Latency Insensitive Protocols

Luca P. Carloni[1], Kenneth L. McMillan[2], and
Alberto L. Sangiovanni-Vincentelli[1]

[1] University of California at Berkeley, Berkeley, CA 94720-1772
{lcarloni,alberto}@ic.eecs.berkeley.edu
[2] Cadence Berkeley Laboratories, Berkeley, CA 94704-1103
mcmillan@cadence.com

Abstract. *The theory of latency insensitive design is presented as the foundation of a new* correct by construction *methodology to design very large digital systems by assembling blocks of Intellectual Properties. Latency insensitive designs are synchronous distributed systems and are realized by assembling functional modules exchanging data on communication channels according to an appropriate protocol. The goal of the protocol is to guarantee that latency insensitive designs composed of functionally correct modules, behave correctly independently of the wire delays. A latency-insensitive protocol is presented that makes use of relay stations buffering signals propagating along long wires. To guarantee correct behavior of the overall system, modules must satisfy weak conditions. The weakness of the conditions makes our method widely applicable.*

1 Introduction

The level of integration available today with Deep Sub-Micron (DSM) technologies ($0.1\mu m$ and below) is so high that entire systems can now be implemented on a single chip. Designs of this kind expose problems that were barely visible at previous levels of integration: the dominance of wire delays on the chip and the strong effects created by the clock skew [2]. It is predicted that a signal will need more than five (and up to more than ten!) clock ticks to traverse the entire chip area. Thus it will be very important to limit the distance traveled by critical signals to guarantee the performance of the design. However, precise data on wire-lengths are available late in the design process and several costly re-designs may be needed to change the placement or the speed of the components of the design to satisfy performance and functionality constraints. We believe that, for deep sub-micron designs where millions of gates are customary, a design method that guarantees by construction that certain properties are satisfied is the only hope to achieve correct designs in short time. In particular, we focus on methods that allow a designer to compose pre-designed and verified components so that the composition formally satisfies certain properties.

In this paper, we present a theory for the design of digital systems that maintains the inherent simplicity of synchronous designs and yet does not suffer of the "long-wire" problem. According to our methodology, the system can be thought as completely synchronous, i.e. just as a collection of modules that communicate by means of channels having a latency of one clock cycle. Unfortunately, the final layout may require more than one clock cycle to transmit the appropriate signals. Our methodology does not require costly re-design cycles or to slow down the clock. The key idea is borrowed from pipelining: partition the long wires into segments whose lengths satisfy the timing requirements imposed by the clock by inserting logic blocks called *relay stations*, which have a function similar to the one of latches on a pipelined data-path. The timing requirements imposed by the real clock are now met by construction. However, the latency of a channel connecting two modules is generally equal to more than one real clock cycle. If the functionality of the design is based on the sequencing of the output signals and not on their exact timing, then this modification of the design does not change functionality provided that the components of the design are *latency insensitive*, i.e., the behavior of each module does not depend on the latency of the communication channels. We have essentially traded off latency for throughput by not slowing down the clock and by inserting relay stations. In this paper, we introduce these concepts formally and prove the properties outlined above. Classical works on trace theory [3, 7] and delay insensitive circuits could be used to address our problem, but these approaches imply that the delay between two events on a communication channel is completely arbitrary, while in our case we obtain stronger results by assuming that this arbitrary delay is a *multiple of the clock period.*

The paper is organized as follows: in Section 2 we give the foundation of latency insensitive design by presenting the notion of patient processes. In Section 3 we discuss how in a system of patient processes communication channels can be segmented by introducing *relay stations.* Section 4 illustrates the overall design methodology and discusses under which assumption a generic system can be transformed in a patient one.

2 Latency Insensitivity

To cast our methodology in a formal framework, we use the approach of Lee and Sangiovanni-Vincentelli to represent signals and processes [5].

2.1 The Tagged-Signal Model

Given a set of *values* $\mathcal{V}$ and a set of *tags* $\mathcal{T}$, an *event* is a member of $\mathcal{V} \times \mathcal{T}$. Two events are *synchronous* if they have the same tag. A *signal* s is a set of events. Two signals are synchronous if each event in one signal is synchronous with an event in the other signal and vice versa. Synchronous signals must have the same set of tags.

The set of all N-tuples of signals is denoted $\mathcal{S}^N$. A *process* P is a subset of $\mathcal{S}^N$. A particular N-tuple $\mathbf{s} \in \mathcal{S}^N$ satisfies the process if $\mathbf{s} \in P$. A N-tuple $\mathbf{s}$ that satisfies a process is called a *behavior* of the process. Thus a process is a set of possible behaviors [1]. A *composition of processes* (also called a *system*) $\{P_1, \ldots, P_M\}$, is a process defined as the intersection of their behaviors $P = \bigcap_{m=1}^{M} P_m$. Since processes can be defined over different signal sets, to form the composition we need to extend the set of signals over which each process is defined to contain all the signals of all processes. Note that the extension changes the behavior of the processes only formally.

Let $J = (j_1, \ldots, j_h)$ be an ordered set of integers in the range $[1, N]$, the projection of a behavior $b = (s_1, \ldots, s_N) \in \mathcal{S}^N$ onto $\mathcal{S}^h$ is $proj_J(b) = (s_{j_1}, \ldots, s_{j_h})$. The projection of a process $P \subseteq \mathcal{S}^N$ onto $\mathcal{S}^h$ is $proj_J(P) = (\mathbf{s}' \mid \exists \mathbf{s} \in P \wedge proj_J(\mathbf{s}) = \mathbf{s}')$. A *connection* C is a particularly simple process where two (or more) of the signals in the N-tuple are constrained

[1] For $N \geq 2$, processes may also be viewed as a relation between the N signals in $\mathbf{s}$.

to be identical: for instance, $C(i,j,k) \subset \mathcal{S}^N : (s_1,\ldots,s_N) \in C(i,j,k) \Leftrightarrow s_i = s_j = s_k$, with $i,j,k \in [1,N]$.

In a *synchronous system* every signal in the system is synchronous with every other signal. In a *timed system* the set $\mathcal{T}$ of tags, also called *timestamps*, is a totally ordered set. The ordering among the timestamps of a signal s induces a natural order on the set of events of s.

2.2 Informative Events and Stalling Events

A latency insensitive system is a synchronous timed system whose set of values $\mathcal{V}$ is equal to $\Sigma \cup \{\tau\}$, where Σ is the set of *informative symbols* which are exchanged among modules and $\tau \notin \Sigma$ is a special symbol, representing the absence of an informative symbol. From now on, all signals are assumed to be synchronous. The set of timestamps is assumed to be in one-to-one correspondence with the set $\mathbb{N}$ of natural numbers. An event is called *informative* if it has an informative symbol ι_i as value [2]. An event whose value is a τ symbol is said a *stalling event* (or τ *event*).

Definition 1. *$\mathcal{E}(s)$ denotes the set of events of signal s while $\mathcal{E}_\iota(s)$ and $\mathcal{E}_\tau(s)$ are respectively the set of informative events and the set of stalling events of s. The k-th event (v_k,t_k) of a signal s is denoted $e_k(s)$. $\mathcal{T}(s)$ denotes the set of timestamps in signal s, while $\mathcal{T}_\iota(s)$ is the set of timestamps corresponding to informative events.*

Processes exchange "useful" data by sending and receiving informative events. Ideally only informative events should be communicated among processes. However, in a latency insensitive system, a process may not have data to output at a given timestamp thus requiring the output of a stalling event at that timestamp.

Definition 2. *The set of all sequences of elements in $\Sigma \cup \{\tau\}$ is denoted by Σ_{lat}. The length of a sequence σ is $|\sigma|$ if it is finite, otherwise is infinity. The empty sequence is denoted as ϵ and, by definition, $|\epsilon| = 0$. The i-th term of a sequence σ is denoted σ_i.*

Definition 3. *Function $\sigma : \mathcal{S} \times \mathcal{T}^2 \to \Sigma_{lat}$ takes a signal $s = \{(v_0,t_0),(v_1,t_1),..\}$ and an ordered pair of timestamps (t_i,t_j), $i \leq j$, and returns a sequence $\sigma_{[t_i,t_j]} \in \Sigma_{lat}$ s.t. $\sigma_{[t_i,t_j]}(s) = v_i, v_{i+1}, \ldots, v_j$. The sequence of values of a signal is denoted $\sigma(s)$. The infinite subsequence of values corresponding to an infinite sequence of events, starting from t_i is denoted $\sigma_{[t_i,\infty]}(s)$.*

For example, considering signal $s = \{(\iota_1,t_1),(\iota_2,t_2),(\tau,t_3),(\iota_2,t_4),(\iota_1,t_5),(\tau,t_6)\}$ we have [3] $\sigma(s) = \iota_1 \ \iota_2 \ \tau \ \iota_2 \ \iota_1 \ \tau$, $\sigma_{[t_2,t_4]}(s) = \iota_2 \ \tau \ \iota_2$, $\sigma_{[t_5,t_5]}(s) = \iota_1$, and respectively, $|\sigma(s)| = 6$, $|\sigma_{t_2,t_4}(s)| = 3$, $|\sigma_{t_5,t_5}(s)| = 1$. To manipulate sequences of values we define the following filtering operators.

Definition 4. *$\mathcal{F}_\iota : \Sigma_{lat} \to \Sigma^\star$ returns a sequence $\sigma' = \mathcal{F}_\iota[\sigma]$ s.t.*

$$\sigma'_i = \begin{cases} \sigma_{[t_i,t_i]}(s) & \text{if } \sigma_{[t_i,t_i]}(s) \in \Sigma \\ \epsilon & \text{otherwise} \end{cases}$$

[2] We use subscripts to distinguish among the different informative symbols of $\Sigma : \iota_1, \iota_2, \iota_3, \ldots$

[3] In this paper we assume that for all timestamps $t_i, t_j \in \mathcal{T}(s)$, $t_i \leq t_j \Leftrightarrow i \leq j$.

Definition 5. *$\mathcal{F}_\tau : \Sigma_{lat} \rightarrow \{\tau\}^\star$ returns a sequence $\sigma' = \mathcal{F}_\tau[\sigma]$ s.t.*

$$\sigma'_i = \begin{cases} \sigma_{[t_i,t_i]}(s) & \text{if } \sigma_{[t_i,t_i]}(s) = \tau \\ \epsilon & \text{otherwise} \end{cases}$$

For instance, if $\sigma(s) = \iota_1\ \iota_2\ \tau\ \iota_2\ \iota_1\ \tau$, then $\mathcal{F}_\iota[\sigma(s)] = \iota_1\ \iota_2\ \iota_2\ \iota_1$ and $\mathcal{F}_\tau[\sigma(s)] = \tau\ \tau$. Obviously, $|\sigma(s)| = |\mathcal{F}_\iota[\sigma(s)]| + |\mathcal{F}_\tau[\sigma(s)]|$. Latency insensitive systems are assumed to have a finite horizon over which informative events appear, i.e., for each signal s there is a greatest timestamp $T \in \mathcal{T}_\iota(s)$ which corresponds to the "last" informative event. However, to build our theory we need to extend the set of signals of a latency insensitive system over an infinite horizon by adding a set of timestamps such that all events with timestamp greater than T have τ values.

Definition 6. *A signal s is* strict *if and only if (*iff*) all informative events precede all stalling events, i.e.,* iff *there exists a $k \in \mathbb{N}$ s.t. $|\mathcal{F}_\tau[\sigma_{[t_0,t_k]}(s)]| = 0$ and $|\mathcal{F}_\iota[\sigma_{[t_k,t_\infty]}(s)]| = 0$. A signal which is not strict is said to be* delayed *(or* stalled*).*

2.3 Latency Equivalence

Two signals are latency equivalent if they present the same sequence of informative events, i.e., they are identical except for different delays between two successive informative events. Formally:

Definition 7. *Two signals s_1, s_2 are latency equivalent $s_1 \equiv_\tau s_2$* iff *$\mathcal{F}_\iota[\sigma(s_1)] = \mathcal{F}_\iota[\sigma(s_2)]$.*

The *reference signal s_{ref}* of a class of latency equivalent signals is a strict signal obtained by assigning the sequence of informative values that characterizes the equivalence class to the first $|\mathcal{F}_\iota[\sigma(s_1)]|$ timestamps. For instance, signals s_1 and s_2 presenting the following sequences of values

$$\sigma(s_1) = \iota_1\ \iota_2\ \tau\ \iota_1\ \iota_2\ \iota_3\ \tau\ \iota_1\ \iota_2\ \tau\ \tau\ \tau\ \ldots$$
$$\sigma(s_2) = \iota_1\ \iota_2\ \tau\ \tau\ \iota_1\ \tau\ \iota_2\ \iota_3\ \tau\ \iota_1\ \tau\ \iota_2\ \tau \ldots$$

are latency equivalent. Their reference signal s_{ref} is characterized by the sequence of values $\sigma(s_{ref}) = \iota_1\ \iota_2\ \iota_1\ \iota_2\ \iota_3\ \iota_1\ \iota_2\ \tau\ \tau\ \tau \ldots$

Latency equivalent signals contain the same sequences of informative values, but with different timestamps. Hence, it is useful to identify their informative events with respect to the common reference signal: the *ordinal* of an informative event coincides with its position in the reference signal.

Definition 8. *The* ordinal *of an informative event $e_k = (v_k, t_k) \in \mathcal{E}_\iota(s)$ is defined as $ord(e_k) = |\mathcal{F}_\iota[\sigma_{[t_0,t_k]}](s)| - 1$. Let s_1 and q_1 be two latency equivalent signals: two informative events $e_k(s_1) \in \mathcal{E}_\iota(s_1)$ and $e_l(q_1) \in \mathcal{E}_\iota(q_1)$ are said* corresponding events iff *$ord(e_k(s_1)) = ord(e_l(q_1))$. The* slack *between two corresponding events is defined as $slack(e_k(s_1), e_l(q_1)) = |k - l|$.*

We extend the notion of latency equivalence to behaviors, in a component-wise manner:

Definition 9. *Two behaviors $(s_1, \ldots, s_N)$ and $(s'_1, \ldots, s'_N)$ are equivalent* iff *$\forall i\ (s_i \equiv_\tau s'_i)$. A behavior $b = (s_1, \ldots, s_N)$ is* strict iff *every signal $s_i \in b$ is strict. Every class of latency equivalent behaviors contains only one strict behavior: this is called the* reference behavior*.*

Definition 10. *Two processes P_1 and P_2 are latency equivalent, $P_1 \equiv_\tau P_2$, if every behavior of one is latency equivalent to some behavior of the other. A process P is strict* iff *every behavior $b \in P$ is strict. Every class of latency equivalent processes contains only one strict process: the* reference process.

Definition 11. *A signal s_1 is latency dominated by s_2, $s_1 \leq_\tau s_2$* iff *$s_1 \equiv_\tau s_2$ and $T_1 \leq T_2$, with $T_k = \max\{t \mid t \in \mathcal{T}_\iota(s_k)\}, k = 1, 2$.*

Hence, referring to the previous example, signal s_1 is dominated by signal s_2 since $T_1 = 9$ while $T_2 = 12$. Notice that a reference signal is latency dominated by every signal belonging to its equivalence class. Latency dominance is extended to behaviors and processes as in the case of latency equivalence. A total order among events of a behavior is necessary to develop our theory. In particular, we introduce an ordering among events that is motivated by causality: events that have smaller ordinal are ordered before the ones with larger ordinal (think of a strict process where the ordinal is related to the timestamp; the order implies that past events do not depend on future events). In addition, to avoid cyclic behaviors created by processing events with the same ordinal, we assume that there is an order among signals. This order in real-life designs corresponds to input-output dependencies. We cast this consideration in the most general form possible to extend maximally the applicability of our method.

Definition 12. *Given a behavior $b = (s_1, \ldots, s_N)$, $\leq_c$ denotes a well-founded order on its set of signals. The well-founded order induces a* lexicographic order *$\leq_{lo}$ over the set of informative events of b, s.t. for all pairs of events (e_1, e_2) with $e_1 \in \mathcal{E}_\iota(s_i)$ and $e_2 \in \mathcal{E}_\iota(s_j)$*

$$e_1 \leq_{lo} e_2 \quad \Leftrightarrow \quad [\, (ord(e_1) < ord(e_2)) \ \vee \ (\, (\, ord(e_1) = ord(e_2)\,) \wedge (s_i \leq_c s_j)\,)\,]$$

The following function returns the first informative event (in signal s_j of behavior b) following an event $e \in b$ with respect to the lexicographic order $\leq_{lo}$.

Definition 13. *Given a behavior $b = (s_1, \ldots, s_N)$ and an informative event $e(s_i) \in \mathcal{E}_\iota(s_i)$, the function* nextEvent *is defined as: $nextEvent(s_j, e(s_i)) = \min_{e_k(s_j) \in \mathcal{E}_\iota(s_j)}\{e(s_i) \leq_{lo} e_k(s_j)\}$*

A *stall move* postpones an informative event of a signal of a given behavior by one timestamp. The stall move is used to account for long delays along wires and to add delays where needed to guarantee functional correctness of the design.

Definition 14. *Given a behavior $b = (s_1, \ldots, s_j, \ldots, s_N)$ and an informative event $e_k(s_j) = (v_k, t_k)$, a* stall move *returns a behavior $b' = stall(e_k(s_j)) = (s_1, \ldots, s'_j, \ldots, s_N)$, s.t. for all $l \in \mathbb{N}$: $\sigma_{[t_0,t_{k-1}]}(s'_j) = \sigma_{[t_0,t_{k-1}]}(s_j)$, $\sigma_{[t_k,t_k]}(s'_j) = \tau$, $\sigma_{[t_{k+l+1},t_{k+l+1}]}(s'_j) = \sigma_{[t_{k+l},t_{k+l}]}(s_j)$.*

A *procrastination effect* represents the "effect" of a stall move $stall(e_k(s_j))$ on other signals of behavior b in correspondence of events following $e_k(s_j)$ in the lexicographic order. The processes will "respond" to the insertion of stalls in some of their signals "delaying" other signals that are causally related to the stalled signals.

Definition 15. *A* procrastination effect *is a point-to-set map which takes a behavior $b' = (s'_1, \ldots, s'_N) = stall(e_k(s_j))$ resulting from the application of a stall move on event $e_k(s_j)$ of behavior $b = (s_1, \ldots, s_N)$ and returns a set of behaviors $\mathcal{PE}[stall(e_k(s_j))]$ s.t. $b'' = (s''_1, \ldots, s''_N) \in \mathcal{PE}[b']$* iff

- $s''_j = s'_j$;
- $\forall i \in [1,N], i \neq j$, $s''_i \equiv_\tau s'_i$ *and* $\sigma_{[t_0,t_{l-1}]}(s''_i) = \sigma_{[t_0,t_{l-1}]}(s'_i)$, *where* t_l *is the timestamp of event* $e_l(s_i) = nextEvent(s_i, e_k(s_j))$;
- $\exists K$ *finite s.t.* $\forall i \in [1,N], i \neq j, \exists k_i \leq K$, $\sigma_{[t_{l+k_i},\infty]}(s''_i) = \sigma_{[t_l,\infty]}(s'_i)$.

Each behavior in $\mathcal{PE}[b']$ is obtained from b' by possibly inserting other stalling events in any signal of b', but only at "later" timestamps, i.e. to postpone informative event which follow $e_k(s_j)$ with respect to the lexicographic order $\leq_{lo}$. Observe that a procrastination effect returns a behavior that latency dominates the original behavior.

2.4 Patient Processes

We are now ready to define the notion of patient process: a patient process can take stall moves on any signal of its behaviors by reacting with the appropriate procrastination effects. Patience is the key condition for the IP blocks to be combinable according to our method. The following theorems [4] guarantee that, for patient processes, the notion of latency equivalence of processes is compositional.

Definition 16. *A process* P *is* patient iff

$$\forall b = (s_1, \ldots, s_N) \in P,\ \forall j \in [1,N],\ \forall e_k(s_j) \in \mathcal{E}_\iota(s_j),\ (\mathcal{PE}[stall(e_k(s_j))] \cap P \neq \emptyset)$$

Hence, the result of a stall move on one of the events of a patient process may not satisfy the process, but one of the behaviors of the procrastination effect corresponding to the stall move does satisfy the process, i.e., if we stall a signal on an input to a functional block, the block will be forced to delay some of its outputs or if we request an output signal to be delayed then an appropriate delay has to be added to the inputs.

Lemma 1. *Let* P_1 *and* P_2 *be two patient processes. Let* $b_1 \in P_1$, $b_2 \in P_2$ *be two behaviors with the same lexicographic order s.t.* $b_1 \equiv_\tau b_2$. *Then, there exists a behavior* $b' \in (P_1 \cap P_2)$, $b_1 \equiv_\tau b' \equiv_\tau b_2$.

Theorem 1. *If* P_1 *and* P_2 *are patient processes then* $(P_1 \cap P_2)$ *is a patient process.*

Theorem 2. *For all patient processes* P_1, P_2, P'_1, P'_2, *if* $P_1 \equiv_\tau P'_1$ *and* $P_2 \equiv_\tau P'_2$ *then* $(P_1 \cap P_2) \equiv_\tau (P'_1 \cap P'_2)$

Therefore, we can replace any process in a system of patient processes by a latency equivalent process, and the resulting system will be latency equivalent. A similar theorem holds for replacing strict processes with patient processes.

Theorem 3. *For all strict processes* P_1, P_2 *and patient processes* P'_1, P'_2, *if* $P_1 \equiv_\tau P'_1$ *and* $P_2 \equiv_\tau P'_2$ *then* $(P_1 \cap P_2) \equiv_\tau (P'_1 \cap P'_2)$

This means that we can replace all processes in a system of strict processes by corresponding patient processes, and the resulting system will be latency equivalent. This is the core of our idea: take a design based on the assumption that computation in one functional block and

[4] The proofs of the lemmas and the theorems presented in this paper can be found in [1].

communication among blocks "take no time" (synchronous hypothesis) [5], i.e., the processes corresponding to the functional blocks and their composition are strict, and replace it with a design where communication does take time (more than one clock cycle) and, as a result, signals are delayed, but without changing the sequence of informative events observed at the system level, i.e., with a set of patient processes.

3 Latency Insensitive Design

As explained in Section 1, one of the goal of the latency insensitive design methodology is to be able to "pipeline" a communication channel by inserting an arbitrary amount of memory elements. In the framework of our theory, this operation corresponds to adding some particular processes, called *relay stations*, to the given system. In this section, we first show how patient systems (i.e. systems of patient processes) are insensitive to the insertion of relay stations and, then, we discuss under which assumption a generic system can be transformed into a patient system.

3.1 Channels and Buffers

A *channel* is a connection [6] constraining two signals to be identical.

Definition 17. *A* channel $C(i,j) \subset \mathcal{S}^N, i,j \in [1,N]$ *is a process s.t.* $b = (s_1, ..., s_N) \in C(i,j) \Leftrightarrow s_i = s_j$.

Lemma 2. *A channel* $C(i,j) \subset \mathcal{S}^N$ *is not a patient process.*

Definition 18. *A* buffer $B^c_{l_f,l_b}(i,j)$ *with capacity* $c \geq 0$*, minimum forward latency* $l_f \geq 0$ *and minimum backward latency* $l_b \geq 0$ *is a process s.t.* $\forall i,j \in [1,N]$: $b = (s_1, ..., s_N) \in B^c_{l_f,l_b}(i,j)$ iff $(s_i \equiv_\tau s_j)$ *and* $\forall k \in \mathbb{N}$

$$0 \leq | \mathcal{F}_\iota [\sigma_{[t_0,t_{(k-l_f)}]}(s_i)] | - | \mathcal{F}_\iota [\sigma_{[t_0,t_k]}(s_j)] | \tag{1}$$

$$c \geq | \mathcal{F}_\iota [\sigma_{[t_0,t_k]}(s_i)] | - | \mathcal{F}_\iota [\sigma_{[t_0,t_{(k-l_b)}]}(s_j)] | \tag{2}$$

By definition, given a pair of indexes $i,j \in [1,N]$, for all $l_b, l_f, c \geq 0$, all buffers $B^c_{l_f,l_b}(i,j)$ are latency equivalent. Observe also that buffer $B^0_{0,0}(i,j)$ coincides with channel $C(i,j)$. In particular, we are interested in buffers having unitary latencies and we want to establish under which conditions such buffers are patient processes.

Theorem 4. *Let* $l_b = l_f = 1$*. For all* $c \geq 1$, $B^c_{1,1}(i,j)$ *is patient* iff $s_i \leq_c s_j$.

Consider a strict system $P_{strict} = \bigcap_{m=1}^{M} P_m$ with N strict signals $s_1, \ldots, s_N$. As explained in section 2.1, processes can be defined over different signal sets and to compose them we may need to formally extend the set of signals of each process to contain all the signals of all processes. However, without loss of generality, consider the particular case of composing

[5] In other words, communication and computation are completed in one clock cycle.
[6] See section 2.1 for the definition of connection.

$$B^1_{1,1}\begin{cases} s_1 &= \iota_1\ \tau\ \iota_2\ \tau\ \iota_3\ \tau\ \iota_4\ \tau\ \tau\ \tau\ \iota_5\ \tau\ \iota_6\ \tau\ \iota_7\ \tau\ \iota_8\ \tau\ \iota_9\ \tau\ \tau\ \tau\ \iota_{10}\ \tau\ \ldots \\ s_2 &= \tau\ \iota_1\ \tau\ \iota_2\ \tau\ \iota_3\ \tau\ \iota_4\ \tau\ \tau\ \tau\ \iota_5\ \tau\ \iota_6\ \tau\ \iota_7\ \tau\ \iota_8\ \tau\ \tau\ \tau\ \iota_9\ \tau\ \iota_{10}\ \tau\ \ldots \end{cases}$$

$$B^2_{1,1}\begin{cases} s_1 &= \iota_1\ \iota_2\ \iota_3\ \tau\ \tau\ \iota_4\ \iota_5\ \iota_6\ \tau\ \tau\ \tau\ \iota_7\ \tau\ \iota_8\ \iota_9\ \iota_{10}\ \ldots \\ s_2 &= \tau\ \iota_1\ \iota_2\ \iota_3\ \tau\ \tau\ \iota_4\ \tau\ \tau\ \tau\ \iota_5\ \iota_6\ \iota_7\ \tau\ \iota_8\ \iota_9\ \iota_{10}\ \ldots \end{cases}$$

Fig. 1. Comparing two possible behaviors of finite buffers $B^1_{1,1}$ and $B^2_{1,1}$.

M processes which are already defined on the same N signals. Hence, any generic behavior $b_m = (s_{m_1}, \ldots, s_{m_N})$ of P_m is also a behavior of P_{strict} *iff* for all $l \in [1, M], l \neq m$ process P_l contains a behavior $b_l = (s_{l_1}, \ldots, s_{l_N})$ s.t. $\forall n \in [1, N]$ $(s_{l_n} = s_{m_n})$. In fact, we may assume to derive system P_{strict} by connecting the M processes with $(M-1)\cdot N$ channel processes $C(l_n, (l+1)_n)$, where $l \in [1, (M-1)]$ and $n \in [1, N]$. Further, we may also assume to "decompose" any channel process $C(m_n, l_n)$ with an arbitrary number X of channel processes $C(m_n, x_1), C(x_1, x_2), \ldots, C(x_{X-1}, l_n)$, by adding $X-1$ auxiliary signals, each of them forced to be equal to $m_n = l_n$. The theory developed in section 2 guarantees that if we replace each process $P_m \in P_{strict}$ with a latency equivalent patient process and each channel $C(i,j)$ with a patient buffer $B^1_{1,1}(i,j)$ we obtain a system $P_{patient}$ which is patient and latency equivalent to P_{strict}. In fact, *"having a patient buffer in a patient system is equivalent to having a channel in a strict system"*. Since "decomposing" a channel $C(i,j)$ has no observable effect on a strict system, we are therefore free to add an arbitrary number of patient buffers into the corresponding patient system to replace this channel. Since we use patient buffers with unitary latencies, we can distribute them along that long wire on the chip which implements $C(i,j)$, in such a way that the wire gets decomposed in segments whose physical lengths can be spanned in a single physical clock cycle.

3.2 Relay Stations

The following Lemma 3 proves that no behaviors in $B^1_{1,1}(i,j)$ may contain two informative events of s_i, s_j which are synchronous: this implies that the maximum achievable throughput across such a buffer is 0.5, which may be considered suboptimal. Instead, buffer $B^2_{1,1}(i,j)$ is the minimum capacity buffer which is able to "transfer" one informative unit per timestamp, thus allowing, in the best case, to communicate with maximum throughput equal to 1. Figure 1 compares two possible behaviors of these buffers.

Lemma 3. *$B^2_{1,1}(i,j)$ is the minimum capacity buffer with $l_f = l_b = 1$ s.t.*

$$\exists b^\star = (s^\star_1, \ldots, s^\star_N) \in B^2_{1,1}(i,j) \ \wedge\ \exists k \in \mathbb{N},\ (e_k(s^\star_i) \in \mathcal{E}_\iota(s^\star_i) \ \wedge\ e_k(s^\star_j) \in \mathcal{E}_\iota(s^\star_j))$$

Definition 19. *The buffer $B^2_{1,1}$ is called a* relay station *RS.*

4 Latency Insensitive Design Methodology

In this section, we move towards the implementation of the theory introduced in the previous sections. To do so, we assume that:

- the pre-designed functional blocks are synchronous processes;
- there is a set of signals for each process that can be considered as inputs to the process and a set of signals that can be considered as outputs of the process, i.e., the processes are *functional*;
- the processes are strictly causal (a process is *strictly causal* if two outputs can only be different at timestamps that strictly follow the timestamps when the inputs producing these outputs show a difference [7]).
- the processes belong to a particular class of processes called *stallable*, a weak condition to ask the processes to obey.

The basic ideas are as follows. Composing a set of pre-designed synchronous functional blocks in the most efficient way is fairly straightforward if we assume that the synchronous hypothesis holds. This composition corresponds to a composition of strict processes since there is a priori no need of inserting stalling events. However, as we have argued in the introduction, it is very likely that the synchronous hypothesis will not be valid for communication. If indeed the processes to be composed are patient, then adding an appropriate number of relay stations yields a process that is latency equivalent to the strict composition. Hence, if we use as the definition of correct behavior the fact that the sequences of informative events do not change, the addition of the relay stations solves the problem. However, requiring processes to be patient at the onset is quite strong. Still, in practice, a patient system can be *derived* from a strict one as follows: first, we take each strict process P_m and we compose it with a set of auxiliary processes to obtain an equivalent patient process P'_m. To be able to do so, all processes P_m must satisfy a simple condition (the processes must be stallable) specified in the next section. Then, we put together all patient processes by connecting them with relay stations. The set of auxiliary processes implements a "queuing mechanism" across the signal of P_m in such a way that informative events are buffered and reordered before being passed to P_m: informative events having the same ordinal are passed to P_m synchronously.

In the sequel, we first introduce the formal definition of functional processes. Then, we present the simple notion of stallable processes and we prove that every stallable process can be encapsulated into a wrapper process which acts as an interface towards a latency insensitive protocol.

4.1 Stallable Processes

An *input* to a process $P \subseteq \mathcal{S}^N$ is an externally imposed constraint $P_I \subseteq \mathcal{S}^N$ such that $P_I \cap P$ is the total set of acceptable behaviors. Commonly, one considers processes having input signals and output signals: in this case, given process P, the set of signals can be partitioned into three disjoint subsets by partitioning the index set as $\{1, \ldots, N\} = I \cup O \cup R$, where I is the ordered set of indexes for the input signals of P, O is the ordered set of indexes for the output signals and R is the ordered set of indexes for the remaining signals (also called irrelevant signals with respect to P). A process is *functional* with respect to (I, O) if for all behaviors $b \in P$ and $b' \in P$, $proj_I(b) = proj_I(b')$ implies $proj_O(b) = proj_O(b')$.

In the sequel, we consider only strictly causal processes and for each of them we assume that the well founded order $\leq_c$ of definition 12 subsumes the causality relations among its signals, i.e. formally: $\forall i \in I, \forall j \in O, (s_i \leq_c s_j)$.

[7] For a more formal definition see [5].

$$\begin{array}{|lcl c lcl|}
\hline
s_1 & = & \iota_1\ \iota_3\ \iota_1\ \tau\ \iota_3\ \tau\ \tau\ \ldots & & s_4 & = & \tau\ \iota_1\ \tau\ \iota_3\ \iota_1\ \tau\ \iota_3\ \ldots \\
s_2 & = & \tau\ \iota_4\ \tau\ \iota_7\ \iota_8\ \tau\ \iota_8\ \ldots & \longrightarrow & s_5 & = & \tau\ \iota_4\ \tau\ \iota_7\ \iota_8\ \tau\ \iota_8\ \ldots \\
s_3 & = & \tau\ \iota_5\ \iota_5\ \tau\ \iota_9\ \tau\ \iota_6\ \ldots & & s_6 & = & \tau\ \iota_5\ \tau\ \iota_5\ \iota_9\ \tau\ \iota_6\ \ldots \\
\hline
\end{array}$$

Fig. 2. Example of a behavior of an equalizer E with $I = \{1, 2, 3\}$ and $O = \{4, 5, 6\}$.

Definition 20. *A process P with $I = \{1, \ldots, Q\}$ and $O = \{Q+1, \ldots, N\}$ is* stallable iff *for all $b = (s_1, \ldots, s_Q, s_{Q+1}, \ldots, s_N) \in P$ and for all $k \in \mathbb{N}$:*

$$\forall i \in I \ \ (\sigma_{[t_k,t_k]}(s_i) = \tau) \ \ \Leftrightarrow \ \ \forall j \in O \ \ (\sigma_{[t_{k+1},t_{k+1}]}(s_j) = \tau)$$

Hence, while a patient process tolerates arbitrary distributions of stalling events among its signals (as long as causality is preserved), a stallable process demands more regular patterns: τ symbols can only be inserted synchronously (i.e., with the same timestamp) on all input signals and this insertion implies the synchronous insertion of τ symbols on all output signals at the following timestamp. To assume that a functional process is stallable is quite reasonable with respect to a practical implementation. In fact, most hardware systems can be stalled: for instance, consider any sequential logic block that has a *gated clock* or a Moore finite state machine M with an extra input, that, if equal to τ, forces M to stay in the current state and to emit τ at the next cycle.

4.2 Encapsulation of Stallable Processes

Now, our goal is to define a group of functional processes that can be composed with a stallable process P to derive a patient process which is latency equivalent to P. We start considering a process that aligns all the informative events across a set of channels.

Definition 21. *An* equalizer *E is a process, with $I = \{1, \ldots, Q\}$ and $O = \{Q+1, \ldots, 2 \cdot Q\}$, s.t. for all behaviors $b = (s_1, \ldots, s_Q, s_{Q+1}, \ldots, s_{2 \cdot Q}) \in E$: $\ \forall i \in I, (s_i \equiv_\tau s_{Q+i})$ and $\forall k \in \mathbb{N}$*

$$\begin{array}{l}
\forall i, j \in O \ \ (\ (\sigma_{[t_k,t_k]}(s_i) = \tau) \Rightarrow (\sigma_{[t_k,t_k]}(s_j) = \tau) \ ) \\
\min\limits_{i \in I} \{ \mid \mathcal{F}_\iota \, [\sigma_{[t_0,t_k]}(s_i)] \mid \} \ - \ \max\limits_{j \in O} \{ \mid \mathcal{F}_\iota \, [\sigma_{[t_0,t_k]}(s_j)] \mid \} \ \ \geq 0
\end{array}$$

The first relation forces the output signals to have stalling events only synchronously, while the second guarantees that at every timestamp the number of informative events occurred at any input is always greater than the number of informative events occurred at any output. In particular, the presence of a stalling event at any input at a given timestamp forces the presence of a stalling event on all outputs at the same timestamp. Figure 2 illustrates a possible behavior of an equalizer.

Definition 22. *An* extended relay station *$\mathcal{ERS}$ is a process with $I = \{i\}$ and $O = \{j, l\}$, $i \neq j \neq l$ s.t. signals s_q, s_2 are related by inequalities (1) and (2) of definition 18 (with $l_f = l_b = 1$ and $c = 2$) and $\forall k \in \mathbb{N}$:*

$$\sigma_{[t_k,t_k]}(s_l) = \begin{cases} 1 \ \textit{if} \ \mid \mathcal{F}_\iota \, [\sigma_{[t_0,t_k]}(s_i)] \mid \ - \ \mid \mathcal{F}_\iota \, [\sigma_{[t_0,t_{k-1}]}(s_j)] \mid = \ 2 \\ 0 \ \textit{otherwise} \end{cases}$$

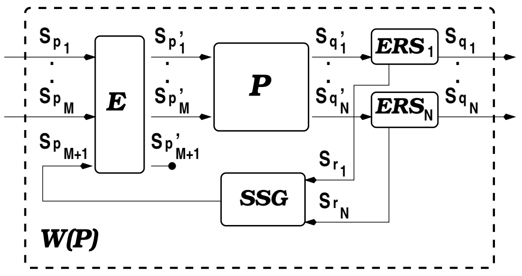

Fig. 3. Encapsulation of a stallable process P into a wrapper $W(P)$.

Definition 23. *A* stalling signal generator $\mathcal{SSG}$ *is a process with* $I = \{1, \ldots, Q\}$ *and* $O = \{Q+1\}$ *s.t.* $\forall b = (s_1, \ldots, s_{Q+1}), \forall k \in \mathbb{N}, \forall i \in [1, Q], (\mathcal{F}_\iota\ [\sigma_{[t_k,t_k]}(s_i)] \in [0,1])$ *and*

$$\sigma_{[t_k,t_k]}(s_{Q+1}) = \begin{cases} \tau \text{ if } \exists j \in [1, Q]\ (\ \mathcal{F}_\iota\ [\sigma_{[t_k,t_k]}(s_j)] = 1\ ) \\ 0 \text{ otherwise} \end{cases}$$

As illustrated in Figure 3, any stallable process P can be composed with an equalizer, a stalling signal generator and some extended relay stations to derive a patient process which is latency equivalent to P.

Definition 24. *Let P be a stallable process with $I_P = \{p'_1, \ldots, p'_M\}$ and $O_P = \{q'_1, \ldots, q'_N\}$. A* wrapper process *(or,* shell process*) $W(P)$ of P is the process with $I_W = \{p_1, \ldots, p_M\}$ and $O_W = \{q_1, \ldots, q_N\}$ which is obtained composing P with the following processes:*

- *an equalizer E with $I_E = \{p_1, \ldots, p_M, p_{M+1}\}$ and $O_E = \{p'_1, \ldots, p'_M, p'_{M+1}\}$,*
- *N extended relay stations $\mathcal{ERS}_1, \mathcal{ERS}_2, \ldots, \mathcal{ERS}_N$ s.t. $I_j = \{q'_j\}$ and $O_j = \{q_j, r_j\}$, with $j \in [1, N]$*
- *a* stalling signal generator *$\mathcal{SSG}$ with $I_G = \{r_1, \ldots, r_N\}$ and $O_G = \{p_{M+1}\}$.*

Theorem 5. *Let $W(P)$ be the wrapper process of def. 24. Process $W = proj_{I_W \cup O_W}(W(P))$ is a patient process that is latency equivalent to P.*

In conclusion, our latency insensitive design methodology can be summarized as follows:

1. Begin with a system of M stallable processes and N channels.
2. Encapsulate each stallable process to yield a wrapper process.
3. Using relay stations decompose each channel in segments whose physical length can be spanned in a single physical clock cycle.

This approach clearly "orthogonalizes" computation and communication: in fact, we can build systems by putting together hardware cores (which can be arbitrarily complex as far as they satisfy the stalling assumption) and wrappers (which interface them with the channels, by "speaking" the latency insensitive protocol). While the specific functionality of the system is distributed in the cores, the wrappers can be automatically generated around them [8]. Finally, the validation of the system can now be efficiently decomposed based on assume-guarantee reasoning [4, 6]: each wrapper is verified assuming a given protocol, and the protocol is verified separately.

[8] This is the reason why wrappers are also called *shells*: they just "protect" the intellectual property (*the pearl*) they contain from the "troubles" of the external communication architecture.

5 Conclusions and Future Work

A new design methodology for large digital systems implemented in DSM technology has been presented. The methodology is based on the assumption that the design is built by assembling blocks of Intellectual Properties (IPs) that have been designed and verified previously. The main goal is to develop a theory for the composition of the IP blocks that *ensures* the correctness of the overall design. The focus is on timing properties since DSM designs suffer (and will continue to suffer even more for the foreseeable future) from delays on long wires that often cause costly redesigns. Designs carried out with our methodology are called latency insensitive design. Latency insensitive designs are synchronous distributed systems and are realized by assembling functional modules exchanging data on communication channels according to a latency-insensitive protocol. The protocol guarantees that latency insensitive designs composed of functionally correct modules, behave correctly independently of the wire delays. This allow us to pipeline long wires by inserting special memory elements called relay stations. The protocol works on the assumption that the functional blocks satisfy certain weak properties.

The method trades-off latency for throughput, hence it is important to optimize the amount of latency that we must allow to obtain correct designs. This optimization leads to the concept of speculative latency insensitive protocols which will be the subject of a future paper.

6 Acknowledgments

We wish to acknowledge the discussions with Luciano Lavagno and Alex Saldanha that led to the theory of latency insensitive designs. Patrick Scaglia gave us strong support based on his experience as a designer of highly complex digital systems and continuous encouragement. This research has been partially sponsored by Cadence Design Systems, SRC and by CNR.

References

1. L. P. Carloni, K. L. McMillan, and Alberto L. Sangiovanni-Vincentelli. Latency-Insensitive Protocols. Technical Report UCB/ERL M99/11, Electronics Research Lab, University of California, Berkeley, CA 94720, February 1999.
2. D. Matzke. Will Physical Scalability Sabotage Performance Gains? *IEEE Computer*, 8(9):37–39, September 1997.
3. D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* The MIT Press, Cambridge, Mass., 1988. An ACM Distinguished Dissertation 1988.
4. T.A. Henzinger, S. Qadeer, and R.K. Rajamani. You Assume, We Guarantee: Methodology and Case Studies. In *Proceedings of the 10th International Conference on Computer-Aided Verification*, Vancouver, Canada, July 1998.
5. E. A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Transactions on Computer-Aided Design*, 17(12):1217–1229, December 1998.
6. K. L. McMillan. A Compositional Rule for Hardware Design Refinement. In *Proceedings of the 9th International Conference on Computer-Aided Verification*, Haifa, Israel, July 1997.
7. J. L. A. van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1985.