A is high.

[3]For those users familiar with petrify, the projection can be obtained by hiding signals with the option $-$hide.

[4]Note that this arc is not explicitly drawn in the picture, because it is redundant.

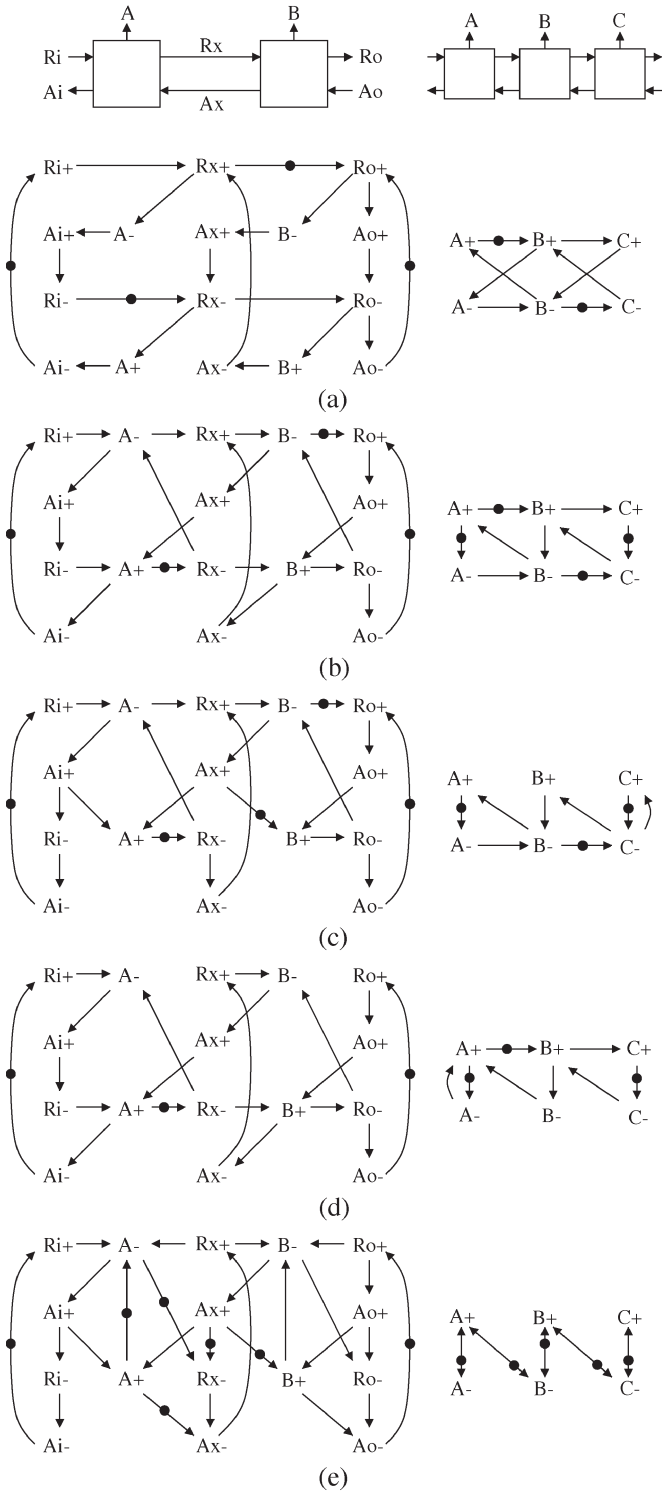


Fig. 10. Handshake protocols. (a) Simple four-phase control (Furber and Day). (b) Semi-decoupled four-phase control (Furber and Day). (c) Fully decoupled four-phase control (Furber and Day). (d) Fall-decoupled control. (e) De-synchronization control (this paper).

3) Desynchronization can be performed by using any hybrid combination of the live and flow equivalent models shown in the figure (i.e., using different types of controllers for different latches).

These results offer a great flexibility to design different schemes for desynchronized circuits.

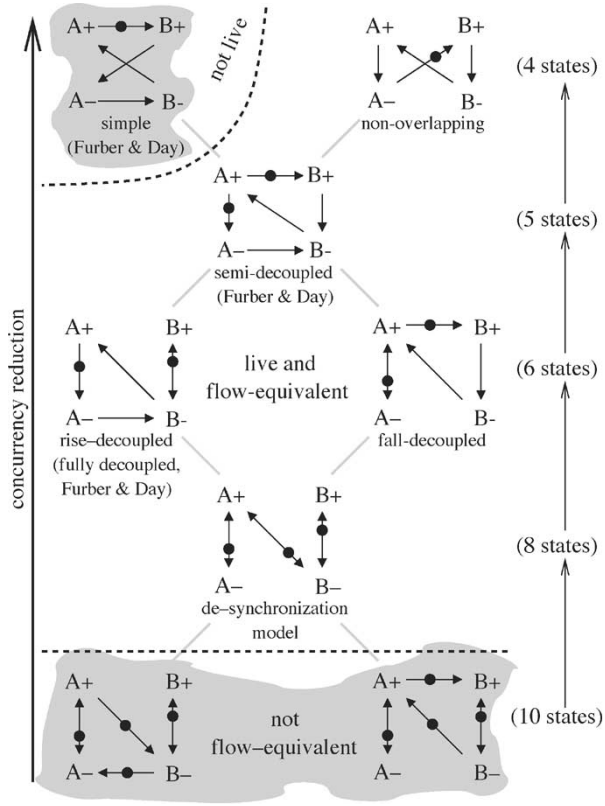


Fig. 11. Different degrees of concurrency in handshake protocols for desynchronization.

VI. Implementation of Desynchronization Controllers

The protocols described in Section V can be implemented in different ways using different design styles. In this section, we describe a possible implementation of the semi-decoupled four-phase handshake protocol proposed by Furber and Day [15]. We present an implementation with static complementary metal–oxide–semiconductor (CMOS) gates, while the original one was designed at transistor level. There are several reasons for the selection of this protocol with this particular design style.

1) We pursue an approach suitable for semicustom design using automatic physical layout tools.
2) The semi-decoupled protocol is a good tradeoff between simplicity and performance.
3) The pulsewidth of the latch control signals will be similar if all controllers are similar. Moreover, the depth of the datapath logic usually has a delay that can be overlapped with the controller's delay. Therefore, the arcs $A+ \rightarrow B+$ and $A- \rightarrow B-$ do not impose performance constraints in most cases.

In case of time-critical applications, other controllers can be used, including hybrid approaches combining protocols different from the ones shown in Fig. 11.

Fig. 12 depicts an implementation of a pair of controllers (even and odd) for a fragment of datapath. The figure also shows the MGs modeling the behavior of each controller. The

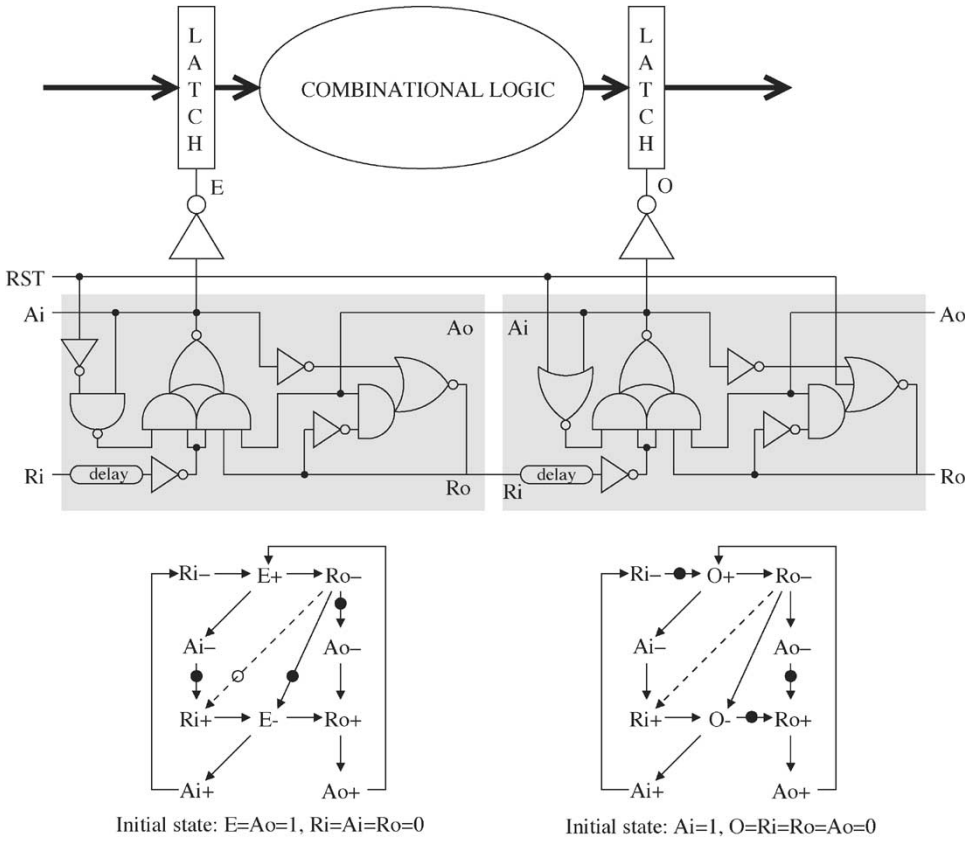


Fig. 12. Implementation of semidecoupled controllers for even (E) and odd (O) latches.

only difference is the initial marking, which determines the reset logic (signal RST).

Resetting the controllers is crucial for a correct behavior. In this case, the even latches are transparent and the odd latches are opaque in the initial state. With this strategy, only the odd latches must be reset in the datapath. The implementation also assumes a relative timing constraint (arc $Ro- \rightarrow Ri+$) that can be easily met in the actual design.[5]

The controllers also include a delay that must match the delay of the combinational logic and the pulsewidth of the latch control signal.

Each latch control signal (E and O) is produced by a buffer (tree) that drives all the latches. If all the buffer delays are similar, they can be neglected during timing analysis. Otherwise, they can be included in the matched delays, with a similar but slightly more complex analysis.

In particular, the delay of the sequence of events

$$E+ \rightarrow \frac{Ro}{Ri}- \rightarrow \boxed{\text{logic delay}} \rightarrow O+$$

is the one that must be matched with the delay of the combinational logic plus the clock-to-output delay of a latch. The event $Ro/Ri-$ corresponds to the falling transition of the signal Ro/Ri between the $E-$ and $O-$ controllers. On the other hand, the delay of the sequence

$$O+ \rightarrow \frac{Ai}{Ao}- \rightarrow \frac{Ro}{Ri}+ \rightarrow \boxed{\text{pulse delay}} \rightarrow O-$$

is the one that must be matched with the minimum pulsewidth. It is interesting to note that both delays appear between transitions of the control signals of Ri and O, and can be implemented with just one asymmetric delay.

The control can be generalized for multiple-input/multiple-output blocks. In that case, the req/ack signals of the protocols must be implemented as a conjunction of those coming from the predecessor and successor controllers, by using C-elements. As an example, Fig. 13 shows the desynchronization control for the circuit depicted in Fig. 2.

VII. Timed Model

The model presented in Section IV guarantees synchronous equivalence with zero-delay components. However, computational blocks and latches have delays that impose a set of timing constraints for the model to be valid.

Fig. 14 depicts the timing diagram for the behavior of two latches in a pipeline. The signals I and O represent the inputs and outputs of the latches. The signal L is the control of the latch ($L = 1$ for transparent).

We focus our attention on latch A. As soon as O_A becomes valid, the computation for block B starts. Latch B can become

[5]This assumption also allows us to simplify the implementation proposed in [15]: The equation for $A+$ becomes R_{in} instead of $R_{\text{in}} \wedge \neg R_{\text{out}}$.

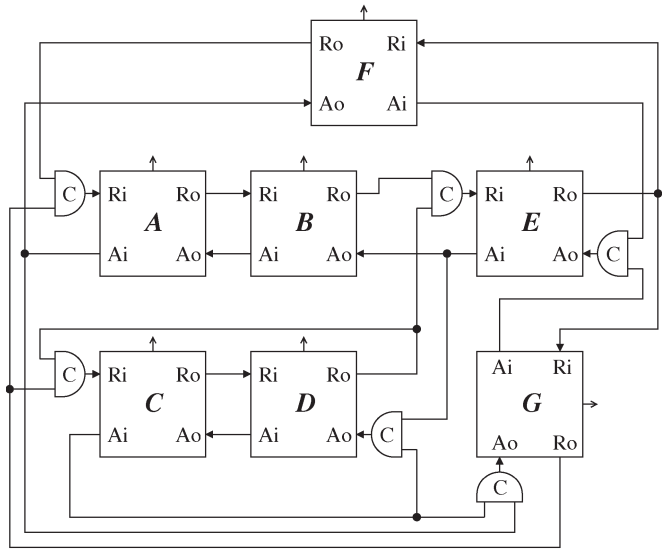


Fig. 13. Desynchronized control for netlist in Fig. 2.

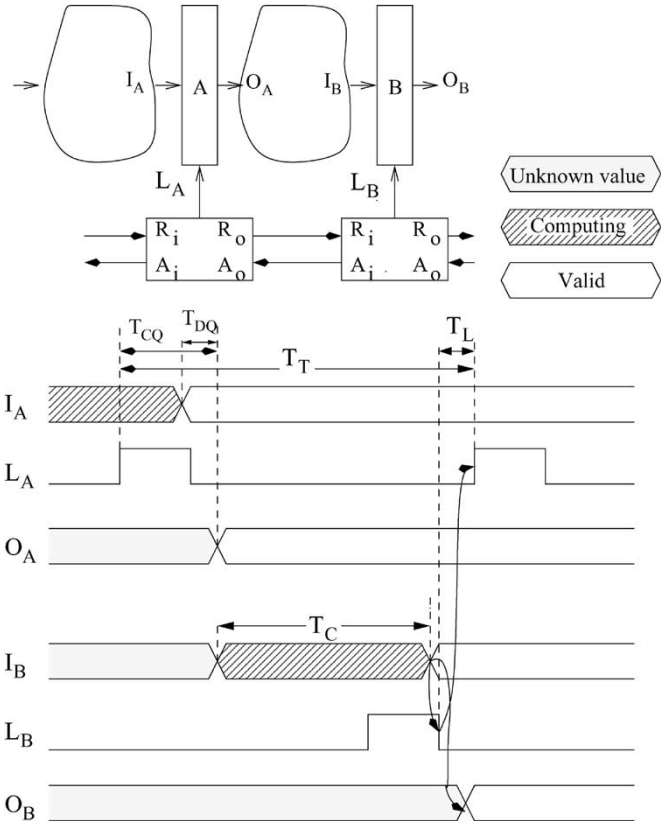


Fig. 14. Timing constraints for asynchronous controllers.

transparent before the computation completes. Opening a latch in advance is beneficial for performance, because it eliminates the time for capturing data from the critical path.

Once the computation is over, the local clock L_B of the destination latch B immediately falls. This is possible, because modern latches have zero setup time [8].

Assuming that all controllers have similar delays, the following constraint is required for correct operation

$$T_{\mathrm{T}} \geq T_{\mathrm{CQ}} + T_{\mathrm{C}} + T_{\mathrm{L}}. \tag{1}$$

The constraint (1) indicates that the cycle time of a local clock (measured as a delay T_{T} between two rising edges of L_A) must be greater than the delay of local clock propagation though a latch (T_{CQ}) plus the delay of the computational block (T_{C}) plus the latch controller delay (T_{L}). The control overhead in this scheme is reduced to a single delay T_{L}, because the control handshake overlaps with the computation cycle due to the early rising of the local clock. The constraint assumes that the depth of combinational logic is sufficiently large to amortize the overlapping part of the handshake. The latter is true for ASIC designs, which typically have more than 20 levels of logic between adjacent registers.

Inequality (1) guarantees the satisfaction of setup constraints for the latch. Note that hold constraints in a desynchronized circuit are ensured automatically, because for any valid protocol (see Fig. 11), the clock of any predecessor latch rises only after the clock of its successor latch had fallen. This makes it impossible to have races between two consecutive data items at latch inputs.

A. Timing Compatibility

In Section IV, we showed that synchronous and desynchronized circuit are indistinguishable when observing event sequences at the outputs of corresponding latches. This section shows that the temporal behaviors of these circuits are also similar, i.e., the deadlines on computation imposed by a clock are met in a desynchronized circuit as well. Based on these two results (temporal and behavioral equivalence), one could replace any synchronous circuit by its desynchronized counterpart without visible changes. This makes the suggested design methodology modular and compositional.

Note that this analysis uses the same models and margins for both designs. However, as discussed in Section I, desynchronized circuits behave much better than synchronous ones with respect to tolerance of design, manufacturing, and environmental uncertainties. Hence, a desynchronized circuit can generally run at typical, as opposed to worst case speed, i.e., 1.2–2× faster than its synchronous counterpart.

In a synchronous flip-flop-based circuit, the cycle time T_{S} is bounded by [8]

$$T_{\mathrm{S}} \geq T_{\mathrm{C}} + T_{\mathrm{setup}} + T_{\mathrm{skew}} + T_{\mathrm{CQ}} \tag{2}$$

where T_{C}, T_{setup}, T_{skew}, and T_{CQ} are maximum combinational logic, setup, skew, and clock-to-output times, respectively. Comparing inequalities (1) and (2) and bearing in mind that due to retiming the maximal computation time in a desynchronized circuit can only be reduced, one can conclude the difference between the cycle time of desynchronized circuit T_{T} and the cycle time T_{S} of the corresponding synchronous design is approximately $T_{\mathrm{L}} - (T_{\mathrm{setup}} + T_{\mathrm{skew}})$. In all our design examples, it is at most a few percent.

There is a small caveat in the above statement. The notion of a cycle time is well defined only for a circuit with a periodic clock. In a desynchronized system, the separation time between adjacent rising edges of the same local clock might change during operation (see, e.g., Fig. 8). Therefore, we should compare

the perfect periodic behavior of the synchronous circuit with the nonperiodic one of the desynchronized circuit.

The following properties provide a basis for relating these two systems in a sound way. Informally, they show that latches that belong to critical computational paths of a desynchronized system have a well-defined constant cycle time, while the rest of the latches operate in plesiochronous mode [14], in which their local clocks have transitions at the same rate, only with bounded time offsets from each other.

Property 7.1: If in a desynchronized circuit, the computation delay $T_{\rm C}$ is the same for every combinational block, then the separation time between adjacent rising edges of every local clock is also the same and equals $T_{\rm T}$.

The proof is trivial, because a perfectly balanced desynchronized system behaves like a synchronous one with all local clocks paced at the same rate.

Suppose that in the initial state of a desynchronized system, local controllers for odd latches produce a rising transition. Then Property 7.1 says that in a perfectly balanced desynchronized system, the ith rising transition of the local clock of any odd latch happens at time $(i-1)*T_{\rm T}$. Below, we show that time stamps $(i-1)*T_{\rm T}$ provide an upper bound for the ith rising transition at an odd latch in an arbitrary (possibly unbalanced) desynchronized system. A similar relationship can be defined for the clocks of even latches by adding a constant phase shift $T_{\rm ph}$ to time stamps $(i-1)*T_{\rm T}$. Without losing generality, one can, thus, consider only one type of latches only (e.g., odd).

Property 7.2: In any desynchronized circuit, the ith rising transition of a local clock of an odd latch cannot appear later than $(i-1)*T_{\rm T}$.

Proof: Analysis of the firing time of the ith instance A_i of event A in an MG G is reduced to the following procedure [27]: 1) annotate each edge of the graph with the corresponding delay; 2) construct the unfolding of the graph up to A_i; and 3) find the longest path from the set of events enabled initially (fireable at time $t=0$) to A_i.

From Property 7.1 it follows that, for a well-balanced desynchronized circuit, the length of the longest path to the ith rising event at any odd latch is $(i-1)*T_{\rm T}$. For an arbitrary unbalanced circuit, the weight of edges in G could only be reduced from their worst case values. This immediately implies that none of the odd latches could have the ith rising transition happening later than $(i-1)*T_{\rm T}$. ■

Let us call a latch critical if the delay of a combinational block connected to its output is equal to the maximal computational delay $T_{\rm C}$. From Properties 7.1 and 7.2, it follows that the separation time between any successive pair of rising edges of clocks for the same critical latch is constant and equal to $T_{\rm T}$. The synchronic distance between adjacent latches does not exceed 1 (Lemma 4.1). Therefore, after at most one cycle, latches adjacent to a critical latch must adapt their cycle time to $T_{\rm T}$ (after one cycle, they are paced by a critical latch). Pushing these arguments further implies that in a connected desynchronized system, any latch sooner or later settles to the cycle time $T_{\rm T}$. This shows that the behavior of a desynchronized circuit has a well-defined periodicity, similar to that of a synchronous one, paced by a common clock.

Embedding of a desynchronized circuit with clock cycle $T_{\rm T}$ into a synchronous environment with a clock cycle $T_{\rm S}$: $T_{\rm S} \geq T_{\rm T}$ results in the latches at the asynchronous/synchronous boundary becoming critical, since they are paced by a slower external clock $T_{\rm S}$.This consideration shows that desynchronized and synchronous systems are compatible in terms of timing, because their external timed behavior is the same as long as both use the same margins to ensure safe operation.

As argued above, when average performance matters more than worst case, the desynchronized circuit can work faster than the synchronous one, because it requires much smaller margins.

VIII. Physical Design, Verification, and Testing

Physical design is a key step of our methodology. At this stage, we insert the matched delay chains that ensure that setup times are satisfied. This can no longer be done during logic synthesis in the case of ASICs implemented using deep submicron technologies, as the wire delay models are too approximate before routing. Even placement information is no longer sufficient for an accurate estimation in large chips.

The placement and global routing step is the ideal point in the flow to compute the delay of the logic and wiring, since detailed routers can exploit multiple layers of metal to ensure a close correspondence with the requirements imposed by the global router (including layer assignment). Unfortunately, inserting large delay chains during placement may be problematic, since it would significantly disrupt the circuit layout and force placement information to be recomputed. In this paper, we take the opposite approach. We propose to use very pessimistic delay models for prelayout timing analysis, insert longer delay chains than needed, and perform the placement. After that, the delay chains can be resized in-place or reduced with minimal effect on the placement.

Note that synchronous timing analysis can be used "as is," by providing the appropriate reference points between which delays must be computed. For the datapath, this does not pose any special problem. In case internal delays of asynchronous controllers, which contain loops, need to be computed, this may be achieved by specifying the path endpoints explicitly in order to apply static analysis only to those linear portions of the circuit.

Controllers are generated for multibit registers in order to reduce the area and wiring overhead. An optimal grouping technique would take into account both a cost function that favors larger groups to reduce overhead, and a similarity function that favors grouping registers with similar fanins and fanouts, as in bit-slice datapaths. This issue is considered in detail in [13].

Contemporary placement tools are able to make incremental modifications when some portions of the delay chains (less than 5% of the total area for a typical design) are removed. Thus, global routing and delay estimations are not significantly affected, and single-iteration convergence can be achieved. This is a very significant advantage in order to speed up design times.

A. Matched Delay Insertion

The flow that we used for the desynchronization approach begins with a synthesizable HDL specification [e.g., Verilog/

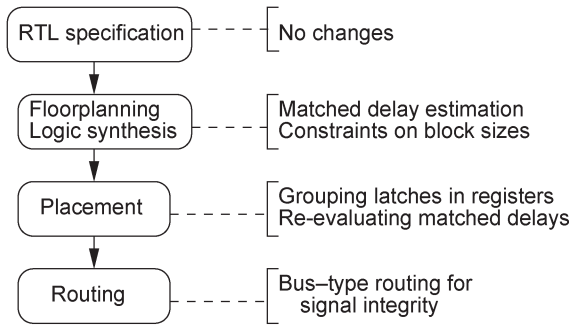


Fig. 15. Changes in standard synchronous design flow.

very-high-speed integrated circuit (VHSIC) HDL (VHDL)], using the conventional synchronous HDL constructs. Next, each datapath element is synthesized for the target cycle time T_{T}, using a conventional synthesis tool. Due to the load of the local clock by the registers of the datapath block, buffers are inserted at this stage.

The circuit is analyzed using conventional static timing analysis (STA) tools to estimate the delay of each matched delay element. These matched delay elements are generated and embedded into latch controllers. At this stage, the datapath blocks and their corresponding latch controllers are combined to form the complete netlist of the desynchronized circuit. Once the complete netlist is assembled, it may be simulated and its correct operation verified using a gate-level simulator.

The circuit is then placed and routed, and the postlayout delays are extracted. The pessimistic delays used for prelayout timing analysis are now more precise, and redundant NOT and NAND gate pairs can be removed from the delay chains by exploiting the incremental place-and-route capabilities of modern tools. The possible modifications of different stages in conventional automatic design flow for doing desynchronization are shown in Fig. 15.

B. Verification

Conventional equivalence checking tools can be used for the verification of the datapath, since desynchronization keeps it intact. Some extra effort is required to check that the matched delays of the controllers generate the appropriate timing separations between the enable signal of the latches to accommodate the delays of the combinational logic. This can be easily verified after layout with static analysis tools.

C. Design for Testability

The datapath can be tested by using scan path insertion with synchronous tools. A clock can be distributed to every register and used only in test mode. Local acknowledge wires in test mode allow one to build this network without skew problems. Thus, it is considerably smaller than in the synchronous case, where it must satisfy tight skew constraints. Moreover, it is kept idle during normal operation.

Asynchronous handshake circuits can also be tested by using a full-scan methodology, as discussed in [28]. This has a performance and area overhead, but it is essential for the acceptance of the methodology. The goal is to ensure full coverage. Handshake circuits are self-checking, and the work in [22] showed that 100% stuck-at coverage can be achieved for asynchronous pipelines using conventional test pattern generation tools.

D. Binning (or Speed Grading) and Yield Improvement

One advantage of desynchronization is that it eases some form of circuit binning (also called speed grading) based on performance. If we assume that the performance of similar objects (e.g., transistors, interconnects on the same layer) track each other within relatively small regions of the layout, then we can assume that the performance of a die is determined by the delay chains, while the delay of the logic is proportionately smaller, and, thus, setup constraints are automatically satisfied. This means that the request and acknowledge wires at the boundaries of the circuit can be used to measure the worst case response time of every individual die.

In other terms, the maximum speed of a die can be established by only looking at the timing of transitions of some output signals with respect to the clock input, without the need for expensive at-speed delay testing equipment. This allows one to classify dies according to their maximum operational speed (binning), which so far was only used for leading-edge central processing units (CPUs; from Intel, AMD, Sun) due to the huge cost of at-speed testing equipment. It also allows one to tune the process, by observing the performance of whole circuits, not just of small delay chains on test chips.

IX. Experiments

In this section, we present results for the application of desynchronization to two large realistic circuits, a DES core and a DLX microprocessor. The DES core was designed using a 0.18-μm standard-cell library from UMC. The DLX core was designed 1) with the same UMC library and 2) with a 0.25-μm library. The latter chip has been fabricated.

A. DES Core

A high-throughput DES core is essentially a 16-stage pipeline, in which each stage implements a single iteration of the DES algorithm. The algorithm operates on a 64-bit data stream and 64-bit keys. It consists of permutations, shifts, and a limited amount of logic. Thus, the depth of each of these stages is small. DES constitutes a worst case for our methodology, since controller overhead could be significant with respect to datapath logic delays.

We first implemented a synchronous edge-triggered flip-flop design for the 16-stage DES design in the 0.18-μm VST-UMC standard-cell technology library. We compared our synchronous implementation with available synchronous open-source DES cores (from www.opencores.org) and verified that it has indeed similar area and performance. We then employed the method of desynchronization in order to derive a desynchronized dual-latch design. The type of controllers used in this design are based on the desynchronization model, i.e., with the maximum possible concurrency. Table II contrasts the

TABLE II
SYNCHRONOUS VERSUS DESYNCHRONOUS DES CASE STUDY

	Sync. Flip-Flop DES	De-Sync. Latch DES
Cycle Time	1.60ns	1.66ns
Latency	25.77ns	26.57ns
Power Cons.	328.92 mW	288.78 mW
Area	565542 μm^2	685406 μm^2

TABLE III
DESYNCHRONIZED DES: AREA BREAKDOWN

	Area	% Total Area
Async. Control	4292.8 μm^2	0.63%
Delay Elements	4032.64 μm^2	0.59%
Registers	281120 μm^2	41.02%
Comb. logic	395952 μm^2	57.77%

characteristics of the two designs. All data are postsynthesis prelayout results based on gate-level simulations.

The DES cycle time is the time it takes to perform a single iteration of the DES algorithm. A total of 16 iterations are required to produce the 64-bit result, thus, resulting in the latency value shown in the table. The power consumption of the DES designs was measured by performing switching activity annotation of the circuit during simulation.

As can be seen from these figures, the desynchronized design, despite an area increase of approximately 22%, presents only a very slight difference in cycle time and a power improvement of slightly over 12%.

Table III shows the area breakdown of the desynchronized DES in terms of asynchronous control, delay elements, registers, and combinational logic. In fact, most of the area overhead comes from using two latches instead of a single flip-flop. The register area of the asynchronous design is approximately 218 560 μm^2.

This example shows that, even for a design containing a very limited amount of combinational logic, desynchronization still manages to hide control overhead and achieve comparable performance at lower power.

B. DLX ASIC Core

The second example that we discuss is a desynchronized version of the DLX processor [19], called ASynchronous open-source Processor Ip of the Dlx Architecture (ASPIDA), designed using the semi-decoupled controllers depicted in Fig. 12. Fig. 16 shows the overall structure, including the multiplexed clocks and five architectural pipeline stages, four of which actually correspond to circuit blocks (at the circuit level, WB is merged with ID). Each block is controlled by its own latch controller. The arrows of the latch controllers correspond to the Ro and Ao signals, and illustrate the datapath dependencies.

Stages ID, EX, and MEM form a ring. ID is the heart of the processor containing the register file and all hazard-detection logic. It also synchronizes instructions leaving MEM (for WB) with instructions coming from IF. Data hazard detection takes place by ID comparing the output register of instructions in other pipeline stages and their opcodes, and deciding on inserting the correct number of NOPs.

After the initial synthesis of each circuit block using latches (without retiming), the whole design is optimized incrementally to meet all timing requirements. Max-delay constraints between latches are used to ensure cycle time in the datapaths. The control blocks are left untouched by the synthesis tool. Then the gate-level netlist and matching timing constraints are placed and routed.

Table IV shows the post-place-and-route results for both a reference synchronous implementation (without controllers and delay lines) and for the asynchronous implementation (including the overhead due to synchronous test mode). Table IV compares the two different designs after placement and routing in the UMC 0.18-μm CMOS process.

Both designs have approximately the same area, speed, and power consumption. Differences between them can be attributed more to the different abilities of the two flows to optimize for different objectives (area versus performance, latches versus flip-flops), rather than to the synchronous or asynchronous implementation of each circuit.

If delay line gates are placed away from each other in the floor plan, then the routing delay becomes unpredictable at synthesis time. Hence, we extensively used floor planning in order to control the routing delay.

C. ASPIDA Fabrication

The ASPIDA fabricated chip contains a DLX processor core and two on-chip memories. It supports multiplexed clocking, i.e., the chip can be operated in the fully synchronous mode, or in the desynchronized mode. Clock multiplexing is implemented at the leaf level, i.e., at every single latch. The advantage of multiplexing internally at the leaves is that no changes in the design flow are necessary. This is because the circuit netlist does not change, except for the introduction of local and global latch drivers, and this makes it possible to automatically generate both the global clock trees and the local low-skew buffers independently and automatically. The disadvantage of this approach is the area increase implied by adding a multiplexer to every latch. An alternative approach would be to multiplex clocks externally, which would require more intervention and cause problems with clock generation tools as the global clocks and the local buffer signals would converge outside leaves.

In the synchronous mode (used essentially for scan testing), the M and S latches are driven by two nonoverlapping clocks from two global clock trees. In the desynchronized mode, the signals that open and close the latches are generated by asynchronous handshaking controllers. We used a coarse-grain partitioning for ASPIDA, as each controller drives the M or S latches of one of the four physical pipeline stages, including the processor's register file, which resides internally within the ID pipeline stage.

The outputs of latch controllers must use low-skew buffering, much like a local clock tree. The synchronous and asynchronous modes are controlled by a global input, which multiplexes the enable clock input (g) of every latch.

The delay elements for the ASIC design were implemented with multiple taps in order to control their magnitude after

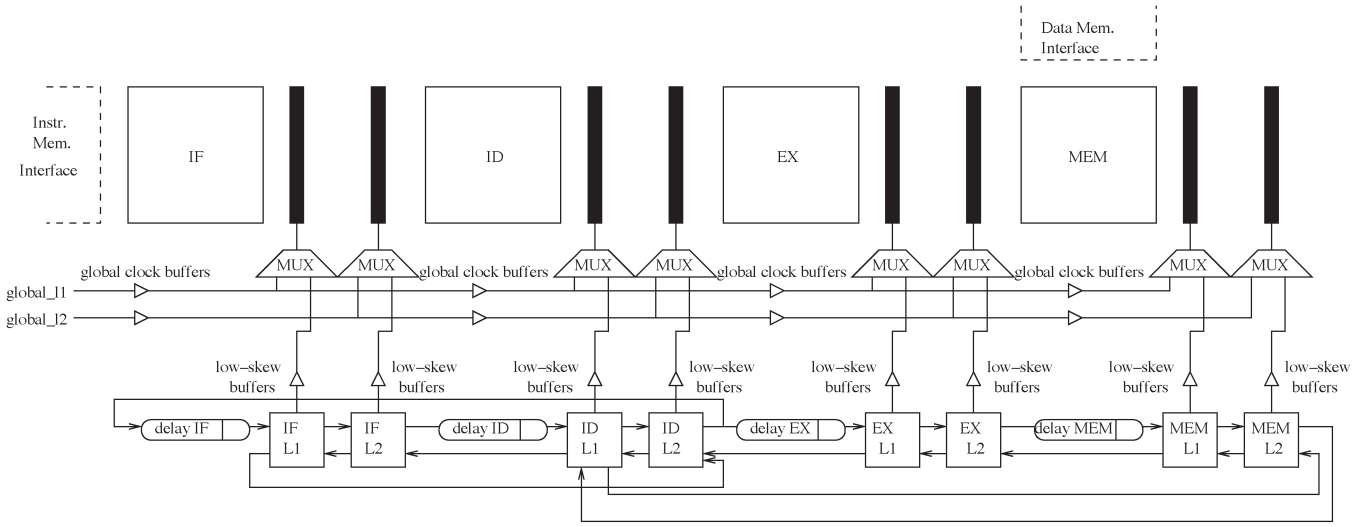


Fig. 16. Desynchronized DLX with multiplexed clocking.

TABLE IV
SYNCHRONOUS VERSUS DESYNCHRONOUS DLX DESIGNS (IN UMC 0.18-μm CMOS)

	Sync. DLX	De-Sync. DLX
Cycle Time	4.4ns	4.45ns
Dyn. Power Cons.	70.9mW	71.2mW
Area	372,656 μm^2	378,058 μm^2

TABLE V
ASPIDA POSTLAYOUT SIMULATION RESULTS (IHP 0.25 μm CMOS)

	Max. Freq.	RMS Power @ 50MHz
Sync.	68.5MHz	182mW (@ 50MHz)
De-Sync.	51.8MHz	200mW (@ 50MHz)

TABLE VI
ASPIDA POSTLAYOUT POWER BREAKDOWN (IN MW)

	Latches (incl. regfile)	Matched Delays	Low-skew Nets	Combinational Logic
Sync.	29 **(16%)**	0 **(0%)**	27 **(15%)**	126 (69%)
De-Sync.	29 **(14%)**	2 **(1%)**	50 **(25%)**	119 (60%)

fabrication. One of the goals of postfabrication tests is to progressively reduce their delay (and the clock period, in synchronous mode) in order to see up to what frequency the design still works. Four taps are implemented, with the longest delay set to 120% of the results of STA using typical gate delays (i.e., we took a 20% margin). Other taps are at 1/2, 1/4, and 1/8 of that delay.

Table V compares simulation results for speed and power for the two modes of operation of the chip. The area of the entire chip, including instruction and data memories, is 13.88 μm^2. As it can be seen from the table, in the synchronous mode, the circuit can reach a higher frequency using the external two phase nonoverlapping clocks by controlling individually the waveform of the two global clocks to the optimal frequency and phase relationship at the current supply voltage and temperature for each individual chip. This is not part of the standard ASIC methodology, which relies on margins and STA to ensure correct operation under any operating conditions. If we relied only on STA, the synchronous circuit would work at about 50 MHz.

Root mean square (rms) power at 50 MHz is slightly higher for the desynchronized operation mode. Table VI shows the power breakdown for both designs at 50 MHz. The total power is divided into the power consumed by the latches (including the register file), matched delays, low-skew clocking nets, and combinational logic. The clocking nets are synchronous mode, the two global clock trees in the synchronous mode, and the low-skew buffer trees of the handshake controllers for the desynchronized mode.

As shown in the table, the desynchronized mode of operation consumes slightly more power due to the higher number of low-skew nets, i.e., eight [two per controller output (Fig. 16)]. This result leads us to believe that using larger groups of latches with a single controller would be beneficial for this design. It also suggests that by incorporating explicit knowledge of the desynchronized design style into clock tree generation would lead to better performance and lower power.

A slight reduction in the consumption of the combinational logic due to the smaller propagation of glitches through the latches can also be observed.

For the desynchronized mode of operation, which adapts to the scaling of the supply voltage, we performed postlayout power measurements at different supply values. As standard-cell libraries and tools do not support scaling of delays for different voltage values, we performed SPICE level simulations on a small number of transistor-level cells in order to estimate and verify the effect of voltage scaling. As expected, for voltage values safely above threshold voltage, gate delay was found to be proportional to the inverse of the supply voltage. As the supply voltage approaches the threshold voltage, gate delay becomes proportional to $1/\mathrm{Vdd}^n$ for n between 2 and 3, while wire delay scales linearly. By introducing appropriate delay scaling factors in the technology for gates and wires, we

TABLE VII
ASPIDA POSTLAYOUT POWER SCALING FOR DESYNCHRONOUS MODE

Voltage	Power	Cycle Time	Energy per Cycle
2.50V	200mW	20.1ns	4.02nJ
2.25V	167mW	22.7ns	3.79nJ
2.10V	152mW	24.1ns	3.66nJ
2.00V	143mW	25.2ns	3.60nJ
1.85V	127mW	27.4ns	3.48nJ
1.50V	98mW	33.8ns	3.31nJ

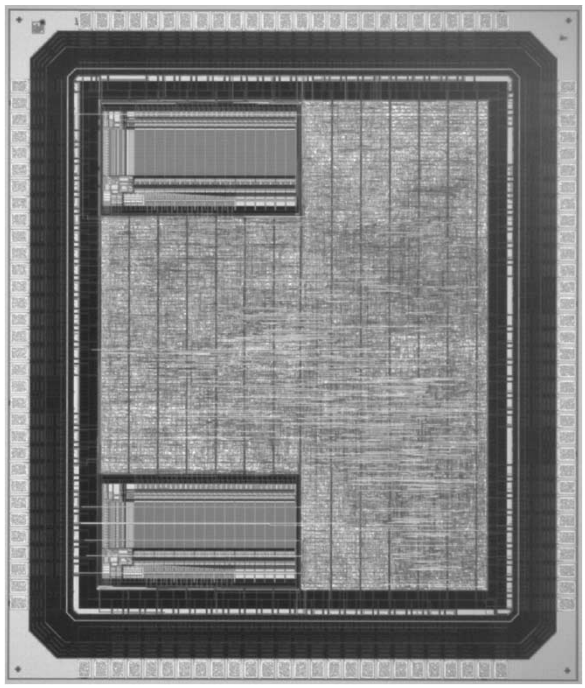


Fig. 17. ASPIDA die photo.

obtained the results shown in Table VII. Thus, the potential power savings by voltage scaling that are made much easier by desynchronization are demonstrated.

The performance difference between synchronous and desynchronized modes stems from the fact that in ASPIDA delay elements were neither optimally tuned nor physically controlled so as to constrain their physical spread or placement location. With ASPIDA being the first desynchronized design being fabricated, the key idea was to demonstrate working silicon and to assess its characteristics with the minimum amount of tuning, both the delay elements and the P&R process. Thus, the synchronous mode showed slightly better performance. The postlayout simulations of a second run of the ASPIDA chip, with more accurately tuned matched delays, showed that the same performance can be obtained for synchronous and desynchronized operation, with the desynchronized version being more robust due to the easier tracking of environment conditions (supply voltage and temperature) ensured by delay chains.

The ASPIDA chip, shown in Fig. 17, was fabricated in early 2005, and postmanufacturing measurements verified the correct operation of the first silicon. Extensive tests over about 90 fabricated chips yielded very interesting results. The desynchronized mode of operation was demonstrated to be an average of 25% faster, over the range of chips, than the worst case synchronous operation predicted by EDA tools. This mismatch demonstrates the inability of existing EDA tools, flows, and technology libraries to accurately predict and characterize silicon performance postmanufacturing, even at 0.25 μm. The current models are simply too pessimistic. With technology scaling in the nanometer era, this problem will get worse.

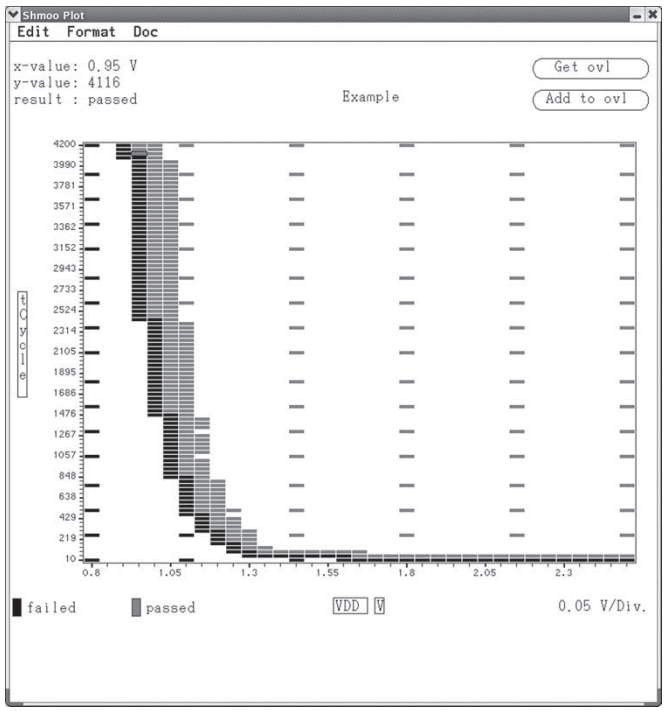


Fig. 18. ASPIDA: schmoo plot in asynchronous operation.

In addition, the desynchronized mode of operation demonstrated excellent coping with variability. This was demonstrated by experiments, where the power supply was varied over an extensive range of voltages, as shown in the schmoo plot of Fig. 18.

Fig. 18 plots voltage on the x-axis. Light gray dots indicate chips that pass the functional test, whereas dark gray dots indicate chips that fail. The plot indicates correct operation in desynchronized mode all the way down to 0.95 V, which is only 0.35 V away from the threshold voltage of the process, which is 0.6 V. This is strong evidence that the desynchronization approach handles variability very well, as at the very low operational voltage of 0.95 V all second- and third-order phenomena of transistor behavior are in full effect.

In desynchronized operation, both the processor speed and voltage can be controlled using a single variable, i.e., supply voltage, whereas in the synchronous mode, two variables must be controlled externally, in a tightly coordinated manner, i.e., both voltage and frequency. ASPIDA has demonstrated the effective single-variable control that desynchronized operation allows. In addition, tests over a range of 90 chips demonstrated, as shown in Fig. 19, that the desynchronized circuit operates much more efficiently when the voltage is varied, compared to its synchronous mode of operation. This is due to the self-adapting nature of the desynchronized design, whereas in synchronous mode, the external clocks must be adjusted according to the capabilities of the external circuits and extensive experimentation.

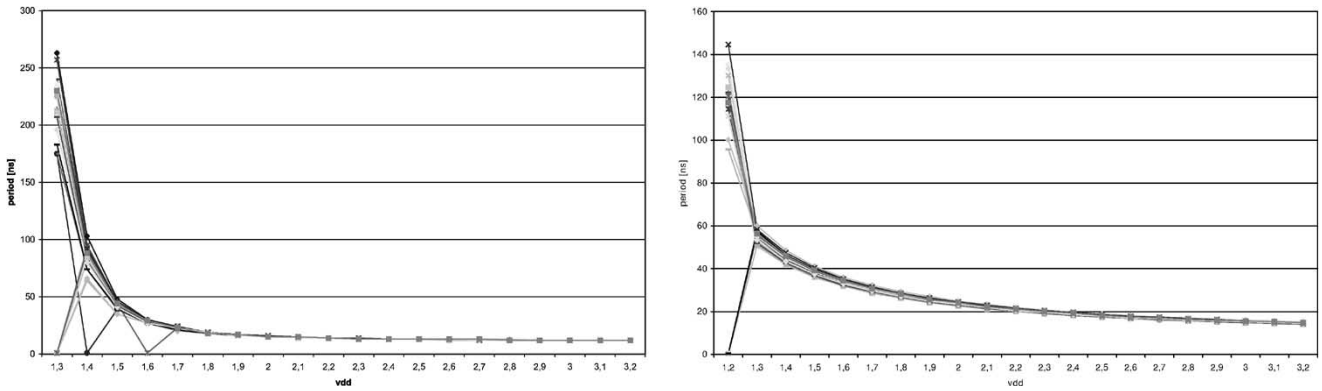


Fig. 19. ASPIDA: period in synchronous (left) and desynchronized (right) operation versus supply voltage Vdd.

X. Conclusion

This paper presented a desynchronization design flow that can be used to automatically substitute the clock network of a synchronous circuit by a set of asynchronous controllers.

To the best of our knowledge, this is the first successful attempt of delivering an automated design flow for asynchronous circuits that does not introduce significant penalties with respect to the corresponding synchronous designs. This opens wide opportunities of exploring the implementation space (both synchronous and asynchronous), by using the very same set of industrial tools. This, we believe, is a valuable feature for a designer.

The suggested methodology can result in easier silicon-on-a-chip (SOC) integration and shorter design cycles. Due to the partitioning of the clock trees, it also offers lower power and possibly lower electromagnetic emission, which is important both to reduce the cost of packaging, to increase security, and to integrate analogue circuitry on-chip. Moreover, it provides the foundation for achieving power savings by tolerating broader performance variations. Early fabrication results confirm simulation results and increase our confidence in the widespread applicability of desynchronization to real ASIC designs.

However, the true advantage of asynchronous implementation cannot be achieved unless one measures the true delay of combinational logic, rather than estimate it by using delay lines that still require margins in order to ensure slower propagation than the longest logic path. This is left to future work, even though preliminary results (e.g., [12]) are quite promising.

References

[1] C. Amin, N. Menezes, K. Killpack, F. Dartu, U. Choudhury, N. Hakim, and Y. Ismail, "Statistical static timing analysis: How simple can we get?" in *Proc. IEEE Design Automation Conf.*, Anaheim, CA, 2005, pp. 652–657.

[2] A. Bardsley and D. Edwards, "Compiling the language Balsa to delay-insensitive hardware," in *Computer Hardware Description Languages and Applications (CHDL)*, C. D. Kloos and E. Cerny, Eds., Toledo, Spain, Apr. 1997, pp. 89–91.

[3] A. Benveniste, B. Caillaud, and P. Le Guernic, "From synchrony to asynchrony," in *Concurrency Theory, 10th Int. Conf. (CONCUR)*, J. Baeten and S. Mauw, Eds. London, U.K.: Springer-Verlag, Aug. 1999, vol. 1664, pp. 162–177.

[4] A. Benveniste, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli, "Heterogeneous reactive systems modeling and correct-by-construction deployment," in *Embedded Software, 3rd Int. Conf., (EMSOFT)*, R. Alur and I. Lee, Eds., Berlin, Germany, Oct. 2003, vol. 2855, pp. 35–50.

[5] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, vol. 5. New York: Cambridge Univ. Press, 1993.

[6] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits Systems*, Crete, Greece, 2004, pp. 149–158.

[7] I. Blunno and L. Lavagno, "Automated synthesis of micro-pipelines from behavioral Verilog HDL," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits Systems*, Eilat, Israel, Apr. 2000, pp. 84–92.

[8] D. Chinnery and K. Keutzer, "Reducing the timing overhead," in *Closing the Gap Between ASIC and Custom: Tools and Techniques for High-Performance ASIC Design*. Norwell, MA: Kluwer, 2002, ch. 3.

[9] F. Commoner, A. W. Holt, S. Even, and A. Pnueli, "Marked directed graphs," *J. Comput. Syst. Sci.*, vol. 5, no. 5, pp. 511–523, Oct. 1971.

[10] J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "From synchronous to asynchronous: An automatic approach," in *Proc. DATE*, Paris, France, 2004, vol. 2, pp. 1368–1369.

[11] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "A concurrent model for de-synchronization," in *Proc. Int. Workshop Logic Synthesis*, Laguna Beach, CA, 2003, pp. 294–301.

[12] ——, "Coping with the variability of combinational logic delays," in *Proc. IEEE Int. Conf. Computer Design*, San Jose, CA, Oct. 2004, pp. 505–508.

[13] A. Davare, K. Lwin, A. Kondratyev, and A. L. Sangiovanni-Vincentelli, "The best of both worlds: The efficient asynchronous implementation of synchronous specifications," in *Proc. IEEE Design Automation Conf.*, San Diego, CA, 2004, pp. 588–591.

[14] L. Dennison, W. Dally, and T. Xanthopoulos, "Low-latency plesiochronous data retiming," in *Advanced Research VLSI*, Chapel Hill, NC, 1995, pp. 304–315.

[15] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 247–253, Jun. 1996.

[16] S. B. Furber, J. D. Garside, and D. A. Gilbert, "AMULET3: A high-performance self-timed ARM microprocessor," in *Proc. ICCD*, Austin, TX, Oct. 1998, pp. 247–252.

[17] P. L. Guernic, J.-P. Talpin, and J.-C. L. Lann, "Polychrony for system design," *J. Circuits Syst. Comput.*, vol. 12, no. 3, pp. 261–304, Apr. 2003.

[18] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Norwell, MA: Kluwer, 1993.

[19] J. L. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1990.

[20] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim, "Clock synchronization through handshake signalling," *Microprocess. Microsyst.*, vol. 27, no. 9, pp. 447–460, Oct. 2003.

[21] R. Kol and R. Ginosar, "A doubly-latched asynchronous pipeline," in *Int. ICCD*, Austin, TX, Oct. 1996, pp. 706–711.

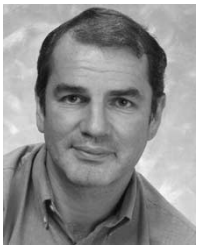
[22] A. Kondratyev, L. Sorenson, and A. Streich, "Testing of asynchronous designs by inappropriate means: Synchronous approach," in *Proc. IEEE Int. Symp. Advanced Research Asynchronous Circuits Systems*, Manchester, U.K., Mar. 2001, pp. 171–180.

[23] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev, "Asynchronous design using commercial HDL synthesis tools," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits Systems*, Eilat, Israel, Apr. 2000, pp. 114–125.
[24] D. H. Linder and J. C. Harden, "Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1031–1044, Sep. 1996.
[25] T. Murata, "Petri Nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[26] S. Nassif, D. Boning, and N. Hakim, "The care and feeding of your statistical static timer," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2004, pp. 138–139.
[27] C. D. Nielsen and M. Kishinevsky, "Performance analysis based on timing simulation," in *Proc. ACM/IEEE Design Automation Conf.*, San Diego, CA, Jun. 1994, pp. 70–76.
[28] O. A. Petlin and S. B. Furber, "Scan testing of micropipelines," in *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, May 1995, pp. 296–301.
[29] R. B. Reese and M. A. T. C. Traver, "A coarse-grain phased logic CPU," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits Systems*, Vancouver, Canada, May 2003, pp. 2–13.
[30] V. Varshavsky, V. Marakhovsky, and T.-A. Chu, "Logical timing (global synchronization of asynchronous arrays)," in *1st Int. Symp. Parallel Algorithm/Architecture Synthesis*, Aizu-Wakamatsu, Japan, Mar. 1995, pp. 130–138.

Jordi Cortadella (S'87–M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politécnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is a Professor in the Department of Software of the same university. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include formal methods and computer-aided design of very large scale integration (VLSI) systems with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis. He has co-authored numerous research papers and has been invited to present tutorials at various conferences.

Dr. Cortadella has served on the technical committees of several international conferences in the field of design automation and concurrent systems. He received Best Paper Awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems (2004) and the Design Automation Conference (2004). In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya.



Alex Kondratyev (M'94–SM'97) received the M.S. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia, in 1983 and 1987, respectively.

From 1993 to 1999, he was an Associate Professor of the Hardware Department, University of Aizu, Aizu-Wakamatsu, Fukushima, Japan. In 2000, he joined Theseus Logic as a Senior Scientist. Since 2001, he has been working as a Research Scientist in Cadence Berkeley Laboratories, Berkeley, CA. He has published over 90 journal and conference papers. His research interests include formal methods in system design, synthesis of asynchronous circuits, and computer-aided design methodology.

Dr. Kondratyev was a Co-Chair of the Async'96 Symposium and the CSD'98 Conference and has served as a member of program committees for several conferences.

Luciano Lavagno (S'88–M'93) received the B.S. degree (*magna cum laude*) in electrical engineering from Politecnico di Torino, Torino, Italy, in 1983, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1992.

From 1984 to 1988, he was with CSELT Laboratories, Torino, Italy. In 1988, he joined the Department of Electrical Engineering and Computer Science, University of California, Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. He is a coauthor of two books on asynchronous circuit design and a book on hardware/software co-design of embedded systems and has published over 100 journal and conference papers. Between 1993 and 1998, he was an Assistant Professor at Politecnico di Torino, and between 1998 and 2001, he was an Associate Professor at the University of Udine. Between 1993 and 2000, he was the architect of the POLIS project (a cooperation between University of California at Berkeley, Cadence Design Systems, Magneti Marelli, and Politecnico di Torino), developing a complete hardware/software co-design environment for control-dominated embedded systems. He is currently an Associate Professor at Politecnico di Torino and a Research Scientist at Cadence Berkeley Laboratories, Berkeley, CA. His research interests include the synthesis of asynchronous and low-power circuits, the concurrent design of mixed hardware and software embedded systems, and dynamically reconfigurable processors.

Dr. Lavagno received the Best Paper Award at the 28th Design Automation Conference, San Francisco, CA, in 1991. He has served on the technical committees of several international conferences in his field (e.g., the Design Automation Conference, the International Conference on Computer-Aided Design, the International Conference on Computer Design, and Design Automation and Test in Europe) and of various other workshops and symposia.

Christos P. Sotiriou (M'00) received the Ph.D. degree in computer science from the University of Edinburgh, Edinburgh, U.K., in 2001.

He is a Research Associate with the Institute of Computer Science (ICS) at the Foundation for Research and Technology-Hellas (FORTH), Crete, Greece, and a Visiting Assistant Professor at the University of Crete. His research interests include very large scale integration (VLSI) design, computer-aided design (CAD) algorithms and tools, and design methods and approaches for asynchronous timing.

Dr. Sotiriou served as the General Chair of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, where he received the Best Paper Award, in 2004.