POLITECNICO DI TORINO
Repository ISTITUZIONALE

Desynchronization: Synthesis of asynchronous circuits from synchronous specifications

Original
Desynchronization: Synthesis of asynchronous circuits from synchronous specifications / Jordi, C., Alex, K., Lavagno, L., Christos, S.. - In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. - ISSN 0278-0070. - 25:(2006), pp. 1904-1921.

Availability:
This version is available at: 11583/1643397 since:

Publisher:

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2026

De-synchronization: synthesis of asynchronous circuits from synchronous specifications

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

***Abstract*— Asynchronous implementation techniques, which measure logic delays at run time 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. *De-synchronization* 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, we first of all study different protocols for de-synchronization and formally prove their correctness, using techniques originally developed for distributed deployment of synchronous language specifications. We also provide a taxonomy of existing protocols for asynchronous latch controllers, covering in particular the four-phase handshake protocols devised in the literature for micro-pipelines. We then propose a new controller which exhibits provably maximal concurrency, and analyze the performance of desynchronized circuits with respect to the original synchronous optimized implementation. We finally prove the feasibility and effectiveness of our approach, by showing its application to a set of real designs, including a complete implementation of the DLX microprocessor architecture.**

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

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 feed-forward 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*.

This approach has worked very well in the past, but currently it shows several signs of weakness. A designer, helped by classical 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], [30]) partially deals with the problem, by separating *uncorrelated* variations, whose effect is reduced because they quickly average out, and *correlated* variations, that 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 non-linearities 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

This work was supported by the Working Group on Asynchronous Circuit Design (ACID-WG, IST-1999-29119), the ASPIDA project (IST-2002-37796) and CICYT TIN2004-07925.

J. Cortadella is with the Software Department, Universitat Politècnica de Catalunya, Barcelona, Spain (email: jordi.cortadella@upc.edu).

A. Kondratyev is with Cadence Berkeley Labs, Berkeley, USA (email:kalex@cadence.com).

L. Lavagno is with Politecnico di Torino, Italy (email:lavagno@polito.it).

C. Sotiriou is with ICS-FORTH, Crete, Greece (email:sotiriou@ics.forth.gr).

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 multi-tasking 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 sec to process 4 or 5 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:

- There are no good CAD tools that completely cover the design flow.
- 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 ASIC design only when it deals with the clock tree at the logical level. I.e. specification using synthesizable Hardware Description Languages, 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 [14].

A. De-synchronization

De-synchronization 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 of all, the notion of *operation cycle* lives in the subconscious of most circuit designers. Finite state machines, pipelined microprocessors, multi-cycle 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 *de-synchronization*. 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: 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 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 de-synchronization is also presented.

We validated our approach by comparing synchronous and de-synchronized designs of large examples, including an implementation of the DES encryption standard and one of the

DLX microprocessor [22], 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 [19] and CAD tools [2], [5], [8]. 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 an LEDR delay-insensitive code [28]. 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 [33], but no direct apples-to-apples comparison with a synchronous design was presented there.

Similarly, Theseus Logic proposed a design flow [27] which 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 non-standard 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 [24], 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 RTL specification.

The generation of local clocks from handshaking circuitry while ensuring the global "synchronicity" was first suggested in [36]. 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 [25]. Our paper extends the results from [25], [36] by using more general synchronization schemes and provides a theoretical foundation for the de-synchronization approach, by proving a behavioral and temporal equivalence between a synchronous circuit and its de-synchronized counterpart.

We also extend with respect to our own previous work in [12], [13] 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 [21] 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 couple 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 prove formally 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*, that 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*, that 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

Marked Graphs (MG) is the formalism used in this paper to model de-synchronization. They are a subclass of Petri nets [29] that can model decision-free concurrent systems.

Definition 3.1 (Marked graph): A *marked graph* 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 marked graph.

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 exist σ such that $M \xrightarrow{\sigma} M'$. The set of reachable markings from M_0 is denoted by $[M_0\rangle$.

An example of marked graph is shown in Figure 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: $B+$ and $D+$. The sequence of events $\langle D+\ D-\ C+\ B+\ B-\ A+\ C-\rangle$ is an example of a feasible sequence of the marked graph.

Definition 3.2 (Liveness): A marked graph 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): A marked graph 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 [11] for *strongly connected* marked graphs.

Theorem 3.1 (Liveness): A marked graph is live *iff* 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): A marked graph is safe *iff* 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 marked graphs.

IV. A ZERO-DELAY DE-SYNCHRONIZATION MODEL

The de-synchronization 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 Figure 1(a)).

A. Steps in the de-synchronization method

The de-synchronization method proceeds in three steps:

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

   The conversion of a flip-flop-based circuit into a latch-based one is not specific to the de-synchronization framework only. It is known to give an improvement in performance for synchronous systems [9] and, for this reason, it has a value by itself.
2) *Generation of matched delays for the combinational logic* (denoted by rounded rectangles in Figure 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.

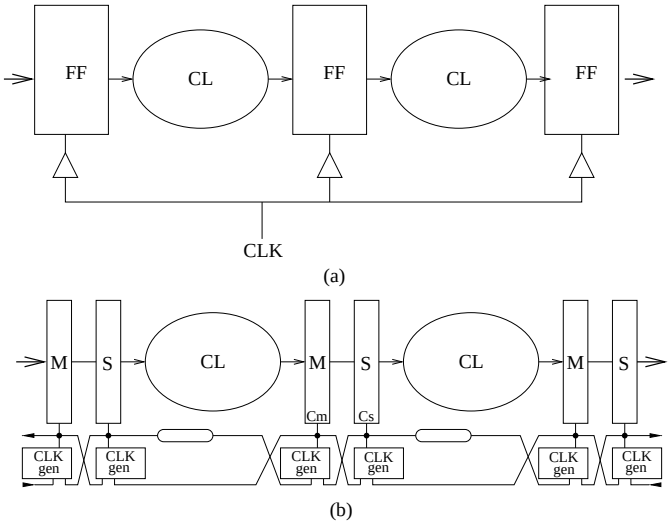


Fig. 1. (a) Synchronous circuit, (b) de-synchronized circuit.

Figure 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) latches and vice-versa. Usually, this latch-based scheme is implemented with two non-overlapping 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. The zero-delay model

This section presents a formal model for de-synchronization. The aim is to produce a set of distributed controllers that

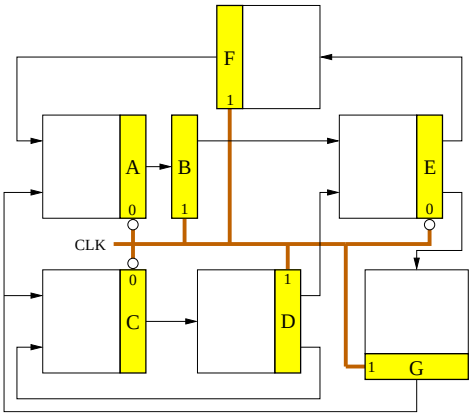


Fig. 2. A synchronous circuit with a single global clock.

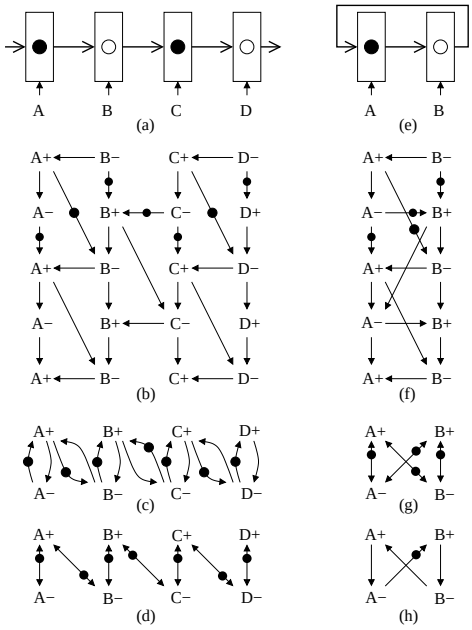


Fig. 3. De-synchronization model for a linear pipeline and a ring.

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 data-path 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 Figure 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.

Figure 3(b) depicts a fragment of the unfolded marked graph 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):

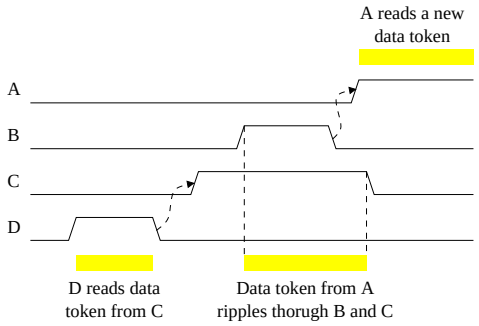


Fig. 4. Timing diagram of the linear pipeline in Figure 3(a-d).

- $A+ \rightarrow A- \rightarrow A+$, that simply denote that the rising and falling transitions of each signal must alternate.
- $B- \rightarrow A+$, that 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 terms *hold constraints can be violated*.
- $A+ \rightarrow B-$, that 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 terms *setup constraints can be violated*.

The marking in Figure 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.

Figure 3(c) shows the marked graph that derives from the unfolded graph in Figure 3(b). A simplified notation is used in Figure 3(d) to represent the same graph, substituting each cycle $x \rightleftarrows y$ by a double arc $x \longleftrightarrow y$, 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 non-overlapping 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 Figure 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* [20].

C. General de-synchronization model

The general de-synchronization model is shown in Figure 5. For each communication between an even latch and an odd latch, the synchronization depicted in Figure 5(a) must be defined. If the communication is between odd and even,

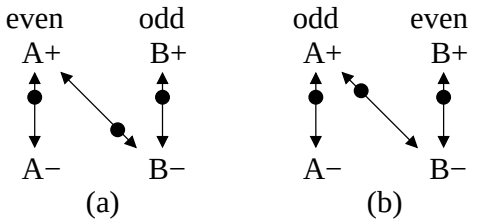


Fig. 5. Synchronization between latches: $A \rightarrow B$.

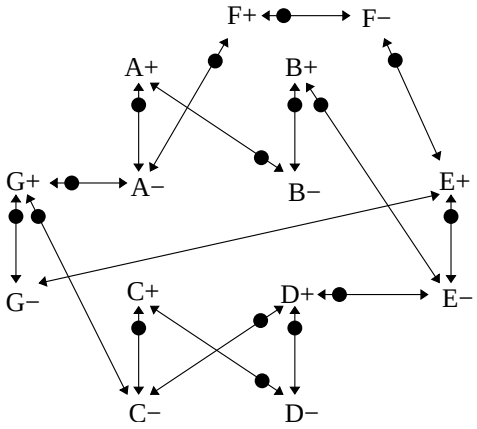


Fig. 6. De-synchronization model for the circuit in Figure 2.

the one in Figure 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 Figure 6. The marked graphs obtained by properly abutting the models in Figure 5 are called *circuit marked graphs* (CMG).

We now show that a de-synchronized circuit mimics the behavior of its synchronous counterpart. For that, it must be proved that:

- a de-synchronized circuit never halts (*liveness*), and
- all computations performed by a de-synchronized circuit are the same as the ones performed by the synchronous counterpart (*flow-equivalence*).

The remaining 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 $x \longleftrightarrow y$, that represent $x \underset{\longleftarrow}{\overset{\bullet}{\rightarrow}} y$.

Theorem 4.1: Any circuit marked graph 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 3 arcs. As an example, let us try to find the longest unmarked path from $D+$ in the CMG of Figure 3(c). After building the path $D+ \rightarrow D- \rightarrow C+ \rightarrow C-$, 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. ■

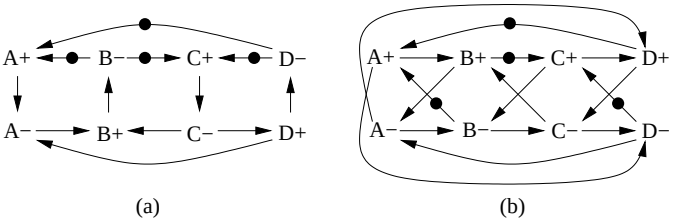


Fig. 7. Synchronization of a ring: (a) live model, (b) non-live model.

Liveness guarantees something crucial for the model: absence of deadlocks. This property does not hold automatically for every "reasonable" model. Figure 7 depicts two different de-synchronization models for a ring, that can be obtained by connecting the output of latch D with the input of latch A in Figure 3(a). Figure 7(a) depicts a non-overlapping model between adjacent latches, whereas Figure 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 Figure 3(e)). Figure 3(f) depicts the unfolded behavior after including all causality constraints for the communication $A \rightarrow B$ and $B \rightarrow A$. The folded behavior is shown in Figure 3(g), that can also be obtained by combining the synchronization models of Figure 5(a) and 5(b). Several arcs become redundant, thus deriving the simplified model shown in Figure 5(h).

Interestingly, the resulting protocol derived from the "aggressive" concurrent model is "naturally" transformed into one that is *non-overlapping, 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 Figure 2 (see Figure 6 also) becomes non-overlapping.

E. Flow-equivalence

In this section we prove that a de-synchronized circuit mimics its synchronous counterpart. We show that, for each latch, the value stored at the i-th pulse of the control signal is the same as the value stored at the i-th 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 calculated by the combinational logic. We call A_i the value stored in A's latch after the i-th clock cycle. Let us call $E^1 \dots E^p$ the (even) predecessor latches of an odd latch O, and $O^1 \dots O^p$ the (odd) predecessor latches of an even latch E. Then,

- $O_i = F_O(E^1_{i-1}, \dots, E^p_{i-1})$, and
- $E_i = F_E(O^1_i, \dots, O^p_i)$

where all even blocks store a known initial value at cycle 0.

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 *non-deterministic function*. The latter mechanism also allows us to show how a de-synchronized 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 $E^1 \dots E^n$ and $O^1 \dots O^m$ 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 values:

cycle	*clk*	trace
initial	0	$E^1_0 \dots E^n_0 \quad O^1_0 \dots O^m_0$
1	1	$E^1_0 \dots E^n_0 \quad O^1_1 \dots O^m_1$
	0	$E^1_1 \dots E^n_1 \quad O^1_1 \dots O^m_1$
2	1	$E^1_1 \dots E^n_1 \quad O^1_2 \dots O^m_2$
$\vdots$	$\vdots$	$\vdots$
i	1	$E^1_{i-1} \dots E^n_{i-1} \quad O^1_i \dots O^m_i$
	0	$E^1_i \dots E^n_i \quad O^1_i \dots O^m_i$
$i+1$	1	$E^1_i \dots E^n_i \quad O^1_{i+1} \dots O^m_{i+1}$

If we project the trace onto one of the latches, say A, we obtain a trace $A_0 A_1 \dots A_i \dots$, 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. ■

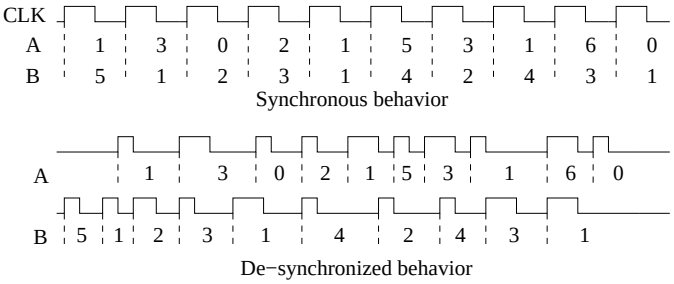


Fig. 8. Flow equivalence.

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* [20], which is related to that of *synchronous behavior* in [28], 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 and
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 Figure 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 de-synchronization. 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 5 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 de-synchronization 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 i-th occurrence of $A-$ enabled (see Figure 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 (Figure 9(a)). For even latches (see Figure 9(b)), it holds after having fired $O^1+ \dots O^p+$

[1]A similar result was derived in [28] also based on Marked Graph Theory, using however a very different circuit structure and implementation philosophy.

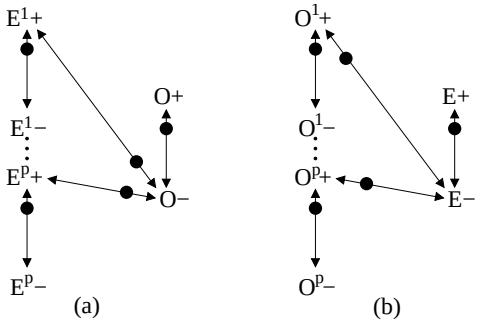


Fig. 9. Illustration of Theorem 4.2.

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

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

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 i-th 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

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

which preserves flow-equivalence. Moreover, the i-th 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:

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

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

■

V. Handshake protocols for de-synchronization

Section IV presented a model for de-synchronization that defines the causality relations among the latch control signals for a correct flow of data in the data-path. 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 question is: are they suitable for a fully automatic de-synchronization approach? Is there any controller that manifests the concurrency of the de-synchronization model proposed in this paper?

We now review the classical four-phase micropipeline latch control circuits presented in [18]. For that, the specification of each controller (figures 5, 7 and 11 in [18]) 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].

Figures 10(a-c) show the projections of the controllers from [18]. 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 [18] show less concurrency than the de-synchronization model. For this reason, we also propose a new controller implementing the protocol with maximum concurrency proposed in this paper (Figure 10(e)). For completeness, a handshake decoupling the falling events of the control signals (*fall-decoupled*) is also described in Figure 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 de-synchronization? Instead of studying them one by one, we present a general study of four-phase protocols, illustrated in Figure 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 marked graphs 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 (5 states) can be obtained from the rise-decoupled protocol (6 states) by changing the arc[4] $A+ \bullet\!\!\rightarrow B-$ to the arc $A+ \bullet\!\!\rightarrow B+$, thus reducing concurrency.

The model with 8 states, labeled as "de-synchronization model", corresponds to the most concurrent model presented earlier in this paper, for which liveness and flow-equivalence have been proved 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 [18].

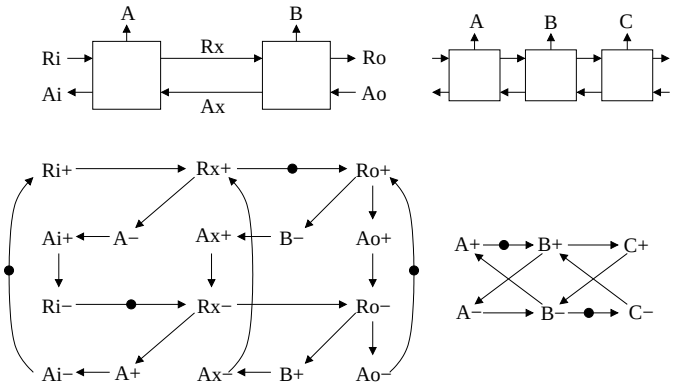
In [7], the following results were proved for the models shown in Fig 11:

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

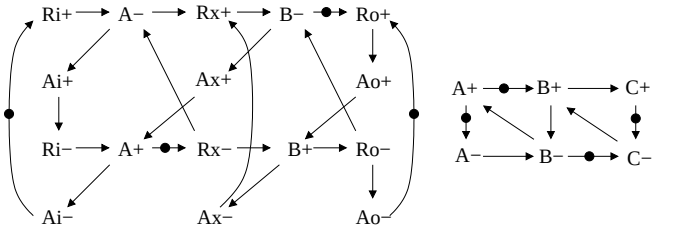
[2]In fact, A is the signal preceding the buffer that feeds the latch control signal. The polarity of the signal has been changed to make the latch transparent when A is high.

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

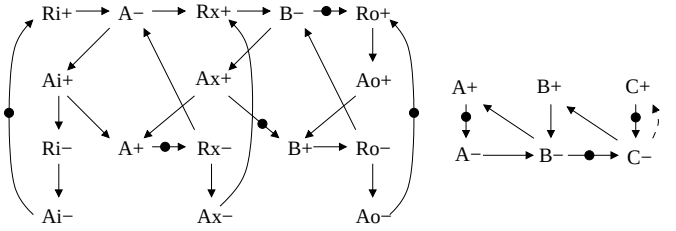
[4]Note that this arc is not explicitly drawn in the picture because it is redundant.



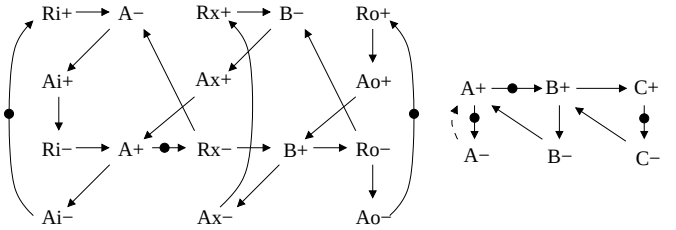
(a) Simple four−phase control (Furber & Day)



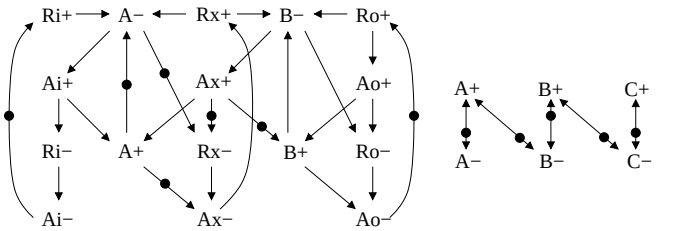
(b) Semi−decoupled four−phase control (Furber & Day)



(c) Fully decoupled four−phase control (Furber & Day)



(d) Fall−decoupled control



(e) De−synchronization control (this paper)

Fig. 10. Handshake protocols.

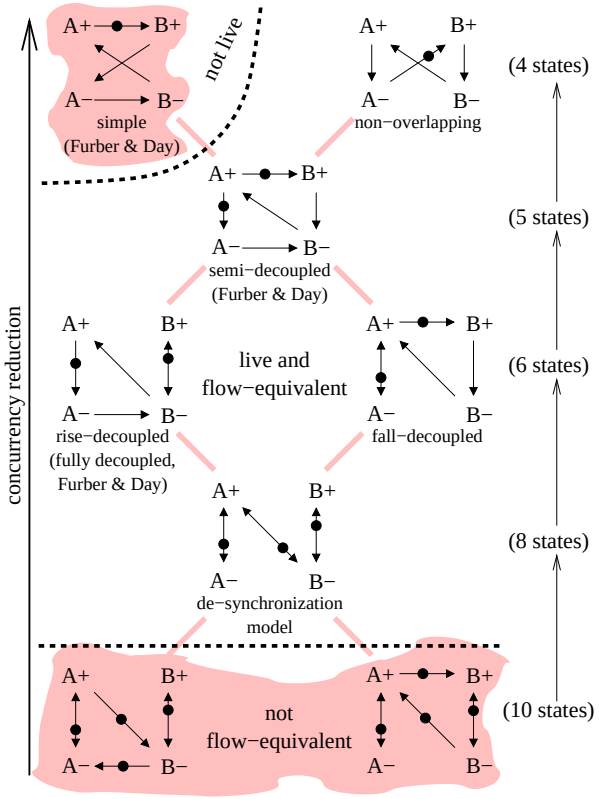


Fig. 11. Different degrees of concurrency in handshake protocols for de-synchronization.

- De-synchronization 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 de-synchronized circuits.

VI. Implementation of De-synchronization 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 [18]. We present an implementation with static CMOS gates, while the original one was designed at transistor level. The reasons for the selection of this protocol with this particular design style are several:

- We pursue an approach suitable for semi-custom design using automatic physical layout tools.
- The semi-decoupled protocol is a good trade-off between simplicity and performance.
- The pulse width of the latch control signals will be similar if all controllers are similar. Moreover, the depth of the data-path 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.

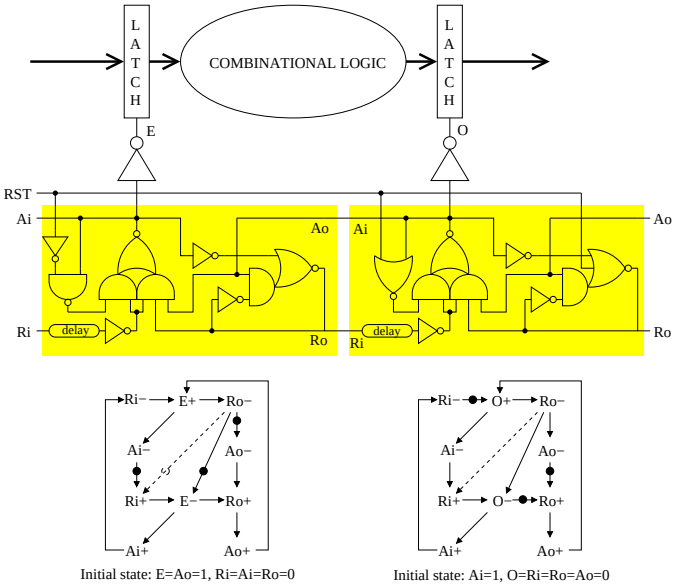


Fig. 12. Implementation of semi-decoupled controllers for even (E) and odd (O) latches.

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

Figure 12 depicts an implementation of a pair of controllers (even and odd) for a fragment of data-path. The figure also shows the marked graphs modeling the behavior of each controller. The only difference is the initial marking, that 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 data-path. 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 pulse width 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

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

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

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

[5]This assumption also allows us to simplify the implementation proposed in [18]: the equation for $A+$ becomes R_{in} instead of $R_{in} \wedge \neg R_{out}$.

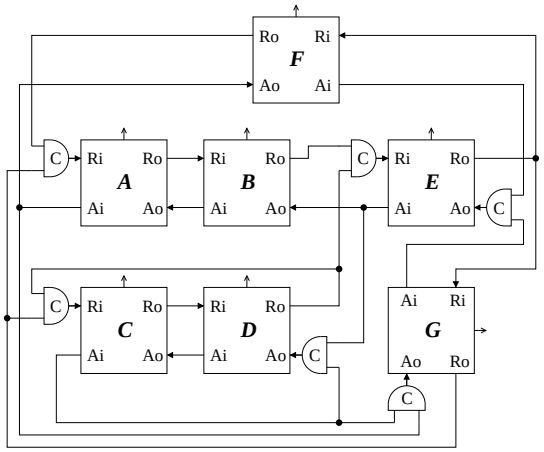


Fig. 13. De-synchronized control for the netlist in Figure 2.

is the one that must be matched with the minimum pulse width. 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, Figure 13 shows the de-synchronization control for the circuit depicted in Figure 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.

Figure 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 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 [9].

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

$$T_T \geq T_{CQ} + T_C + T_L \quad (1)$$

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

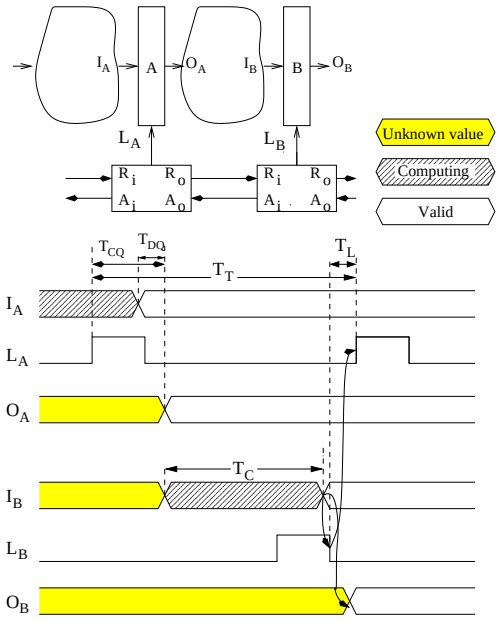


Fig. 14. Timing constraints for the asynchronous controllers.

that the depth of combinational logic is sufficiently large to amortize the overlapping part of the handshake. The latter is true for ASIC designs, that typically have more than 20 levels of logic between adjacent registers.

Inequality (1) guarantees the satisfaction of set-up constraints for the latch. Note that hold constraints in a de-synchronized circuit are ensured automatically, because for any valid protocol (see Figure 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 de-synchronized 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 de-synchronized circuit as well. Based on these two results (temporal and behavioral equivalence) one could replace any synchronous circuit by its de-synchronized 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. 1.2-2X faster than its synchronous counterpart.

In a synchronous flip-flop-based circuit, the cycle time T_S is bounded by [9]:

$$T_S \geq T_C + T_{setup} + T_{skew} + T_{CQ} \qquad (2)$$

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 de-synchronized circuit can only be reduced, one can conclude the difference between the cycle time of de-synchronized circuit T_T and the cycle time T_S of the corresponding synchronous design is approximately $T_L - (T_{setup} + T_{skew})$. 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 de-synchronized system the separation time between adjacent rising edges of the same local clock might change during operation (see Figure 8 e.g.). Therefore we should compare the perfect periodic behavior of the synchronous circuit with the non-periodic one of the de-synchronized 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 de-synchronized system have a well-defined constant cycle time, while the rest of the latches operate in *plesiochronous* mode [16], 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 de-synchronized circuit the computation delay T_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_T.

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

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

Property 7.2: In any de-synchronized circuit the i-th rising transition of a local clock of an odd latch cannot appear later than $(i-1) * T_T$.

Proof: Analysis of the firing time of the i-th instance A_i of event A in a marked graph G is reduced to the following procedure [31]: (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 de-synchronized circuit, the length of the longest path to the i-th rising event at any odd latch is $(i-1) * T_T$. 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 i-th rising

transition happening later than $(i-1) * T_T$. ■

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_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_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_T (after one cycle they are paced by a critical latch). Pushing these arguments further implies that in a connected de-synchronized system, any latch sooner or later settles to the cycle time T_T. This shows that the behavior of a de-synchronized circuit has a well-defined periodicity, similar to that of a synchronous one, paced by a common clock.

Embedding of a de-synchronized circuit with clock cycle T_T into a synchronous environment with a clock cycle $T_S : T_S \geq T_T$ results in the latches at the asynchronous/synchronous boundary becoming critical, since they are paced by a slower external clock T_S.

This consideration shows that de-synchronized 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 set-up times are satisfied. This can no longer be done during logic synthesis in the case of ASICs implemented using deep sub-micron 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 re-computed. In this work we take the opposite approach: we propose to use very pessimistic delay models for pre-layout 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.

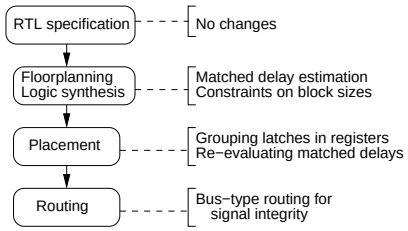


Fig. 15. Changes in the standard synchronous design flow.

Controllers are generated for multi-bit 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 [15].

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.

Matched Delay Insertion. The flow that we used for the de-synchronization approach begins with a synthesizable HDL specification (e.g. Verilog/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 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 de-synchronized 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 post-layout delays are extracted. The pessimistic delays used for pre-layout 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 de-synchronization are shown in Figure 15.

Verification. Conventional equivalence checking tools can be used for the verification of the datapath, since de-synchronization 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.

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 [32]. 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 [26] showed that 100% stuck-at coverage can be achieved for asynchronous pipelines using conventional test pattern generation tools.

Binning (or Speed-Grading) and Yield Improvement. One advantage of de-synchronization 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 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 de-synchronization to two large realistic circuits, a DES core and a DLX microprocessor. The DES core was designed using a 0.18μm standard-cell library from UMC. The DLX core was designed (i) with the same UMC library and (ii) with a 0.25μm 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.

	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 I
Synchronous vs. De-Synchronized DES Case Study

	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%

TABLE II
De-Synchronized DES: Area Breakdown

We first implemented a synchronous, edge-triggered flip-flop design for the 16-stage DES design in the 0.18μm 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 de-synchronization in order to derive a de-synchronized dual-latch design. The type of controllers used in this design are based on the de-synchronization model, *i.e.* with the maximum possible concurrency. Table I contrasts the characteristics of the two designs. All data are post-synthesis, pre-layout 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 sixteen 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 de-synchronized 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 II shows the area breakdown of the de-synchronized 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 218560μm^2.

This example shows that, even for a design containing a very limited amount of combinational logic, de-synchronization 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 de-synchronized version of the DLX processor [22], called ASPIDA (ASynchronous open-source Processor Ip of the Dlx Architecture),

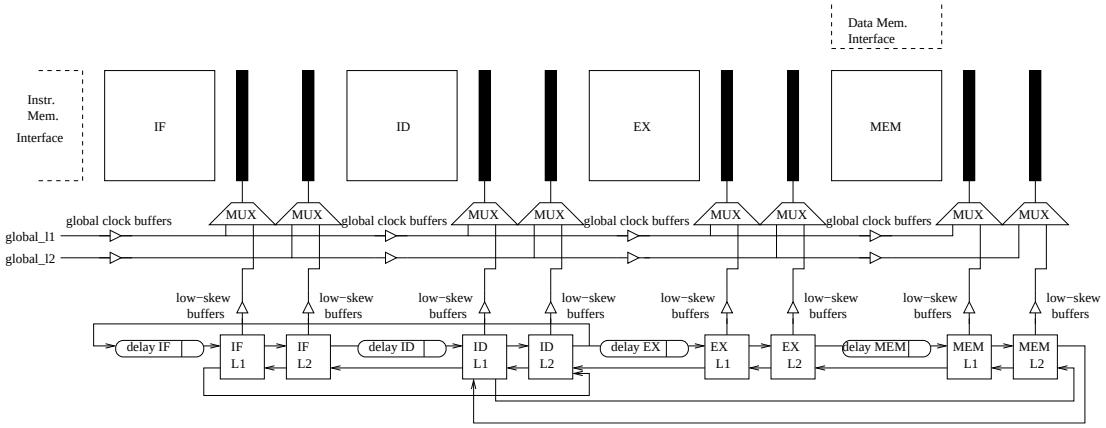


Fig. 16. De-synchronized DLX with Multiplexed-Clocking

	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 III
SYNCHRONOUS VS. DE-SYNCHRONIZED DLX DESIGNS (IN UMC $0.18\mu m$ CMOS)

designed using the semi-decoupled controllers depicted in Figure 12. Figure 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 III 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 III compares the two different designs *after placement and routing* in the UMC 0.18μm 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 vs. performance, latches vs. 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 2 on-chip memories. It supports multiplexed clocking, *i.e.* the chip can be operated in fully-synchronous mode, or in de-synchronized mode. Clock multiplexing is implemented at the leaf level, i.e. at every single latch. The advantage of multiplexing internally at 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; that 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 synchronous mode (used essentially for scan testing) the M and S latches are driven by two non-overlapping clocks from two global clock trees. In de-synchronized 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

fabrication. One of the goals of post-fabrication 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 Static Timing Analysis 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 IV 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 $13.88\mu m^2$. As it can be seen from the table, in synchronous mode the circuit can reach a higher frequency using the external two phase non-overlapping 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 Statical Timing Analysis to ensure correct operation under any operating conditions. If we relied only on STA, the synchronous circuit would work at about 50MHz.

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

TABLE IV
ASPIDA POST-LAYOUT SIMULATION RESULTS (IHP 0.25 μm CMOS)

RMS power at 50MHz is slightly higher for the de-synchronized operation mode. Table V shows the power break-down for both designs at 50MHz. Total power is divided into power consumed by the latches, by the delay elements and by the low-skew clocking nets. The latter are: in synchronous mode, the two global clock trees; in de-synchronized mode, the low-skew buffer trees of the controllers. As can be seen from the table, the de-synchronized mode of operation consumes slightly more power due to the higher number of low-skew nets, *i.e.* eight (two per controller output, Figure 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.

	Latches	**Delay Elements**	**Low-skew Nets**
Sync.	28.81mW **(15.82%)**	0mW **(0%)**	27.3mW **(15%)**
De-Sync.	28.81mW **(14.41%)**	1.7mW **(0.85%)**	49.98mW **(25%)**

TABLE V
ASPIDA POST-LAYOUT POWER BREAKDOWN

For the de-synchronized mode of operation, which adapts to the scaling of the supply voltage, we performed post-layout 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 $\frac{1}{Vdd^n}$ 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 obtained the results shown in Figure VI. Thus, the potential power savings by voltage scaling that are made much easier by desynchronization are demonstrated.

Fig. 17. ASPIDA Die Photo

Supply Voltage	**Power**
2.5V	200mW
2.25V	167mW
2.1V	152mW
2V	143mW
1.85V	127mW
1.5V	98mW

TABLE VI
ASPIDA POST-LAYOUT POWER SCALING FOR DE-SYNC. MODE

The performance difference between synchronous and de-synchronized 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 de-synchronized 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. A 2nd run of the ASPIDA chip, which has been scheduled for fabrication in July 2005, has carefully tuned delay-elements and exercises P&R control. Post-P&R simulations of this chip show that the same performance can be obtained for synchronous and de-synchronized operation, with the de-synchronized 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 Figure 17, was fabricated in early 2005 and post-manufacturing measurements verified the correct operation of the first silicon. Extensive tests over about 90 fabricated chips yielded very interesting results. The de-synchronized 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 post-manufacturing, even at 0.25μm. The current models are simply too pessimistic. With technology scaling in the nanometer era, this problem will get worse.

In addition, the de-synchronized 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

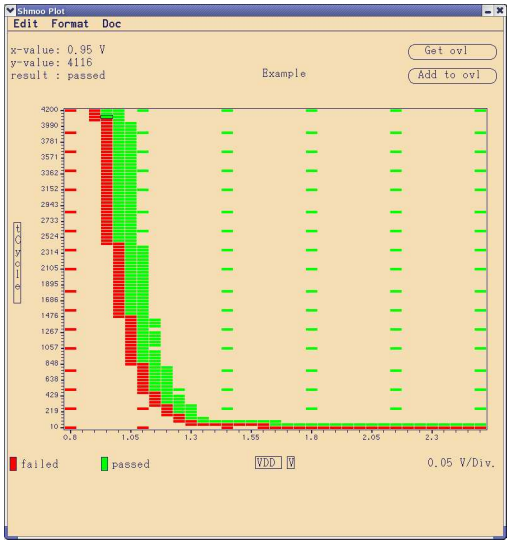


Fig. 18. ASPIDA: Schmoo Plot in Asynchronous Operation

plot of Figure 18.

Figure 18 plots voltage on the X-axis. Green (or light gray) dots indicate chips that pass the functional test, whereas red (or dark gray) dots indicate chips that fail. The plot indicates correct operation in de-synchronized mode all the way down to 0.95V, which is only 0.35V away from the threshold voltage of the process, which is 0.6V. This is strong evidence that the de-synchronization approach handles variability very well, as at the very low operational voltage of 0.95V all second and third order phenomena of transistor behaviour are in full effect.

In de-synchronized operation both the processor speed and voltage can be controlled using a single variable, i.e. supply voltage, whereas in 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 de-synchronized operation allows. In addition, tests over a range of 90 chips demonstrated, as shown in Figure 19, that the de-synchronized 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 de-synchronized design, whereas in synchronous mode, the external clocks must be adjusted according to the capabilities of the external circuits and extensive experimentation.

X. Conclusions

This paper presented a de-synchronization 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 SOC integration and shorter design cycles. Due to the partitioning of the clock trees, it also offers lower power and possibly lower Electro-magnetic 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 de-synchronization 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. [14]) 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 *Proceedings of the IEEE Design Automation Conference*, 2005.

[2] A. Bardsley and D. Edwards. Compiling the language Balsa to delay-insensitive hardware. In C. D. Kloos and E. Cerny, editors, *Hardware Description Languages and their Applications (CHDL)*, pages 89–91, Apr. 1997.

[3] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 162–177. Springer, August 1999.

[4] A. Benveniste, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli. Heterogeneous reactive systems modeling and correct-by-construction deployment. In R. Alur and I. Lee, editors, *Embedded software, third international conference, EMSOFT 2003*, volume 2855 of *Lecture notes in computer science*, pages 35–50. Springer, October 2003.

[5] K. v. Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[6] K. v. Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, pages 384–389, 1991.

[7] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou. Handshake protocols for de-synchronization. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 149–158, 2004.

[8] I. Blunno and L. Lavagno. Automated synthesis of micro-pipelines from behavioral Verilog HDL. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 84–92. IEEE Computer Society Press, Apr. 2000.

[9] 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*, chapter 3. Kluwer Academic Publishers, 2002.

[10] M. Coenen. On-chip measures to achieve EMC. In *IEEE International Symposium on EMC*, pages 31–36, Feb. 1997.

[11] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.

[12] J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou. From synchronous to asynchronous: An automatic approach. In *Proc. Design, Automation and Test in Europe (DATE)*, volume 2, pages 1368–1369, 2004.

[13] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. A concurrent model for de-synchronization. In *Proceedings of the International Workshop on Logic Synthesis*, pages 294–301, 2003.

[14] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. Coping with the variability of combinational logic delays. In *Proc. IEEE Int. Conf. on Computer Design*, Oct. 2004.

Fig. 19. ASPIDA: Period in Synchronous (left) and De-synchronized (right) operation vs. Supply Voltage Vdd

[15] A. Davare, K. Lwin, A. Kondratyev, and A. L. Sangiovanni-Vincentelli. The best of both worlds: the efficient asynchronous implementation of synchronous specifications. In *Proceedings of the IEEE Design Automation Conference*, pages 588–591, 2004.

[16] L. Dennison, W. Dally, and T. Xanthopoulos. Low-latency plesiochronous data retiming. In *Advanced Research in VLSI*, pages 304–315, 1995.

[17] S. Furber, P. Day, J. Garside, N. Paver, et al. AMULET1: a micropipelined ARM. In *Proceedings of the IEEE COMPCON*, pages 476–485, 1994.

[18] S. B. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE Transactions on VLSI Systems*, 4(2):247–253, June 1996.

[19] S. B. Furber, J. D. Garside, and D. A. Gilbert. AMULET3: A high-performance self-timed ARM microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, Oct. 1998.

[20] P. L. Guernic, J.-P. Talpin, and J.-C. L. Lann. Polychrony for system design. *Journal of Circuits, Systems and Computers*, Apr. 2003.

[21] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.

[22] J. L. Hennessy and D. Patterson. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Publisher Inc., 1990.

[23] J. Kessels and A. Peeters. The Tangram framework: Asynchronous circuits for low power. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 255–260, Feb. 2001.

[24] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim. Clock synchronization through handshake signalling. *Microprocessors and Microsystems*, 27(9):447–460, Oct. 2003.

[25] R. Kol and R. Ginosar. A doubly-latched asynchronous pipeline. In *Proc. International Conf. Computer Design (ICCD)*, pages 706–711, Oct. 1996.

[26] A. Kondratyev, L. Sorenson, and A. Streich. Testing of asynchronous designs by inappropriate means: Synchronous approach. In *Proc. IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, March 2001.

[27] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125. IEEE Computer Society Press, Apr. 2000.

[28] D. H. Linder and J. C. Harden. Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry. *IEEE Transactions on Computers*, 45(9):1031–1044, Sept. 1996.

[29] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

[30] S. Nassif, D. Boning, and N. Hakim. The care and feeding of your statistical static timer. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2005.

[31] C. D. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *Proc. ACM/IEEE Design Automation Conference*, pages 70–76, June 1994.

[32] O. A. Petlin and S. B. Furber. Scan testing of micropipelines. In *Proc. IEEE VLSI Test Symposium*, pages 296–301, May 1995.

[33] R. B. Reese and M. A. T. C. Traver. A coarse-grain phased logic CPU. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–13. IEEE Computer Society Press, May 2003.

[34] R. Smith, K. Fant, D. Parker, R. Stephani, and C. Y. Wang. An asynchronous 2-D discrete cosine transform chip. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 224–233, 1998.

[35] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, et al. A fully asynchronous low-power error corrector for the DCC player. *Journal of Solid State Circuits*, 29(12):1429–1439, Dec. 1994.

[36] V. Varshavsky, V. Marakhovsky, and T.-A. Chu. Logical timing (global synchronization of asynchronous arrays). In *The First International Symposium on Parallel Algorithm/Architecture Synthesis*, pages 130–138, Aizu-Wakamatsu, Japan, Mar. 1995.

Jordi Cortadella (M'88) received the M.S. and Ph.D. degrees in Computer Science from the Universitat Politècnica de Catalunya, Barcelona, 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 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 the best paper awards at the Int. Symp. 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.