RECYCLE, REUSE, REDUCE

by

Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M99/53

9 October 1999

RECYCLE, REUSE, REDUCE

by

Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M99/53

9 October 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Recycle, Reuse, Reduce

Luca P. Carloni Alberto L. Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Science
University of California at Berkeley, Berkeley, CA 94720-1772
E-mail: {lcarloni, alberto}@ic.eecs.berkeley.edu

9th October 1999

Abstract

The theory of recycling *is presented as the foundation of a correct-by-construction methodology to* reuse *Intellectual Properties (IP) cores for building complex digital systems while* reducing *the number of iterations in the design process. The recently proposed theory of latency insensitive protocols lies at the basis of the present work. Latency insensitive systems are synchronous distributed systems, whose functionality is robust with respect to arbitrary variations in interconnect latency. However, the same robustness is not guaranteed for the performance of the design, which indeed may experience a notable degradation. This paper presents a simple, yet rigorous, method to (1) model the key properties of a latency insensitive system, (2) analyze the impact of interconnect latency on the overall throughput, and (3) optimize the performance of the final implementation.*

1 Introduction

As system complexity increases and market windows continue to shrink, effective reuse of existing designs or Intellectual Property (IP) cores seems the only way to produce a reliable design within a reasonable time [6, 19]. In fact, most semiconductor companies share both a strong will to broaden IP reuse and an open complaint towards the effectiveness of the tools offered by the EDA companies to meet this request [20, 24].

If to permit an easy trade, reuse and assembly of individual components of the chip is the goal, the new design methodologies should primarily facilitate (if not automatically provide) the solution of the communication and synchronization issues which arise while assembling pre-designed components. In this perspective, it is necessary to define a methodology which effectively addresses the increasing impact of interconnect delay in future design generations. Despite the increase in number of layers and in aspect ratio, the RC delay of an average metal line with constant length is getting worse with each process generation [17, 21] [1]. This effect, combined with the increases in operating frequency, die size, and average interconnect length, makes interconnect delay becoming a larger fraction of the clock cycle time [18]. Furthermore, while the number of gates reachable in a cycle will not change significantly and the on-chip bandwidth that wires provide will continue to grow, the percentage of the die reachable within one clock cycle will decrease dramatically: we will soon reach a point where more gates can be fit on a chip than can communicate in one cycle [8, 16].

On-chip communication has been cheap for a long time. This fact has lead to a number of architectural models that rely on low-latency to shared global resources. The popularity of these models is due to the fact that they provide the most uniform computational framework and the best functional unit utilization. As suggested in [16], this focus on function rather than communication is the fundamental conceptual roadblock to overcome. Meanwhile, currently available CAD flows force the designers to iterate many times between synthesis and layout, because the two steps are performed independently and synthesis uses statistical model that do not estimate the post-layout wire load capacitance accurately (*timing-closure problem*) [6, 14].

Long-term solutions must involve the adoption of interconnect structures with predictable performance [6], the definition of computational models that explicitly account for communication costs [16], and the development of machine architectures which expose their communication structure to the software compilers [1, 10]. A step in all these directions is represented by the latency insensitive design methodology, which has been recently proposed in literature [5]. Furthermore, this methodology may also accelerate the solution of the IP integration problem: in fact, it is fairly easy to assemble complex latency insensitive systems by reusing synchronously-specified functional modules because their interaction is controlled by a communication protocol that is insensitive to the latency of the channels connecting them. Yet, although the functionality of a latency insensitive system is robust with respect to interconnect delays, the same is not necessarily true for its performance. In [5], the authors do not address this problem nor do they suggest a technique for analyzing the latency/throughput trade-offs.

[1] Introducing copper metalization and low-κ dielectric insulators helps reduce interconnect delay, but these one-time improvements will not suffice in the long run as feature size continues to shrink [18, 19].

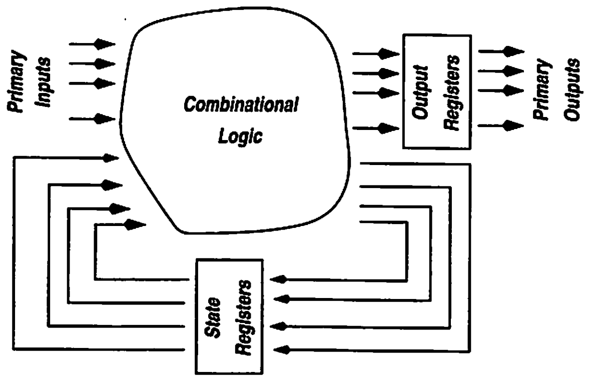

Figure 1: A Sequential Module.

In this paper, we introduce the theory of ***recycling*** as a formal way to capture the communication and synchronization properties of a latency insensitive system, thus enabling the analysis of the impact of a particular increase in latency (of one, or more, channels) on the overall system performance. This directly addresses the fundamental cause of the timing closure problem, while ***reducing*** the amount of iterations between logic synthesis and physical design and, ultimately, the entire design process time. Meanwhile, it simplifies the integration of IP components coming from different sources, thus handling the design complexity by ***reusing*** IP cores. Finally, allows us to derive a procedure that optimizes directly the system throughput, and that, in the case of a design made entirely of pre-designed IP blocks, provides the best achievable performance.

In Section 2 we discuss the realization of a hardware design by assembling Register-Transfer Level (RTL) modules as an example of a design approach based on IP reuse. The latency insensitive design methodology is summarized and criticized in Section 3. In Section 4 we give the definition of lis-graph as a formal model to represent the key properties of a latency insensitive system. This model allows us to specify the notion of recycling as a rigorous way to model the variations in latency of different wires and to compute exactly its impact on the performance of the system (Section 5). Finally, Section 6 illustrates the previous concept through the analysis of a case study, an implementation of the MPEG-2 Video Encoder.

2 Assembling IP Cores

The latency insensitive methodology together with the ideas proposed in the present work can be used to design complex distributed systems by assembling IP modules, which may be hardware blocks as well as software modules. However, in the rest of this paper we focus on synchronous digital circuits and we propose to design them by putting together IP cores having the structure of Figure 1, i.e. sequential circuits where: (1) any path between an input port and an output port contains one and only one register, which latches the output signal at the end of the path, and (2) both the state and output registers are controlled by a common clock signal. Such modules are commonly used to implement finite state machines (FSM), but any arbitrarily complex pipelined datapath can be seen as a cascade of stages having this structure. We indifferently refer to circuits with this structure as sequential modules, FSM modules, or RTL modules and, from now we assume that all IP cores that we encapsulate and compose to derive digital systems have these characteristics.

Example 2.1 A Multiplier-Accumulator (MAC) is a very common digital circuit, because it facilitates the implementation of an operation as $\sum x(n)\cdot y(n-k)$, which is ubiquitous in filters and vector arithmetic [13]. Fig. 2 illustrates an implementation of a MAC circuit: *inX* and *inY* are the input values to be multiplied, *inD* forces the accumulator register to be preset to the value of *inB*, *inTag* controls the index of the partial sum, and, finally, *outT* and *outW* are the outputs representing the sequence of indexed partial sums. If *inX* and *inY* are two N-bit signals, *regM* a $(2\cdot N)$-bit signal, and *regA* a $((2\cdot N)+M)$-bit signal (as well as *regC*, *inB* and *outO*), then 2^M repetitive *MPY/ACC* operations can be performed without overflow. We decompose the block diagram of the MAC in three major subcircuits, such that each of them can be realized with an RTL module. Then, we represent the structure of the system with a directed graph as illustrated in Fig. 3: nodes v_2, v_3, v_4 are associated to modules M_2, M_3, M_4, while nodes v_1 and v_5 represent respectively the input and the output buffers. The operations performed at each nodes are defined as follows, where s^i_n denotes the value of the signal traveling on arc a^i during the n-th clock cycle: node v_2 performs $s^6_n = s^3_{n-1}\cdot s^4_{n-1}$, while node v_3 does two operations, one for each leaving edges, namely $s^8_n = s^6_{n-1} + s^7_{n-1}$ and

$$s^7_n = \begin{cases} s^2_{n-1} & \text{if } s^1_{n-1} = 1 \\ s^6_{n-1} + s^7_{n-1} & \text{if } s^1_{n-1} = 0 \end{cases}$$

Finally, node v_4 composes the current result of a MAC operation with a tag index, i.e. $s^9_n = s^8_{n-1} \oplus s^5_{n-1}$ and $s^{10}_n = s^5_{n-2}$. Table 1 illustrates a possible MAC behavior spanning 12 clock cycles, during which the following computations are performed: $m_n = x_n \cdot y_n$, $w_n = z_n \oplus t_n$,

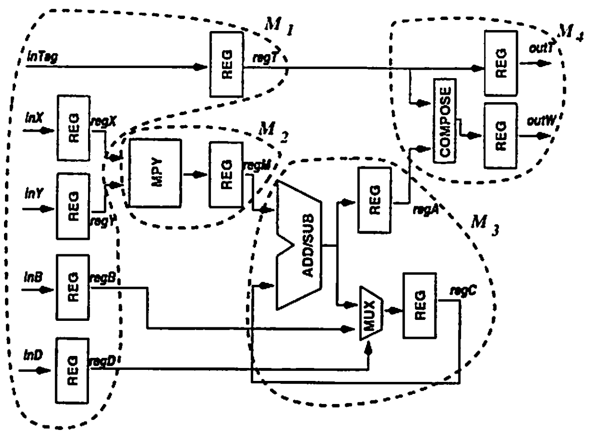

Figure 2: Block Diagram of a MAC circuit.

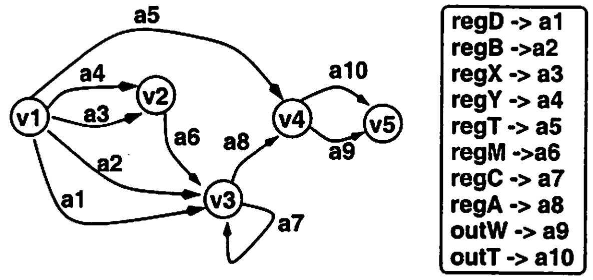

Figure 3: Graph representing the RTL structure of the MAC.

and

$$z_n = \begin{cases} \sum_{i=1}^{i=n} m_i & \text{if } n \in [1,5] \\ b_1 + \sum_{i=6}^{i=n} m_i & \text{if } n \in [6,9] \end{cases}$$

□

The advantage of providing an RTL specification of a digital system using sequential modules is that it is easy to derive the timing constraints imposed by each of them. In fact, the longest combinational path inside a module dictates the minimum clock period π_i which makes it operate correctly. Therefore, the task of specifying a large digital circuit can be decomposed in sub-tasks aimed to specify RTL modules having close π_i. This approach permits to handle the complexity of the overall design, by separating functional specification from performance analysis: once all modules are composed, the final system works correctly as far as it is running with a clock satisfying the constraint imposed by the slowest module, i.e. a clock having a period $\pi \geq \max_i\{\pi_i\}$. Thanks to its simplicity, this approach has been the basis of most digital design methodologies during last two decades, but its effectiveness is based on the assumption that the delay of any path connecting two modules is comparable to delay of the combinatorial paths inside the slowest modules in the system. Until the interconnect delay has been small with respect to gate delay, it has been easy to effectively decompose the system in such a way that inter-module combinational delays and intra-module wire delays are comparable. As discussed in the previous section, this is in not going to be the case for chip realized with DSM technologies. On the other hand, to estimate the delay of global interconnect early in the design process (e.g. at the floorplan phase) is extremely difficult and may easily lead to over-constrain the design, thus resulting in poor performance. We argue that to design high-performance complex digital system is necessary a methodology which (1) facilitates the composition of sequential modules in *pipeline mode* and (2) allows the potential insertion of extra-stages between one module and the other with the only purpose of buffering signals propagating on long wires. A new approach based on these ideas has been recently presented in literature and we discuss its pros and cons in the next section.

3 Latency Insensitive Systems

In [5], Carloni *et al.* have proposed a methodology to design very large digital systems by assembling IP cores exchanging data on point-to-point communication channels in accordance with a *latency insensitive protocol*. The protocol guarantees that a signals composed of functionally correct modules, behaves correctly independently from the delays of the channels connecting the modules.

Inputs					*Internal*			*Outputs*	
s^1	s^2	s^3	s^4	s^5	s^6	s^7	s^8	s^9	s^{10}
0	–	y_1	x_1	–	0	0	0	–	–
0	–	y_2	x_2	t_1	m_1	0	0	–	–
0	–	y_3	x_3	t_2	m_2	z_1	z_1	–	–
0	–	y_4	x_4	t_3	m_3	z_2	z_2	w_1	t_1
0	–	y_5	x_5	t_4	m_4	z_3	z_3	w_2	t_2
1	b_1	y_6	x_6	t_5	m_5	z_4	z_4	w_3	t_3
0	–	y_7	x_7	t_6	m_6	b_1	z_5	w_4	t_4
0	–	y_8	x_8	t_7	m_7	z_6	z_6	w_5	t_5
0	–	y_9	x_9	t_8	m_8	z_7	z_7	w_6	t_6
0	–	–	–	t_9	m_9	z_8	z_8	w_7	t_7
0	–	–	–	–	–	z_9	z_9	w_8	t_8
0	–	–	–	–	–	–	–	w_9	t_9

Table 1: Example of a behavior of the MAC circuit

As a consequence, a hardware implementation of the system can be automatically synthesized such that its functional behavior is robust with respect to large variations in wiring delays between modules. In fact, a long wire having a delay larger that the desired clock period can be pipelined in shorter segment by inserting special memory elements called *relay stations*, which buffer the signals traveling along the wire. The latency insensitive methodology can be summarized as follows:

1. the designer specifies a system as a collection of sequential modules (called *pearls*), whose interaction relies on the *synchronous hypothesis*, i.e. signals take one clock period to move from one pearl to another;
2. each pearl is encapsulated within an automatically generated *shell*: a shell is simply a collection of buffering queues (one for each port) plus the control logic that interfaces the pearl with the latency insensitive protocol;
3. traditional logic synthesis and place & route steps are applied to derive the layout of the chip implementing the system;
4. every wire whose latency is greater than the clock period is segmented by distributing on it the necessary amount of relay stations.

The only pre-condition required by this methodology is that the sequential modules be stallable, meaning that their operation can be frozen for an arbitrary amount of time without losing their internal state. This is a weak requirement because most hardware systems can be made stallable, for instance, implementing a *gated clock* mechanism. Observe that the final implementation of a latency insensitive system doesn't necessarily have to be synchronous, in the sense that is controlled by a single clock signal reaching every point on the chip. For instance, the relay stations as well as the overall communication architecture can be realized using asynchronous logic.

The big advantage of a latency insensitive design is that its functionality is robust with respect to arbitrarily large variations in wire delays as proven in [5]. Unfortunately the same can not be said for the performance of the design. In fact, the mechanism to implement a latency insensitive communication architecture is based on the distinction between *informative event* and *stalling event*. According to the synchronous specification of the system, at each clock cycle n every sequential module receives a new informative event on each input port and emits an informative event on each output port, the latter being the result of the processing based on the informative events received up to cycle $(n-1)$. Hence, no stalling events travel between the modules at the specification level. However, the presence of relay stations in the final implementation does introduce stalling events: in particular, the initialization value stored inside each relay station is a stalling event. According to the latency insensitive protocol, if a module receives one (or more) stalling event at a given clock cycle it means that misses one (or more) data item to perform the computation and, therefore, it is forced to stall, generating a new stalling event for each output port (meanwhile the informative events received on the other input ports are stored in the shell queues). This mechanism does not affect the overall performance if the design does not present any feedback path between the sequential modules, but obviously this a condition too strong to demand. In [5] Carloni *et al.* do not present an analysis of the impact of their ideas on the system performance. In the following section we propose a formal model to analyze the properties of a latency insensitive system, which allows us to define the notion of *recycling* as an elegant way to capture the latency variations of the communication channels and to compute exactly the final throughput of the system.

4 Latency Insensitive System Graphs

In this section, we formally introduce *latency insensitive system graphs (lis-graphs)* as a way to model the structure of a latency insensitive system, and we give the notion of lis-graph behavior, which allows us to capture their communication and synchronization properties.

Definition 4.1 *A* lis-graph $G = (s,t,V,A,w)$ *is a weighted directed connected graph* (V,A,w)*, where* $w(a_i) \in \mathbf{Z}^{*2}$ *for each arc* $a_i \in A$*, with two special nodes: a* source $s \in V$ *with* 0 *indegree and* 1 *outdegree, and a* sink $t \in V$ *with* 0 *outdegree and* 1 *indegree.*

[2] $\mathbf{Z}^*$ denotes the set of non-negative integers.

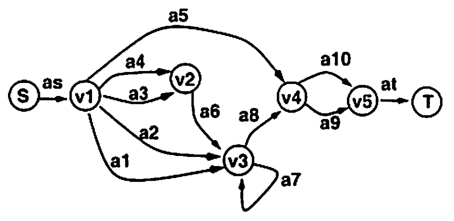

Figure 4: Lis-graph for the MAC of Figure 3.

Definition 4.2 *Given a lis-graph $G = (s,t,V,A,w)$, the single arc a_s leaving source s is called* source arc, *while the single arc a_t entering t is called* sink arc. *The* core *of a lis-graph G is the directed graph $(V \setminus \{s,t\}, A \setminus \{a_s,a_t\}, w)$.*

Using lis-graphs we can focus on the structure of a latency insensitive system, without getting lost into the details of the logic inside each module. Every lis-graph node (but source and sink) is associated to a pearl/shell pair, while the weights on the arcs denote the amount of relay stations on the corresponding channels. The source and the sink model the interaction of the system with the environment inside which it operates. As we know from the previous section, the initialization value stored inside inside each relay station is a stalling event, while the output register of each module is initialized with an informative event according to the synchronous specification of the system. Therefore, an arc a_i with weight $w(a_i) = 3$ indicates that during the first cycle of system operation there are 3 stalling event occupying the 3 buffering spots on the corresponding channel. At the subsequent cycle the first stalling event will be withdrawn by the receiving module M_r at the end of the channel, while the first informative event will be put on the channel by the sending module M_s: this informative event will be read by M_r only 3 cycles later, after the remaining two stalling events.

The fact that changing the number of relay stations on some channels does not change the functionality of the system motivates the following definition of lis-graph equivalence.

Definition 4.3 *Two lis-graphs $G = (s,t,V,A,w)$ and $G' = (s,t,V,A,w')$ differing only for some weights are said* structurally equivalent, *or, simply,* equivalent *($G \equiv G'$). The* reference lis-graph *of a class of equivalent lis-graphs is the graph $G_{ref} = (s,t,V,A,w_{ref})$ s.t. $\forall a_i \in A, (w_{ref}(a_i) = 0)$.*

Example 4.1 The graph of Fig. 3 represents the RTL structure of the MAC circuit of Fig. 2. This graph is also the core of lis-graph $G = (s,t,V,A)$ of Fig. 4, with $V = \{s, v_1, v_2, v_3, v_4, v_5, t\}$ and $A = \{a_s, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_t\}$, which is obtained adding the source and the sink vertices. □

Two distinct arcs of a lis-graphs which respectively enter and leave the same vertex are in a dependency relation.

Definition 4.4 *Given a lis-graph $G = (s,t,V,A,w)$, for all vertices $u_1, u_2, v \in V$, arcs $a_1 = (u_1, v)$ and $a_2 = (v, u_2)$ are in a dependency relation $a_1 \leq_d a_2$ (a_1* depends on *a_2). If and only if $a_1 \leq_d a_2$, a_1 is a predecessor of a_2 and a_2 is a successor of a_1. Given an arc a, $\mathcal{PRED}(a)$ and $\mathcal{SUCC}(a)$ denote respectively the sets of predecessors and successors of a. Obviously, $\mathcal{PRED}(a_s) = \mathcal{SUCC}(a_t) = \emptyset$ holds for any lis-graph.*

Example 4.2 Referring to the lis-graph of Fig. 4, we have $a_3 \leq_d a_6$ and $a_5 \leq_d a_9$. Then, $\mathcal{PRED}(a_6) = \{a_3, a_4\}$, while $\mathcal{SUCC}(a_6) = \{a_7, a_8\}$. Further, a self-loop $a = (v,v)$ depends on all arcs entering v (including itself), while all the arcs leaving v depend on it. In Fig. 4, this is the case of $a_7 = (v_7, v_7)$, for which $\mathcal{PRED}(a_7) = \{a_1, a_2, a_6, a_7\}$ and $\mathcal{SUCC}(a_7) = \{a_7, a_8\}$. □

We are not interested in capturing the particular value of a signal traveling on a wire from a module to another at a given clock cycle, but we want to know whether that signal represents an informative event or a stalling event. The progressive trace of a signal (associated to a channel represented by an arc in the lis-graph) is used to represent an interleaved stream of informative events and stalling events: the value j of a natural number in a trace denotes the ordinal of the j-th informative event while the τ symbol denotes a stalling event.

Definition 4.5 *Let s be a signal with L informative events, let K be the index of the last informative event, and let $\mathcal{T} \subseteq \mathbb{N}$ be the set of indices of the stalling events between 1 and K, where $\mathbb{N}$ is the set of natural numbers. The* progressive trace, *(or, simply,* trace*) σ of* signature *L and* co-signature set *(or, simply,* co-signature*) $\mathcal{T}$ is an infinite sequence of symbols of $\mathbb{N} \cup \{\tau\}$ s.t. the n-th term is*

$$\sigma_n = \begin{cases} \tau & \text{if } s_n = \tau \\ j & \text{if } s_n \text{ is the } j\text{-th element in } s \text{ s.t. } s_n \neq \tau \end{cases}$$

Given a progressive trace σ, its signature is denoted as $\|\sigma\|$, while its co-signature set *is denoted as $\langle\sigma\rangle$.*

Example 4.3 Consider arc a_7 of the lis-graph of Fig 4. This arc represents the feedback path of module M_3 in the MAC of Fig. 2. Consider the behavior reported in Table 1 and observe particularly the values of signal s^7 associated to *regC*. Now, assume that a latency insensitive implementation of the MAC presents only one relay station, placed exactly in the middle of this feedback path. Hence $w(a_7) = 1$, while $w(a_i) = 0$ for $i \neq 7$. Since the initialization value stored in any relay station is τ, the sequence of events on the

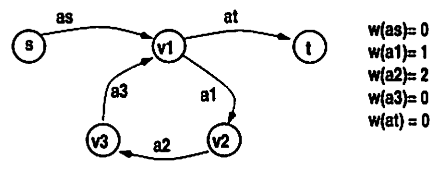

Figure 5: The Lis-graph of Example 4.4.

$$\begin{array}{l}
\sigma(a_s) = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ \tau\ \tau \ldots \\
\sigma(a_1) = \tau\ 1\ 2\ \tau\ \tau\ 3\ \tau\ 4\ 5\ \tau\ \tau\ 6\ \tau\ 7\ 8\ \tau\ \tau\ 9\ \tau\ 10\ 11\ \tau\ \tau\ \tau\ \tau \ldots \\
\sigma(a_2) = \tau\ \tau\ 1\ \tau\ 2\ 3\ \tau\ \tau\ 4\ \tau\ 5\ 6\ \tau\ \tau\ 7\ \tau\ 8\ 9\ \tau\ \tau\ 10\ \tau\ 11\ \tau\ \tau \ldots \\
\sigma(a_3) = 1\ \tau\ \tau\ 2\ \tau\ 3\ 4\ \tau\ \tau\ 5\ \tau\ 6\ 7\ \tau\ \tau\ 8\ \tau\ 9\ 10\ \tau\ \tau\ 11\ \tau\ \tau\ \tau \ldots \\
\sigma(a_t) = 1\ 2\ \tau\ \tau\ 3\ \tau\ 4\ 5\ \tau\ \tau\ 6\ \tau\ 7\ 8\ \tau\ \tau\ 9\ \tau\ 10\ 11\ \tau\ \tau\ \tau\ \tau \ldots
\end{array}$$

Figure 6: Behavior β of Example 4.4, having $\|\beta\| = 11$.

feedback path is $s^7 = \tau\, z_1\, \tau\, z_2\, \tau\, z_3\, \tau\, z_4\, \tau\, b_1\, \tau\, z_6\, \tau\, z_7\, \tau\, z_8\, \tau\, z_9\, \tau\, \tau \ldots$ The corresponding progressive trace, having signature $\|\sigma\| = 10$ and and co-signature $\langle\sigma\rangle = \{1,3,5,7,9,11,13,15,17\}$ is $\sigma = \tau\, 1\, \tau\, 2\, \tau\, 3\, \tau\, 4\, \tau\, 5\, \tau\, 6\, \tau\, 7\, \tau\, 8\, \tau\, 9\, \tau\, \tau \ldots$ The subtrace of σ from the 7-th to the 12-th term is $\sigma_{7,12} = \tau\, 4\, \tau\, 5\, \tau\, 6$ and its signature is $\|\sigma_{7,12}\| = 5$. □

Only after receiving at each input channel the j-th informative event, a module is able to produce (at the next clock cycle) the $(j+1)$-th informative event for each of its output channels. If this is not the case, it means that one or more input channels present a stalling event. Therefore, the module stalls, "consuming" the stalling events, producing output events on all output channels, and storing those informative events which can not be processed in the shell queues. To capture the essence of this mechanism we use the notion of lis-graph behavior.

Definition 4.6 *Given a lis-graph $G = (s,t,V,A,w)$, a* trace assignment $(\sigma(a_1),\ldots,\sigma(a_{|A|}))$ *is a tuple of progressive traces in one-to-one correspondence with the arcs of A. For each arc $a_i \in A$, $\sigma(a_i)$ denotes the progressive trace associated to a_i.*

Definition 4.7 *A* behavior β *of a lis-graph $G = (s,t,V,A,w)$ is a trace assignment $(\sigma(a_1),\ldots,\sigma(a_{|A|}))$ s.t. $\forall i \in |A|$ the n-th term of $\sigma(a_i)$ is*

$$\sigma_n(a_i) = \begin{cases} \tau & \text{if } n \in [1, w(a_i)] \\ 1 & \text{if } n = w(a_i)+1 \\ (\|\sigma_{n-1}(a_i)\| + 1) & \text{if } (n \geq w(a_i)+2) \text{ and} \\ & (\|\sigma_{n-1}(a_k)\| < L) \text{ and} \\ & \forall a_k \in \mathcal{PRED}(a_i) \\ & (\|\sigma_{n-w(a_i)-1}(a_k)\| \geq \|\sigma_{n-1}(a_i)\|) \\ \tau & \text{otherwise} \end{cases} \tag{1}$$

where $L \in \mathbb{N}$ is called the behavior signature *(also denoted as $\|\beta\|$). The set of behaviors of G is denoted as $\mathcal{B}(G)$.*

Example 4.4 Given the lis-graph of Fig. 5, the behavior $\beta = (\sigma(a_s),\sigma(a_1),\sigma(a_2),\sigma(a_3),\sigma(a_t))$ with signature $\|\beta\| = 11$ is illustrated in Fig. 6. Notice that trace $\sigma(a_s)$, being associated to the source arc a_s, is independent from the other traces. Observe that $w(a_1) = 1$ and $w(a_2) = 2$, the first symbol of $\sigma(a_1)$ as well as the first two symbols of $\sigma(a_2)$ are τ. Then, consider that $\mathcal{PRED}(a_2) = \{a_1\}$ and $\mathcal{PRED}(a_3) = \{a_2\}$. For all $n \in \mathbb{N}$, the n-th term of trace $\sigma(a_2)$ depends on the (n-3)-th term of $\sigma(a_1)$ since $w(a_2) = 2$ and the n-th term of trace $\sigma(a_3)$ depends on the (n-1)-th term of $\sigma(a_2)$ since $w(a_3) = 0$. Finally, arc a_t and a_1 have a common set of predecessors containing a_s and a_3. However, $\sigma(a_t)$ and $\sigma(a_1)$ depend "practically" only on trace $\sigma(a_3)$, since $\sigma(a_3)$ is constantly lagging behind $\sigma(a_s)$ with respect to the amount of informative events seen up to any instant. In particular, the n-th term of trace $\sigma(a_t)$ depends on the (n-1)-th term of $\sigma(a_3)$ because $w(a_t) = 0$, and the n-th term of trace $\sigma(a_1)$ depends on the (n-2)-th term of $\sigma(a_3)$ because $w(a_1) = 1$. □

The definition of lis-graph behavior is reminiscent of the *firing semantic* of an *event graph*, a common abstraction used to model discrete event systems [2]. In fact, the presence of either τ or a natural number on an arc $a_i = (v_j, v_k)$ can be seen as the result of the processing completed by vertex v_j in accordance with the *lis-graph firing semantic* expressed by the following rules:

- *Independence Rule*: Every vertex v_j fires the first informative event (corresponding to natural number 1) independently. However, a trace associated to arc $a_i = (v_j, v_k)$ with weight $w(a_i)$ starts with $w(a_i)$ stalling symbols and only the $(w(a_i)+1)$-th symbol corresponds to the value 1 independently "produced" by v_j.

- *And-Causality Rule*: Every vertex v_j fires the n-th event only after the $(n-1)$-th event has appeared on all arcs entering v_j.

This firing semantic is equivalent to Definition 4.7 and suggests that a lis-graph models a cyclic system [2]. Obviously, this is true as far as the core of the lis-graph contains at least one cycle among some of its nodes (in the most trivial case the only cycle could be simply a self-loop as in Figure 4). For cyclic systems, the *cycle time* T_i of arc a_i is defined as

$$T_i = \lim_{n \to \infty} \frac{t_i(n)}{n} \tag{2}$$

where $t_i(n)$ denotes the time at which trace $\sigma(a_i)$ associated to arc a_i presents the n-th informative event (i.e. natural number n). Since the system is cyclic, for all arcs a_i, $T_i = T$ and the previous equation gives the *cycle time* T of the entire system. The cycle time is equal to the inverse of the *system throughput* ϑ (i.e., the rate at which informative events appear on the channels) and, represents the key performance metric for the system: for all informative events $n \geq 1$, the difference $|(t_i(1) + T \cdot n) - t_i(n)|$ is bounded [4, 9, 22] We naturally refer to cycle time $T(G)$ and throughput $\vartheta(G)$ of a lis-graph G, meaning the cycle time and the throughput of the system model by G. Furthermore, the cycle time coincides with the *maximum cycle mean* of lis-graph G, defined as

$$\lambda(G) = \max_{\forall C \in G} \lambda(C) = \max_{\forall C \in G} \left(\frac{w(C) + |C|}{|C|}\right) \tag{3}$$

where $\lambda(C)$ is the *cycle mean* of a cycle C of G, $w(C)$ is the sum of the weights of the arcs on C, and $|C|$ is equal to the number of arcs on C [3]. A cycle whose mean coincides with the maximum cycle mean is said *critical*. The maximum cycle mean can be found solving the *Maximum Cycle Mean Problem*, for which many algorithms have been proposed, dating back to Karp's Algorithm [15] [4].

Example 4.5 Lis-graph $G = (s,t,V,A)$ of Fig. 5 contains one cycle $C = (a_1, a_2, a_3)$ with $|C| = 3$. Since $w(a_1) = 1, w(a_2) = 2, w(a_3) = 0$, the maximum cycle mean of G is $\lambda(G) = (\frac{w(C)+|C|}{|C|}) = \frac{(3+3)}{3} = 2$. Then, the throughput is $\vartheta(G) = \frac{1}{\lambda(G)} = \frac{1}{2}$. □

If no cycles are present in the lis-graph core, both cycle time and throughput are equal to 1. This confirms the intuition, because lis-graphs with no cycles represent pipelined systems with no feedback paths: hence, for any possible weight assignment on the arcs, there exists a natural number k after which, all τ in the system have been ejected through the sink t and only informative events travel on the channels, thus delivering the best possible communication throughput. If the lis-graph core contains only one strongly connected component, all its cycles can be detected in $O((|V| + |A|) \cdot (K+1))$ operations, where K is the number of cycles in the graph [23]. Finally, the most general case is when the lis-graph core contains more than one strongly-connected component: being a directed graph, a lis-graph can be efficiently partitioned in strongly-connected components using Tarjan's Algorithm [25]. Then, the maximum cycle mean can be determined for each strongly connected component and the largest of these means is clearly the maximum cycle mean for the lis-graph.

5 Recycling

As discussed in Section 3, moving from the specification of a system to the final layout, the latency of some communication channels may be higher than the clock period, due to the length and the delay of the wires implementing them. In particular, from the analysis of the layout we can determine for each wire what is the smallest multiple of the desired clock period which is larger than its delay. If this multiple is greater than 1 then the wire is marked *illegal*. The presence of illegal wires in the layout implies that the final implementation is not correct. To capture this concept in our model we attach to each arc of the corresponding lis-graph a value denoted as the length of the arc.

Definition 5.1 *A lis-graph $G = (s,t,V,A,w,l)$ is* annotated iff *a* length $l(a_i) \in \mathbb{N}$ *is associated to each arc $a_i \in A$.*

Definition 5.2 *Let $G = (s,t,V,A,w,l)$ be an annotated lis-graph. G is* legal iff $\forall a_i \in A$, $(w(a_i) \geq l(a_i) - 1)$. *An arc $a_j \in A$ s.t. $(w(a_j) < l(a_j) - 1)$ is an* illegal arc.

All the definitions given for lis-graphs are naturally extended to annotated lis-graphs.

Now, thanks to the latency insensitive methodology, we can correct the final layout by introducing the necessary amount of relay stations to make sure that the delay of each wire is less than the desired clock period. Similarly, a non-legal annotated lis-graph G can be transformed into an equivalent legal annotated lis-graph G' by simply incrementing the weights of its arcs by the appropriate quantity. Since $G \equiv G'$, for all behaviors of G there is a correspondent equivalent behavior of G'. This transformation is called *recycling*.

Lemma 5.1 *Let $G = (s,t,V,A,w,l)$ be a non-legal annotated lis-graph. A legal annotated lis-graph $G' = (s,t,V,A,w',l)$ equivalent to G is obtained from G by adding the quantity $\Delta w(a_i) = l(a_i) - 1 - w(a_i)$ to the weight of arc a_i, where $i \in [1, |A|]$.*

[3]The usual definition of the cycle mean of a cycle C is $\lambda(C) = \frac{w(C)}{|C|}$ as in [2, 15]. We added the term $|C|$ to the numerator, because the firing of a lis-graph node takes one time unit, modeling the fact that every module in a latency insensitive system is sequential.

[4]See [9] for a survey of the proposed algorithms.

$\sigma(a_s)$	$=$	$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_1)$	$=$	$\tau\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_2)$	$=$	$\tau\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_3)$	$=$	$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_4)$	$=$	$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_5)$	$=$	$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_6)$	$=$	$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_7)$	$=$	$1\ \tau\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_8)$	$=$	$\tau\ \tau\ 1\ \tau\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_9)$	$=$	$1\ \tau\ \tau\ 2\ \tau\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_{10})$	$=$	$1\ \tau\ \tau\ 2\ \tau\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots$
$\sigma(a_t)$	$=$	$1\ 2\ \tau\ \tau\ 3\ \tau\ 4\ 5\ 6\ 7\ 8\ 9\ldots$

Figure 7: Case (a) of Ex. 5.1: $\vartheta(G') = 1, (\Delta\vartheta(G,G') = 0)$.

Proof. For all $i \in [1,|A|]$, $w'(a_i) = w(a_i) + \Delta w(a_i) = l(a_i) - 1$. Hence, by definition 5.2, G' is legal. Further $G' \equiv G$, since G and G' differ only for their weights. □

Although recycling is an easy way to correct the final implementation of the system and satisfy the timing constraints imposed by the clock, it doesn't come without a cost. In fact, augmenting the weights of some arcs of G (i.e., inserting memory elements on the communication channels) may increase the maximum cycle mean of G (i.e., increase the cycle time of the system modeled by G and, symmetrically, decrease its throughput). This is always the case if any arc a_i, whose weight $w(a_i)$ is augmented, belongs to the set of critical cycles of G, simply because the numerator in Equation 3 increases by the quantity $\Delta w(a_i)$ while the denominator remains unaffected. It may also happen that increasing the weights of some arcs makes a non-critical cycle of G becoming a critical cycle of G'. In any case, after completing the recycling transformation, we may exactly compute the consequent *throughput degradation* $\Delta\vartheta(G,G') = \vartheta(G) - \vartheta(G')$, using one of the following methods:

- solve the *Maximum Cycle Mean Problem* for graph G', and simply set $\vartheta(G') = \frac{1}{\lambda(G')}$;
- after establishing the set $\mathcal{A}$ of cycles having at least one arc whose weight has been augmented, increment the cycle mean of each element C of $\mathcal{A}$ by the quantity $\frac{1}{|C|} \cdot \sum_i \Delta w(a_i)$, where a_i are the arcs of C which have been corrected. Let λ^* be the maximum among all these cycle means, then

$$\Delta\vartheta(G,G') \quad = \quad \begin{cases} 0 & \text{if } \lambda^* \le \lambda(G) \\ \frac{\lambda^* - \lambda(G)}{\lambda(G)\cdot\lambda^*} & \text{otherwise} \end{cases}$$

Example 5.1 Consider the lis-graph $G = (s,t,V,A,w)$ of Fig. 4, representing the MAC of Fig. 2. Let us study the following two cases:

(a) Assume that all arcs have zero weights, i.e that the vector of weights associated to $(a_s,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8,a_9,a_{10},a_t)$ is $w = (0,0,0,0,0,0,0,0,0,0,0,0)$. Now, assume that after completing the implementation of the MAC, the wires associated to arcs a_1, a_2 and a_8 are illegal wires, e.g. the first two have a delay between 1 and 2 clock periods, while the third one has a delay between 2 and 3 clock periods. Hence, we annotate the lis-graph with the vector of arc lengths $l = (1,2,2,1,1,1,1,1,3,1,1,1)$. Then, to avoid illegal arcs we must insert at least one relay station on both arc a_1 and a_2 and two relay stations on arc a_8. In other words we transform the weight vector into $w' = (0,1,1,0,0,0,0,0,2,0,0,0)$. Not only the final design is functionally equivalent, but the performance of the system remains unaffected! In fact, the added relay stations will contain τ symbols as initial values and will provoke a certain amount of stalling in the down-link nodes (in particular, node v_2 will stall once, while node v_3 and v_4 will stall twice), but after few instants the system will reach its steady state processing data with throughput $\vartheta(G') = 1$. The corresponding lis-graph behavior is illustrated in Fig. 7.

(b) On the other case, assume that only the wire associated to a_7 has a delay larger than 1 clock period, particularly between 2 and 3 clock periods. Then we legalize the annotated lis-graph G' by setting only $w(a_7) = 2$, while the other weights could stay equal to 0. Since $w(a_7)$ belongs to a cycle of the lis-graph (it is a self-loop!), the system throughput becomes $\vartheta(G') = \frac{1}{3}$, with degradation $\Delta\vartheta(G,G') = 66\%$, as illustrated in Fig. 8. □

Observing case (a) of the previous example, we see that the shell encapsulating the module corresponding to node v_3 must have a queue of length 1 at the input port of the feedback path associated to a_7. Similarly, the shell corresponding to node v_4 must have a queue of length 2 at the input port of the path associated to a_5. All the other shells do not need to have any queue. Since the arcs having non-zero weight are not part of any cycle (i.e. no relay stations are inserted on feedback paths), this queues are necessary only for the few clock cycles elapsing before the system reaches its steady state. Instead, for case (b), we see that we would need to insert *infinite queues* at the input ports of the paths associated to a_2,a_4,a_5 and a_6. Since infinite queues can not be realized in practice, the only choice is to adapt the throughput of the rest of the system to the one of cycle a_7. This is why a critical cycle dictates the throughput of the overall system. In this particular case to "slow down" the rest of the system is sufficient to reduce the input throughput. From a theoretical point of view, this can be achieved by simply adding a self-loop to the source node and putting a couple of extra relay stations on it. Obviously,

$$\begin{array}{lcl}
\sigma(a_s) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_1) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_2) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_3) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_4) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_5) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_6) & = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ldots \\
\sigma(a_7) & = & \tau\ \tau\ 1\ \tau\ \tau\ 2\ \tau\ \tau\ 3\ \tau\ \tau\ 4\ \tau\ \tau\ 5\ \tau\ \tau\ 6\ \tau\ \tau\ 7\ \tau\ \tau\ 8\ \tau\ \tau\ 9\ldots \\
\sigma(a_8) & = & 1\ \tau\ \tau\ 2\ \tau\ \tau\ 3\ \tau\ \tau\ 4\ \tau\ \tau\ 5\ \tau\ \tau\ 6\ \tau\ \tau\ 7\ \tau\ \tau\ 8\ \tau\ \tau\ 9\ldots \\
\sigma(a_9) & = & 1\ 2\ \tau\ \tau\ 3\ \tau\ \tau\ 4\ \tau\ \tau\ 5\ \tau\ \tau\ 6\ \tau\ \tau\ 7\ \tau\ \tau\ 8\ \tau\ \tau\ 9\ldots \\
\sigma(a_{10}) & = & 1\ 2\ \tau\ \tau\ 3\ \tau\ \tau\ 4\ \tau\ \tau\ 5\ \tau\ \tau\ 6\ \tau\ \tau\ 7\ \tau\ \tau\ 8\ \tau\ \tau\ 9\ldots \\
\sigma(a_t) & = & 1\ 2\ 3\ \tau\ \tau\ 4\ \tau\ \tau\ 5\ \tau\ \tau\ 6\ \tau\ \tau\ 7\ \tau\ \tau\ 8\ \tau\ \tau\ 9\ldots
\end{array}$$

Figure 8: Case (b) of Ex. 5.1: $\vartheta(G') = \frac{1}{3}, (\Delta\vartheta(G,G') = 66\%)$.

in practice, this corresponds to slow down the environment inside which the system operates. In general, if lis-graph G contains more than one strongly connected component, the recycling transformation must be decomposed in two steps:

1. **(legalization)**. After deriving the the annotated lis-graph G', legalize it by augmenting the weights of the wires by the appropriate quantity, as specified in Lemma 5.1;

2. **(equalization)**. Compute the maximum throughput $\vartheta(S_k) = \frac{a_k}{b_k} \in]0,1]$ which is sustainable by each strongly connected component $S_k \in G'$ (recall that $\vartheta(S_k)$ is equal to the inverse of the maximum cycle mean $\lambda(S_k)$. Equalize the throughputs by adding a quantity $n_k \in \mathbf{Z}^*$ to the denominator of each $\vartheta(S_k)$. This corresponds to augment by a quantity n_k the weight of the critical cycle $C_k \in S_k$, i.e. to distribute n_k *extra relay stations* among the corresponding paths. To find the quantities n_k is necessary to solve the optimization Problem 8.2 specified in the Appendix.

As Example 5.1 illustrates, the key to avoid big performance losses while recycling a lis-graph G is to avoid being forced to augment the weights of those arcs which belong to a critical cycle C of G. Furthermore, as Equation 3 suggests, for the same $w(C)$, the smaller is the cardinality $|C|$ of the cycle the worse is the loss in throughput for G. The worst case is clearly represented by self-loops.

These considerations must be kept in mind while partitioning the functionality of the system in tasks to be assigned to different IP cores. It is true that the latency insensitive methodology guarantees that no matter how bad is the final implementation of the system (in terms of lengths of the wires realizing the communication architecture), it is always possible to fix it by adding relay stations. Still, to achieve good performance, one should adopt a design strategy based on the following guidelines:

- all modules should put comparable timing constraints on the global clock (i.e. the delays of the longest combinatorial paths inside each module should be similar);

- modules whose corresponding lis-graph nodes belong to the same cycle should be kept close while deriving the final implementation.

In general, the insertion of relay stations should be completed by an automatic tool as part of the physical design process (similarly to the *buffer insertion* techniques available in current design flows [7]). In fact, the real advantage of the latency insensitive methodology is the new freedom offered to "move around the latency" once the final implementation has been derived: not only problematic layouts can be fixed without changing the design of the individual modules, but also latency/throughput trade-offs can be explored and optimized up to the late stages of the design process. Meanwhile, the traditional design methodology, which relegates this exploration at the floorplan level, is losing effectiveness, because the interaction among the components of a modern system on silicon is becoming too complex, as the following case study illustrates.

6 Case Study: MPEG-2 Video Encoder

Examples of SOC are not abundant in literature. We have chosen an MPEG-2 Video Encoder [3] as an example for illustrating our method. This encoder has been first (in 1996) implemented as a chip-set with two integrated circuits, and three years later as a single chip [11, 12]. Figure 9 illustrates its functional diagram. We assume that this diagram corresponds also to the block diagram of the final implementation and that, at each clock cycle, every block in the pipeline provides a new informative data item to the down-link block: in other words, at every cycle, on each arc we have either a stalling symbol or a data value which is not a *don't care* for the receiving block. We understand that this may not always be the case at this level of granularity, because, for instance, the *Quantizer* may take more than one clock cycle to produce a result which can trigger a new computation of the down-link *Inverse Quantizer*. Still, the presence of these types of *don't cares* may only help from the performance point of view and, in any case, to address the relationships between them and the latency insensitive protocols goes beyond the scope of this paper. The lis-graph G for the MPEG-2 Video Encoder is reported in

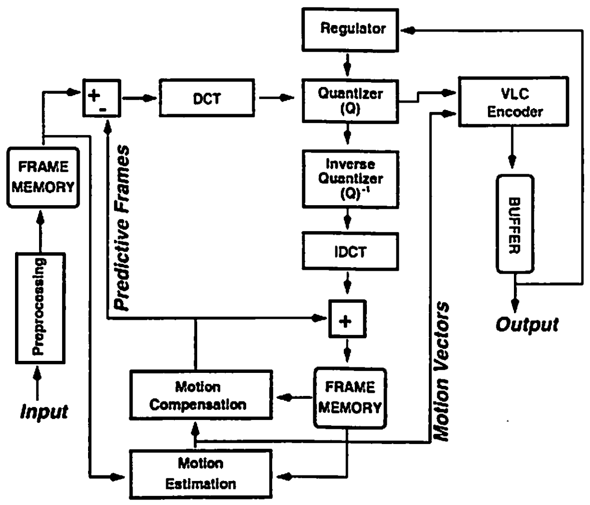

Figure 9: Block Diagram of a MPEG-2 Video Encoder.

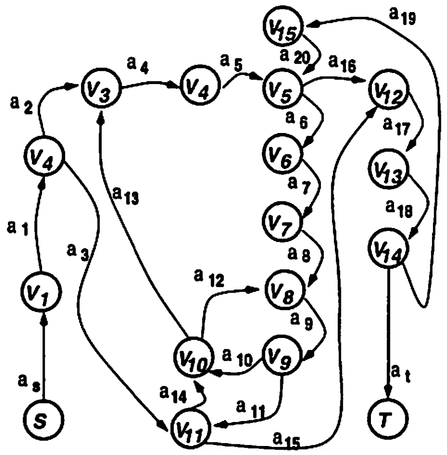

Figure 10: The Lis-graph of the MPEG-2 Video Encoder.

Fig. 10 and contains 6 distinct cycles:

$$
\begin{aligned}
C_1 &= \{a_9, a_{10}, a_{12}\} \\
C_2 &= \{a_9, a_{11}, a_{14}, a_{12}\} \\
C_3 &= \{a_{16}, a_{17}, a_{18}, a_{19}, a_{20}\} \\
C_4 &= \{a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{13}\} \\
C_5 &= \{a_4, a_5, a_6, a_7, a_8, a_9, a_{11}, a_{14}, a_{13}\} \\
C_6 &= \{a_6, a_7, a_8, a_9, a_{11}, a_{15}, a_{17}, a_{18}, a_{19}, a_{20}\}
\end{aligned}
$$

Most arcs are common to more than one cycle, e.g. a_9 is contained in C_1, C_2, C_4, C_5, and C_6 ,but others are contained only in one cycle, e.g. a_{19} is part only of C_6. Finally, some arcs, such as a_3 are not contained in any cycle. We already know that increasing the weight of these arcs does not affect the system performance. But, what about the ones belonging to one or more cycles? Can we compute the degradation in performance in advance? As a matter of fact, yes. Figure 11 reports the results of the analysis that can be done based on the lis-graph model. The six curves in the chart are associated to the above cycles and their shapes should be interpreted as follows: each point of curve C_i shows the amount of system throughput degradation which is detected after setting the total sum $w(C_i)$ of the weight of the arcs of C_i equal to integer x, with $x \in [0,20]$. Obviously, the underlying assumption is that C_i is a critical cycle of G, and this limits the choice of those arcs of C_i whose weights can be augmented. For example, assume that $w(C_2) = 5$, as a result of summing $w(a_9) = 4, w(a_{11}) = 1, w(a_{14}) = 0$, and $w(a_{12}) = 0$. In this case C_2 is definitely not a critical cycle. In fact, its cycle mean is $\lambda(C_2) = \frac{5+4}{4} = \frac{9}{4} = 2.250$, while, even if $w(a_{10}) = w(a_{12}) = 0$, cycle C_1, with $w(C_1) = w(a_9) = 4$, has a larger cycle mean, exactly $\lambda(C_1) = \frac{4+3}{3} = \frac{9}{4} = 2.333$.

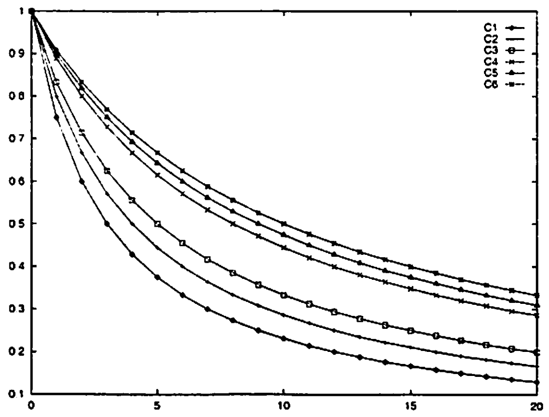

Figure 11: Analysis of Throughput Degradation for the MPEG-2.

As Figure 11 confirms, the best way to avoid losing performance is to increase the weights of those arcs that belong to bigger cycles. While performing the recycling transformation this may or may not be possible: for example if the length of arc a_{10} is big, cycle C_1 will ultimately dictate the system throughput. However, in general the latency insensitive methodology allows us to push around relay stations without need of re-designing any module. This, may be useful for example to reduce the length of an arc such as a_9, which belongs to both big and small cycles, while increasing in exchange the length of a_{19}, which is only part C_6: assuming that, before re-balancing, $l(a_9) = 3$ and $l(a_{19}) = 1$, while, after re-balancing, they become respectively 1 and 3, and assuming that all other arcs have unit lengths, we have a final throughput of $\frac{10}{12} = 0.833$ instead of $\frac{3}{5} = 0.6$, a 38% improvement.

7 Conclusions

A methodology based on the theory of latency insensitive protocols has been recently proposed in literature [5]: a latency insensitive system is a synchronous digital system composed by functional modules exchanging data on point-to-point communication channels in accordance with a communication protocol that allows them to operate independently from the latencies of the channels. As a consequence, a hardware implementation of the system can be automatically synthesized such that its functional behavior is robust with respect to large variations in wiring delays between modules. However, the method does not guarantee the same robustness for the performance of the design, which indeed may experience a notable degradation. In [5], the authors do not address this problem nor they suggest a technique for analyzing the latency/throughput trade-offs. We have presented the definition of *lis-graph* as a formal model to analyze the properties of a latency insensitive system. The model allows us to specify the notion of *recycling* as a rigorous way to capture the latency variations of the communication channels and to compute exactly the final throughput of the system. By discussing a case study (an industrial MPEG-2 Video Encoder) we have illustrated how the present work enables the exploration of latency/throughput trade-offs at any stages of the design process, thus facilitating the integration of pre-designed IP cores on a single chip. Future work will focus on the application of these concepts to the optimization of the computation/communication trade-offs that arise while designing software compilers for those machines which have a communication architecture with variable latency [1].

Acknowledgments

This research was supported partially by Cadence Berkeley Laboratories, the SRC. and the GSRC/Marco center at Berkeley. The authors would like to thank Luciano Lavagno, Ken McMillan, Richard Newton, Jan Rabaey and Alexander Saldanha for their support and precious feedback. Discussions with Fernando De Bernardinis, Roberto Passerone, Claudio Pinello, Mukul Prasad, Marco Sgroi and Tiziano Villa are gratefully acknowledged.

References

[1] A. Agarwal. Raw Computation. *Scientific American*, 281(2):60–63, August 1999.

[2] F. Bacelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. Wiley, New York, 1992.

[3] Vasudev Bhaskaran. *Image and video compression standards : algorithms and architectures*. Kluwer Academic Publishers, Boston, MA, 1997.

[4] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.

[5] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Latency Insensitive Protocols. In *Proc. of the 11th Intl. Conf. on Computer-Aided Verification*, pages 123–133, July 1999.

[6] J. Cong. Challenges and Opportunities for Design Innovations in Nanometer Technologies. In *SRC Design Sciences Concept Paper*, December 1997.

[7] J. Cong, L. He, K.Y. Khoo, C.K. Koh, and Z. Pan. Interconnect Design for Deep Submicron ICs. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 478–585. IEEE, November 1997.

[8] D. Matzke. Will Physical Scalability Sabotage Performance Gains? *IEEE Computer*, 8(9):37–39, September 1997.

[9] A. Dasdan and Rajesh K. Gupta. Faster Maximum and Minimum Mean Cycle Algorithms for System-Perofrmance Analysis. *IEEE Transactions on Computer-Aided Design*, 17(10):889–899, October 1998.

[10] E. Waingold *et. al.* Baring It All to Software: Raw Machines. *IEEE Computer*, 30(9):86–93, September 1997.

[11] M. Ikeda *et. al.* A Hardware/Software Concurrent Design for Real-Time SP@ML MPEG2 Video-Encoder Chip Set. In *Proc. European Design and Test Conf.*, pages 320–326, March 1996.

[12] M. Ikeda *et. al.* SuperENC: MPEG-2 Video Encoder Chip. *IEEE Micro*, 19(4):56–65, July 1999.

[13] R. J. Higgins. *Digital Signal Processing in VLSI.* Analog Devices Technical Reference Books. Prentice Hall, 1990.

[14] H. Kapadia and M. Horowitz. Using Partitioning to Help Convergence in the Standard-Cell Design Automation Method. In *Proc. of the Design Automation Conf.*, pages 592–597, June 1999.

[15] R. M. Karp. A Characterization of the Minimum Cycle Mean in a Digraph. *Discrete Math.*, 23:309–311, 1978.

[16] M. Horowitz and R. Ho and K. Mai. The Future of Wires. In *Invited Workshop Paper for SRC Conference.* Available at "http://velox.stanford.edu/ ronho/resume.html", May 1999.

[17] M. T. Bohr. Interconnect Scaling - The Real Limiter to High Performance ULSI. *IEEE International Electron Devices Meeting*, pages 241–244, December 1995.

[18] M. T. Bohr. Silicon Trends and Limits for Advanced Microprocessors. *Communication of the ACM*, 41(3):80–87, March 1998.

[19] M.J. Flynn and P. Hung and K.W. Rudd. Deep-Submicron Microprocessor Design Issues. *IEEE Micro*, 19(4):11–13, July 1999.

[20] F. Pogodalla, R. Hersemeule, and P. Coulomb. Fast Prototyping: a System Design Flow for Fast Design, Prototyping and Efficient IP Reuse. In *Proc. of the 7th Intl. Conf. on Hardware/Software Codesign(CODES'99)*, pages 69–73, Rome, Italy, May 1999.

[21] R. Ho and K. Mai and H. Kapadia and M. Horowitz. Interconnect Scaling Implications for CAD. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1999.

[22] C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.

[23] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice Hall, Englewood Cliffs, NJ, 1977.

[24] T. McHale. Electronic Design Automation: Time to Grow Up. *Electronic Business*, June 1999.

[25] R. E. Tarjan. Depth-First Search and Linear Graph Algorithms. In *SIAM J. Comput.*, volume 1, pages 146–160. Society for Industrial and Applied Mathematics, 1972.

```
PairEqualizer $(a,b,a',b')$ {
    /* Find pair $(n,n')$ s.t. $\frac{a}{b+n} = \frac{a'}{b'+n'}$ */
    $n = n' = 0$;  $d = b$;  $d' = b'$
    while $(\frac{a}{d} \neq \frac{a'}{d'})$ {
        if $(\frac{a}{d} > \frac{a'}{d'})$ {
            $d' = d' + n'$
            $n = \lceil \frac{a \cdot d' - a' \cdot b}{a'} \rceil$
        }
        else {
            $d = d + n$
            $n' = \lceil \frac{a' \cdot d - a \cdot b'}{a} \rceil$
        }
    }
    return $(n,n')$
}
```

Figure 12: Algorithm to solve Problem 8.1.

8 Appendix

Let $\mathbf{Z}^*$ denote the set of nonnegative integers, $\mathbf{Z}^+$ the set of positive integers, and $\mathbf{Q}^+$ the set of positive rationale numbers. For all $a,b \in \mathbf{Z}^+, (0 < a \leq b)$, let $\phi_{a,b} : \mathbf{Z}^+ \to \mathbf{Q}^+$ be the infinite sequence whose n-term is $\phi_{a,b}(n) = \frac{a}{b+n}$. Clearly, $\phi_{a,b}$ is decreasing monotone and converging to 0. The following lemma shows that for all pairs (a,b), there exists a positive integer B such that one term of sequence $\phi_{a,b}(n)$ is equal to $\frac{1}{B}$ and, furthermore, that the sequence touches all the values $\frac{1}{B+j}$ for $j = 1,2,\ldots,$ while converging to 0.

Lemma 8.1 $\forall a,b \in \mathbf{Z}^+, (0 < a \leq b), \exists B \in \mathbf{Z}^+, (B \geq \lceil \frac{b}{a} \rceil)$, *s.t.* $\forall j \in \mathbf{Z}^*, \exists \bar{n} \in \mathbf{Z}^* \ (\phi_{a,b}(\bar{n}) = \frac{1}{B+j})$.

Proof. For all $a,b \in \mathbf{Z}^+, (0 < a \leq b)$, and for all $j \in \mathbf{Z}^*$ suppose $\phi_{a,b}(\bar{n}) = \frac{1}{B+j}$. Then, $\bar{n} = a \cdot (B+j) - b$. Now, let $B = \lceil \frac{b}{a} \rceil$, then $a \cdot B$ is the smallest integer greater than (or equal to) b and for all $j \in \mathbf{Z}^+$, $\bar{n}$ is a nonnegative integer. □

Any two sequences $\phi_{a,b}$ and $\phi_{a',b'}$ share an infinite set of common terms. The following lemma provides a pair of equations to find one of these common terms.

Lemma 8.2 $\forall a,b,a',b' \in \mathbf{Z}^+, (0 < a \leq b), (0 < a' \leq b'), \exists \bar{n}, \bar{n}' \in \mathbf{Z}^*$ *s.t.* $\phi_{a,b}(\bar{n}) = \phi_{a',b'}(\bar{n}')$.

Proof. For all $a,b,a',b' \in \mathbf{Z}^*, (0 < a \leq b), (0 < a' \leq b')$, let $\beta = \lceil \frac{b}{a} \rceil$ and $\beta' = \lceil \frac{b'}{a'} \rceil$. Then, apply Lemma 8.1 for the two sequences $\phi_{a,b}, \phi_{a',b'}$ with $B = \max\{\beta, \beta'\}$. Therefore, $\forall j \in \mathbf{Z}^*, \exists \bar{n}, \bar{n}' \in \mathbf{Z}^*$ s.t. $\phi_{a,b}(\bar{n}) = \frac{1}{B+j} = \phi_{a',b'}(\bar{n}')$. □

However, we are interested in finding the first common term between two sequences $\phi_{a,b}$ and $\phi_{a',b'}$. Since the two sequences are decreasing monotone this means that we are looking for the pair of positive integers n,n' s.t. $\phi_{a,b}(\bar{n}) = \phi_{a',b'}(\bar{n}')$ and $n+n'$ is minimum. This problem can be casted as the following instance of the Integer Linear Programming (ILP) problem.

Problem 8.1 Given: Two pairs of nonnegative integers $(a,b),(a',b') \in (\mathbf{Z}^*)^2$ and such that $a \leq b$, $a' \leq b'$.
Minimize: The cost $C = n + n'$ over all nonnegative integers $n,n' \in \mathbf{Z}^*$.

$$\textbf{Subject to}: \quad \frac{a}{b+n} = \frac{a'}{b'+n'} \tag{4}$$

Lemma 8.2 guarantees that this problem has solution, since there exists an infinite number of pairs (n,n') which satisfy constraint (4). Fig. 12 reports a a simple algorithm to solve this problem.

Problem 8.1 can be extended to the case of K pairs of nonnegative integers as follows.

Problem 8.2 Given: K pairs of nonnegative integers $(a_1,b_1),(a_2,b_2),\ldots,(a_K,b_K) \in (\mathbf{Z}^*)^2$
Minimize: The cost $C = \sum_{k=1}^{K} n_k$ over all nonnegative integers $n_k \in \mathbf{Z}^*$.

$$\textbf{Subject to}: \quad \exists R, \ \forall k \in [1,K], \ (\frac{a_k}{b_k + n_k} = R) \tag{5}$$

The solution of this problem can be obtained by applying recursively the procedure in Figure 12 on a binary tree computational structure, where each node corresponds to one call of the procedure. In fact, the final solution is independent from the order adopted to subsequently select pairs of fractions to equalize.