
COPING WITH LATENCY IN SOC DESIGN

LATENCY-INSENSITIVE DESIGN IS THE FOUNDATION OF A CORRECT-BY-CONSTRUCTION METHODOLOGY FOR SOC DESIGN. THIS APPROACH CAN HANDLE LATENCY'S INCREASING IMPACT ON DEEP-SUBMICRON TECHNOLOGIES AND FACILITATE THE REUSE OF INTELLECTUAL-PROPERTY CORES FOR BUILDING COMPLEX SYSTEMS ON CHIPS, REDUCING THE NUMBER OF COSTLY ITERATIONS IN THE DESIGN PROCESS.

Luca P. Carloni

**Alberto L.
Sangiovanni-
Vincentelli**
University of California,
Berkeley

..... As commercial demand for system-on-a-chip (SOC)-based products grows, the effective reuse of existing intellectual-property design modules (also known as *IP cores*) is essential to meet the challenges posed by deep-submicron (DSM) technologies and to complete a reliable design within time-to-market constraints. Originally, IP cores were mostly functional blocks built for previous design generations within the same company. Frequently, today's IP cores are optimized modules marketed as off-the-shelf components by specialized vendors.

An IP core must be both flexible—to collaborate with other modules within different environments—and independent from the particular details of one-among-many possible implementations. The prerequisite for an easy trade, reuse, and assembly of IP cores is the ability to assemble predesigned components with little or no effort. The consequent challenge is addressing the communication and synchronization issues that naturally arise while assembling predesigned components.

We believe that the semiconductor industry will experience a paradigm shift from computation- to communication-bound design: The number of transistors that a signal can

reach in a clock cycle—not the number that designers can integrate on a chip—will drive the design process. The “From computation- to communication-bound design” sidebar discusses this trend. The strategic importance of developing a communication-based design methodology naturally follows. Researchers have proposed the principle of orthogonalization of concerns as essential to dealing with the complexity of systems-on-a-chip design.¹ Here, we address the orthogonalization of communication versus computation—in particular, separating the design of the block functionalities from communication architecture development.

An essential element of communication-based design is the encapsulation of predesigned functional modules within automatically generated interface structures. Such a strategy ensures a correct-by-construction composition of the system. Latency-insensitive design and the recycling paradigm are a step in this direction.

Coping with volatile latency

High-end-microprocessor designers have traditionally anticipated the challenges that ASIC designers are going to face in working with the

From computation- to communication-bound design

According to the "2001 Technology Roadmap for Semiconductors," the historical half-pitch technology gap between microprocessors and ASICs on the one hand and DRAM on the other will disappear by 2005.¹ As we proceed into deep-submicron (DSM) technologies, the 130-nm microprocessor half-pitch in 2002 will shrink to the projected 32-nm length in 2013, allowing the integration of more than 1 billion transistors on a single die. At the same time, the *ITRS* predicts that the on-chip local-clock frequency will rise from today's 1 to 2 GHz, to 19 to 20 GHz. The so-called interconnect problem, however, threatens the outstanding pace of technological progress that has shaped the semiconductor industry. Despite the increase in metal layers and in aspect ratio, the resistance-capacitance delay of an average metal line with constant length is becoming worse with each process generation.^{2,3} The current migration from aluminum to copper metallization is compensating for this trend by reducing the interconnect resistivity. The introduction of low-k dielectric insulators may also alleviate the problem, but these one-time improvements will not suffice in the long run as feature size continues to shrink.⁴

The increasing resistance-capacitance delays combined with the increases in operating frequency, die size, and average interconnect length cause interconnect delay to become the largest fraction of the clock cycle time. In 1997, *Computer* magazine published a study by D. Matzke⁵ con-

taining a gloomy forecast of how on-chip interconnect latency (predicted to soon measure in the tens of clock cycles) will hamper Moore's law. Since then, researchers have fiercely debated the real magnitude of the wire-delay impact and the consequent need for revolutionizing established design methodologies already challenged by the timing-closure problem.

The debate's core has been about the scaling properties of interconnect wires relative to gate scaling. On one side, some researchers share an optimistic view based on the fact that wires that scale in length together with gate lengths offer approximately a constant resistance and a falling capacitance. Hence, as long as designers adopt a modular design approach and treat functional modules of up to 50,000 gates as the main components, these researchers' position is that current design flows can sustain the challenges of DSM design.⁶

On the other side of the debate, researchers argue that the previous argument does not account for the presence in SOCs of many global wires that cannot scale in length. As Figure A⁷ (next page) shows, such wires must span multiple modules to connect distant gates, so aren't scalable.

As Table A³ illustrates, the intrinsic interconnect delay of a 1-mm length wire for a 35-nm technology will be longer than the transistor delay by two orders of magnitude. Some researchers argue that, even under the best

continued on p. 26

Table A. Interconnect delay of 1-mm line vs. transistor delay for various process technologies.³

Technology (type of wire and substrate)	MOSFET* switching delay, approximate (ps)	Minimum, scaled, 1-mm interconnect intrinsic delay, approximate (ps)	Reverse, scaled, 1-mm interconnect intrinsic delay, approximate (ps)	Ratio of the wire size to the minimum lithographic size
10 μm (Al, SiO ₂)	20	5	5	1
0.1 μm (Al, SiO ₂)	5	30	5	1.5
35 nm (Cu, low-k)	2.5	250	5	4.5

*Metal-oxide-semiconductor field-effect transistor

next process generation. Latency is increasingly affecting the design of state-of-the-art microprocessors. (Latency, for example, drove the design of so-called *drive stages* in the new hyper-pipelined Netburst microarchitecture² of Intel's Pentium 4). The "impact of latency in the one-billion-transistor microprocessor design" sidebar (on p. 27) discusses latency as a force shaping this future architecture, which is expected within the next 10 years.

We have argued that interconnect latency will have a significant impact on the design of the communication architecture among the modules of a system on a chip. However, it is difficult to estimate the actual interconnect

latency early in the design cycle because several phenomena affect it. Process variations, crosstalk, and power supply drop variations all affect interconnect latency. Furthermore, their combined effect can vary across chip regions and periods of chip operation. Hence, finding the exact delay value for a global wire is often impossible. On the other hand, relying on conservative estimates to establish value intervals can lead to suboptimal designs.

We believe that recently proposed design flows advocating new CAD tools that couple logic synthesis and physical design will suffer from the impracticality of accurately estimating the latency of global wires. Indeed, logic

continued from p. 25

conditions, the latency to transmit a signal across the chip in a top-level metal wire will vary between 12 and 32 cycles, depending on the clock rate.⁸ In fact, although the number of gates reachable in a cycle will not change significantly and the on-chip bandwidth will continue to grow, the percentage of the die reachable within one clock cycle is inexorably and dramati-

cally decreasing. Designs will soon reach a point where more gates can fit on a chip that can communicate in one cycle.^{5,7} Hence, instead of being limited by the number of transistors integratable on a single die (computation bound), designs will be limited by the amount of state and logic reachable within the required number of clock cycles (communication bound).

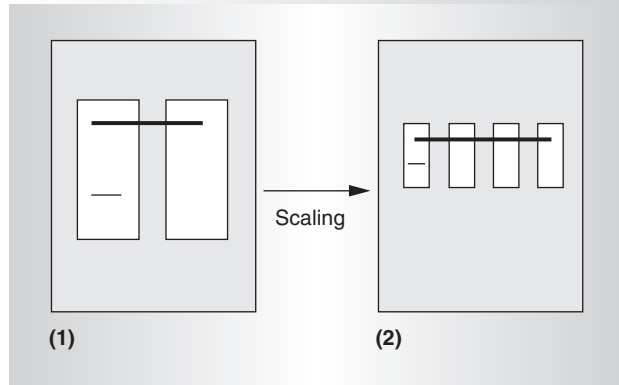


Figure A. Local wires scale in length; global wires do not. The figure shows the different impact of technology scaling on devices, local wires, and global wires. The newer technology process in SOC (2) allows a higher level of integration than SOC (1) as more (and smaller) devices and modules are accommodated on the chip. Local wires connect devices within a module and shrink with the module. Global wires connect devices located in different modules and do not shrink because they need to span significant proportions of the die.⁷

References

1. A. Allan et al., "2001 Technology Roadmap for Semiconductors," *Computer*, vol. 35, no. 1, Jan. 2002, pp. 42-53.
2. M.T. Bohr, "Interconnect Scaling: The Real Limiter to High Performance ULSI," *Proc. 1995 IEEE Int'l Electron Devices Meeting*, IEEE Press, Piscataway, N.J., 1995, pp. 241-244.
3. J.A. Davis et al., "Interconnect Limits on Gigascale Integration (GSI) in the 21st Century," *Proc. IEEE*, vol. 89, no. 3, Mar. 2001, pp. 305-324.
4. M.J. Flynn, P. Hung, and K.W. Rudd, "Deep-Submicron Microprocessor Design Issues," *IEEE Micro*, vol. 19, no. 4, July 1999, pp. 11-13.
5. D. Matzke, "Will Physical Scalability Sabotage Performance Gains?" *Computer*, vol. 30, no. 9, Sept. 1997, pp. 37-39.
6. D. Sylvester and K. Keutzer, "Rethinking Deep-Submicron Circuit Design," *Computer*, vol. 32, no. 11, Nov. 1999, pp. 25-33.
7. R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proc. IEEE*, vol. 89, no. 4, Apr. 2001, pp. 490-504.
8. V. Agarwal et al., "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA 00)*, ACM Press, New York, 2000, pp. 248-259.

synthesis tools presently suffer from several drawbacks. They are

- inherently unstable, and
- based on the synchronous design methodology.

Small variations to the hardware description language's input specification—like those a designer might make to fix a slow path—can lead to major variations in the output netlist and, consequently, in the final layout.

The synchronous design methodology is the foundation of the design flows for the majority of commercial chips today, but, if left unchanged, will lead to an exacerbation of the timing closure problem for tomorrow's design flows. In fact, the main assumption in synchronous design is that the delay of each combinational path is smaller than the system clock period. By combinational path,

we mean each signal path leaving a latch and traversing only combinational logic and wires to reach another latch. The slowest combinational path (critical path) dictates the maximum operating frequency for the system. However, it is often the case that the desired operating frequency represents a fixed design constraint. Then, once designers derive the final layout, every path with a delay longer than the desired clock period simply represents a design exception to be fixed. To fix these exceptions, designers use techniques from wire buffering and transistor resizing to rerouting wires, replacing modules, and even redesigning entire portions of the system.

Replacing, rerouting, and redesigning clearly do not alleviate the timing closure problem. Buffering is an efficient technique but carries precise limitations, because there is a limit to the number of buffers that designers can insert

The impact of latency in the one-billion-transistor microprocessor design

The evolution toward communication-bound design in microprocessors implies that the amount of state reachable in a clock cycle, and not the number of transistors, becomes the major factor limiting the growth of instruction throughput. Furthermore, the increasing interconnect latency will particularly penalize current memory-oriented microprocessor architectures that strongly rely on the assumption of low-latency communication with structures such as caches, register files, and rename/reorder tables. Recent studies employ cache-delay analysis tools that account for cache parameters as well as technology generation figures. These studies predict that in a 35-nm design running at 10 GHz and accessing even a 4-Kbyte level-one cache will require three clock cycles.¹ In fact, this trend has already started to affect design: The architects of the Alpha 21264 adopted clustered functional units and a partitioned register file to continue offering high computational bandwidth despite longer wire delays. Similarly, Intel's Pentium 4 microprocessor presents two so-called drive stages in its new hyperpipelined Netburst microarchitecture; designers dedicated these stages purely to instruction distribution and data movement.² Exposing interconnect latency to the microarchitecture and possibly to the instruction set will be key in controlling system performance. Several research teams have already started investigating this idea.^{3,8}

In particular, Bill Dally proposes a model for a hypothetical *2009 multicomputer chip*. He divides the proposed chip into 64 tiles (each containing a processor and a memory), where the memory processor's round-trip intratile communication latency is two cycles (1 ns). The worst-case latency for a corner-to-corner communication with the most distant memory is 56 cycles. More importantly, Dally points out the following features of his approach:

- On-chip communication bandwidth is not an issue because of the many wiring tracks.

on a given wire and still reduce delay. In fact, as a last resort, designers often must break long wires by inserting latches, which is similar to the insertion of new stages in a pipeline.

This operation, which trades off fixing a wire exception with increasing its latency by one or more clock cycles, will become pervasive in deep-submicron design, where most global wires will be heavily pipelined anyway. In fact, latency (measured in clock cycles) between SOC components will vary considerably based on the components' reciprocal distances—even without considering the need for increasing latency to further pipeline long global wires. Inserting latches (*stateful* repeaters) has a different impact on the surrounding control logic with respect to inserting buffers (*stateless* repeaters). If the interface logic design of two communication compo-

- Unlike modern multiprocessors with their all-or-nothing locality, latency on the 2009 multicomputer varies continuously with distance.
- This approach controls latency by placing data near their point of use, not just at their point of use.

References

1. V. Agarwal et al., "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA 00)*, ACM Press, New York, 2000, pp. 248-259.
2. P. Glaskowski, "Pentium 4 (Partially) Previewed," *Microprocessor Report*, vol. 14, no. 8, Aug. 2000, pp. 10-13.
3. W.J. Dally and S. Lacy, "VLSI Architecture: Past, Present and Future," *Proc. Advanced Research in VLSI Conf.*, IEEE Press, N.J., 1999, pp. 232-241.
4. K. Mai et al., "Smart Memories: A Modular Reconfigurable Architecture," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA 00)*, ACM Press, New York, 2000, pp. 161-171.
5. C.E. Kozyrakis et al., "Scalable Processors in the Billion-Transistor Era: IRAM," *Computer*, vol. 30, no. 9, Sept. 1997, pp. 75-78.
6. R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proc. IEEE*, vol. 89, no. 4, Apr. 2001, pp. 490-504.
7. E. Waingold et al., "Baring It All to Software: Raw Machines," *Computer*, vol. 30, no. 9, Sept. 1997, pp. 86-93.
8. R. Nagarajan et al., "A Design Space Evaluation of Grid Processor Architectures," *Proc. 34th Ann. Int'l Symp. Microarchitecture (Micro-34)*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 40-51.

nents assumes a certain latency, then designers must rework it to account for additional pipeline stages. Such rework has serious consequences on design productivity.

On the one hand, the increasing impact of latency variations will drive architectures toward modular designs with an explicit global latency mechanism. In the case of multiprocessor architectures, latency variation will lead the designers to expose computation/communication tradeoffs to the software compilers. At the same time, the focus on latency will open the way to new synthesis tools that can automatically generate the hardware interfaces responsible for implementing the appropriate synchronization and communication strategies, such as channel multiplexing, data coherency, and so on. All these considerations lead us to propose a design methodology that

guarantees the robustness of the system's functionality and performance with respect to arbitrary latency variations.

Latency-insensitive design

The foundation of latency-insensitive design is the theory of latency-insensitive protocols.³ A latency-insensitive protocol controls communication among components of a patient system—a synchronous system whose functionality depends only on the order of each signal's events and not on their exact timing. Designers can model a synchronous system as a set of modules. These modules communicate by exchanging signals on a set of point-to-point channels. The protocol guarantees that a system, if composed of functionally correct modules, behaves correctly, independent from delays in the channels connecting the modules. Consequently, it's possible to automatically synthesize a hardware implementation of the system such that its functional behavior is robust with respect to large variations in communication latency.⁴ In practice, the channel implementation can vary and does not necessarily follow the point-to-point structure. Still, this methodology orthogonalizes computation and communication because it separates the module design from the communication architecture options, while enabling automatic synthesis of the interface logic. This separation is useful in two ways:

- It simplifies module design because designers can assume the synchronous hypothesis. That is, intermodule communication will take no time (or, in other words, it's completed within one virtual clock cycle).
- It permits the exploration of tradeoffs in deriving the communication architecture up to the design process' late stages, because the protocol guarantees that the interface logic can absorb arbitrary latency variations.

In an earlier work, we discussed the application of the latency-insensitive methodology to the case where the channels are simply implemented as sets of point-to-point metal wires.³ Here, our discussion addresses an immediate advantage of this methodology: After physical design, it lets designers pipeline any long wire having a delay larger than the

desired clock period into shorter segments by inserting special memory elements called relay stations. Hence, we divide the latency-insensitive design flow into four basic steps:

1. Specification of synchronous components. Designers must first specify the system as a collection of synchronous components, relying on the synchronous hypothesis. These components can be custom modules or IP cores; we refer to them as pearls.
2. Encapsulation. Next, it's necessary to encapsulate each pearl within an automatically generated shell. A shell is a collection of buffering queues—one for each port—plus the control logic that interfaces the pearl with the latency-insensitive protocol.
3. Physical layout. In this traditional step, designers use logic synthesis, and place-and-route tools to derive the layout of the chip implementing the system.
4. Relay station insertion. Designers segment every wire whose latency is greater than the clock period by distributing the necessary relay stations. Relay stations are similar to regular pipeline latches and, therefore, inherently different from traditional buffering repeaters.

The only precondition this methodology requires is that the module's pearls are stallable, that is, they can freeze their operation for an arbitrary time without losing their internal state. This is a weak requirement because most hardware systems can be made stallable by, for instance, implementing a gated clock mechanism. But, this precondition is sufficient to permit the automatic synthesis of a shell around the pearl such that the shell-pearl combination satisfies the necessary, and much stronger, patience property.³ Figure 1 illustrates a system where five automatically generated shells encapsulate five pearls to make them patient. The resulting shell-pearl components communicate by means of eight point-to-point communication channels that implement the back-pressure mechanism described in the next section. Finally, the insertion of six relay stations enables channel pipelining to meet the system clock period.

Channels and back-pressure

Channels are point-to-point unidirectional links between a source and a sink module. Data are transmitted on a channel by means of packets that consist of a variable number of fields. Here, we consider only two basic fields: payload, which contains the transmitted data; and void, a one-bit flag that, if set to 1, denotes that no data are present in the packet. If a packet does contain meaningful payload data (that is, it has void set to 0), we call it a true packet.

A channel consists of wires and relay stations. The number of relay stations in a channel is finite and represents the channel's buffering capability. At each clock cycle, the source module can either put a new true packet on the channel or, when no output data are available, put a void packet on it. Conversely, at each clock cycle the sink module retrieves the incoming packet from the channel. It then discards or stores the packet on the input channel queue for later use, basing its decision on the void field value.

A source module might not be ready to send a true packet, and a sink module might not be ready to receive it if, for instance, its input queue is full. However, the latency-insensitive protocol demands fully reliable communication among the modules, and does not allow lossy communication links; it requires the proper delivery of all packets. Consequently, the sink module must interact with the channel (and ultimately with the corresponding source module) to momentarily stop the communication flow and avoid the loss of any packet. Therefore, we slightly relax our definition of a channel as unidirectional to allow a bit of information (the channel stop flag) to move in the opposite direction. The dashed wires in Figure 1 represent this signal, which is similar to the NACK signal in request/acknowledge protocols of asynchronous design. See the "Latency-insensitive versus asynchronous design" sidebar for a discussion of the two approaches.

By setting the stop flag equal to one during a certain clock cycle, the sink module informs

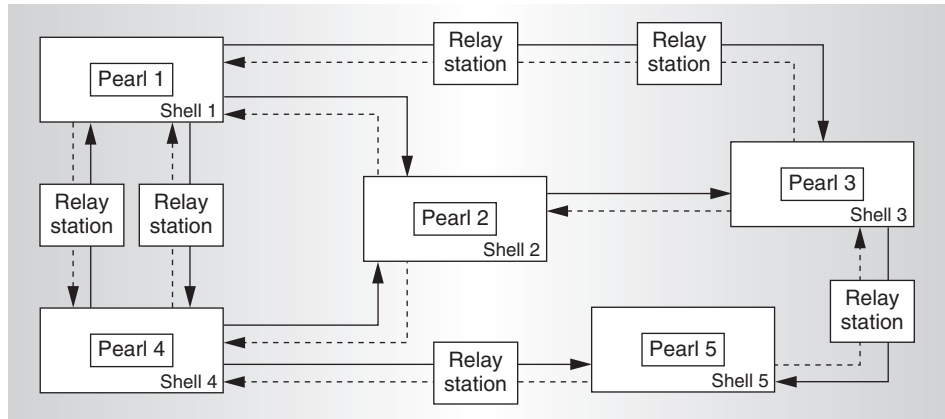


Figure 1. Shell encapsulation, relay station insertion, and channel back-pressure.

Latency-insensitive versus asynchronous design

The latency-insensitive design methodology is clearly reminiscent of many ideas that the asynchronous-design community has proposed during the past three decades.^{1,2} Informally, asynchronous design's underlying assumption is that the delay between two subsequent events on a communication channel is completely arbitrary. In the case of a latency-insensitive system, designers constrain this arbitrary delay to be a multiple of the clock period. The key point is that this type of discretization lets designers leverage well-accepted design tools and methodologies for the design and validation of synchronous circuits. In fact, the basic distinction between the two approaches is the specification of a latency-insensitive system as a synchronous system. Notice that we say *specified* because, from an implementation point of view, you could realize a latency-insensitive protocol using handshake-driven signaling techniques (for example, request-acknowledge mechanisms), which are typically asynchronous. On the other hand, synchronous systems specify a complex system as a collection of modules whose state is updated collectively in one zero-time step. This update is naturally simpler than specifying the same system as the interaction of many components whose state is updated following an intricate set of interdependency relations. IC designers could legitimately see latency-insensitive design as a compromise that accommodates important properties of asynchronous circuits within the familiar synchronous design framework.

References

1. W.A. Clark and C.E. Molnar, "The Promise of Macromodular Systems," *Digest of Papers 6th Ann. IEEE Computer Soc. Int'l Conf.*, IEEE Press, Piscataway, N.J., 1972, pp. 309-312.
2. I.E. Sutherland, "Micropipelines," *Comm. ACM*, vol. 32, no. 6, June 1989, pp. 720-738.

the channel that it can't receive the next packet, and the channel must hold the packet until the sink module resets the stop flag. Because the sink module and channel have limited buffering resources, a channel dealing with a sink module that requires a long stall might fill up all its relay stations and have to send a stop flag to the source module to stall its packet pro-

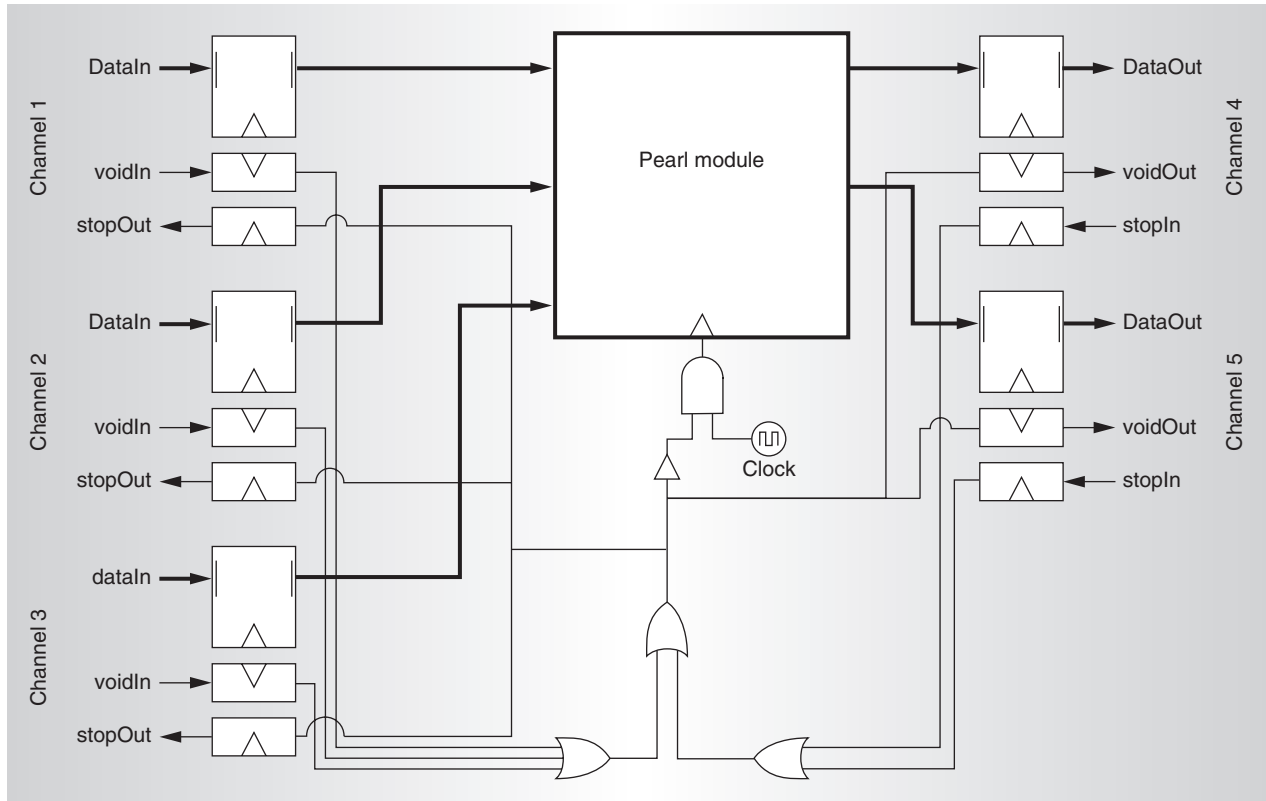


Figure 2. Shell encapsulation: Making an IP core patient.

duction. This back-pressure mechanism controls the flow of information on a channel while guaranteeing that no packets are lost.

Shell encapsulation

Given particular module M , we can devise a method to automatically synthesize an instance of a shell as a wrapper to encapsulate M . We can further interface the shell with the channels so that M becomes a patient system. To do so, the only necessary condition is that M be stallable. At each clock cycle, the module's internal computation must fire only if all inputs have arrived. The module shell's first task is to guarantee this input synchronization. Its second task is output propagation: At each clock cycle, if M has produced new output values and no output channel has previously raised a stop flag, then the shell can transmit these output values by generating new true packets. If the shell does not verify either of these two conditions, then it must retransmit the packet transmitted in the previous cycle as a void packet. In summary, a shell for module M cyclically performs the following actions:

1. It obtains incoming packets from the input channels, filters away the void packets, and extracts the input values for M from the payload fields of the true packets.
2. When all input values are available for the next computation, it passes them to M and initiates the computation.
3. It obtains the computation results from M .
4. If no output channel has previously raised a stop flag, it routes the result into the output channels.

Figure 2 illustrates a simple unoptimized implementation of a shell wrapping a module with three input and two output channels. It's also possible to insert queues between the shell I/Os and M 's I/Os to increase buffering and avoid fragmented stalling. Using queues lets designers optimize implementations. Because such implementations follow the protocol outlined previously, the shell-pearl combination operates in a way similar to that of an actor in a static data flow. Hence, designers can take advantage of useful properties,

such as the ability to statically size the shell queues and to statically compute the performance of the overall system.

Relay stations

A relay station is a patient process communicating with two channels, c_i and c_o . Let s_i and s_o be the signals associated with the channels. Further, let $I(l, k, s_i)$, $l \leq k$, denote the sequence of informative events (true packets) of s_i between the l th and k th clock cycle. If these conditions are true, then s_i and s_o are latency equivalent and for all k

$$I[1, (k-1), s_i] - I[1, k, s_o] \geq 0$$

$$I[1, k, s_i] - I[1, (k-1), s_o] \leq 2$$

The following is an example of relay station behavior, where τ denotes a stalling event (void packet) and l_i a generic informative event:

$$s_i = l_1 l_2 l_3 \tau \tau l_4 l_5 l_6 \tau \tau \tau l_7 \tau l_8 l_9 l_{10} \dots$$

$$s_o = \tau l_1 l_2 l_3 \tau \tau \tau l_4 \tau \tau \tau l_5 l_6 l_7 \tau l_8 l_9 l_{10} \dots$$

Notice that we don't further specify signals s_i and s_o ; not even, for example, saying that s_i is the input and s_o is the output. The definition of relay station simply involves a set of relations—a protocol, between s_i and s_o without any implementation detail. Still, it's clear that each informative event received on channel c_i is later emitted on c_o , while the presence of a stalling event on c_o might induce a stalling event on c_i in a later cycle. In fact, an informative event takes at least one clock cycle to pass through a relay station (minimum forward latency equal to one), and at most two informative events can arrive on c_i while no informative events are emitted on c_o (internal storage capacity equal to two). Finally, one extra stalling event on c_o will move into c_i in at least one cycle (minimum backward latency equal to one). The double storage capacity of a relay station permits, in the best case, communication with maximum throughput (equal to one).

Because relay stations are patient processes, and patience is a compositional property, the insertion of a relay station in a patient system guarantees that the system remains patient.³ Further, because relay stations have minimum latencies equal to one, designers can repetitively insert them into a channel to

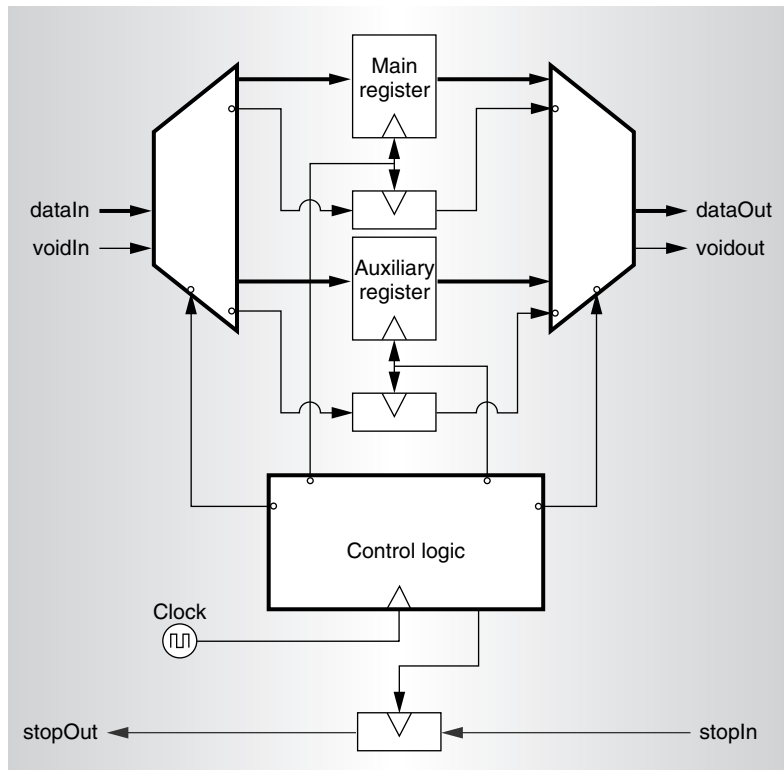


Figure 3. Example relay station implementation.

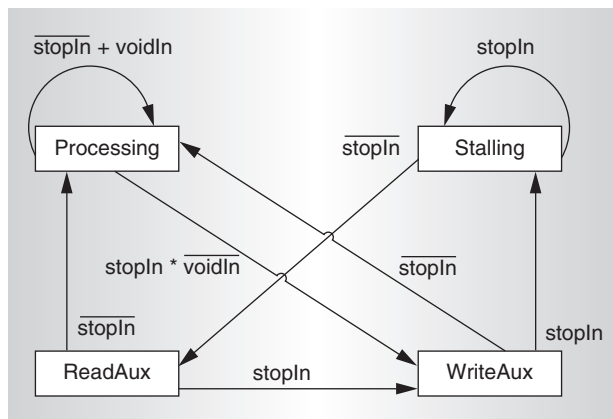


Figure 4. Finite state machine abstracting the relay station logic from Figure 3.

segment it, thereby increasing the channel's latency. Figure 3 illustrates a possible relay station implementation. Researchers have recently proposed other interesting relay station implementations.⁵

Figure 4 shows a finite state machine representing the control logic in the relay station of Figure 3. Basically, the relay station switches between two main states: *processing*, when

a flow of true packets traverses the relay station; and *stalling*, when communication is interrupted. Switching through states WriteAux and ReadAux governs the transition between these two states. These states ensure proper control of the auxiliary register to avoid losing a packet, because the stop flag takes a cycle to propagate backward through the relay station.

Recycling

In general, an automatic tool should complete the insertion of relay stations as part of the physical design process (similarly to the buffer insertion techniques available in current design flows). In fact, the latency-insensitive methodology's main advantage is the freedom offered to designers in moving the latency after deriving the final implementation. Designers can fix problematic layouts without changing the design of individual modules and also explore and optimize latency-throughput tradeoffs up to the late stages of the design process. The recycling paradigm formally captures the latency variations of the communication channels as you add (and move) relay stations. It lets you exactly compute the system's final throughput.⁶

For recycling, we model a latency-insensitive system as a directed (possibly cyclic) graph G that associates each vertex with a shell-pearl pair and in which each arc corresponds to a channel. We annotate each arc a with length $l(a)$ and weight $w(a)$. The length denotes the smallest multiple of the desired clock period that is larger than the delay of the corresponding channel. The weight represents the number of relay stations inserted on the channel. We call arc a illegal if $w(a_i) < l(a_i) - 1$. The presence of illegal wires in the layout implies that the final implementation is incorrect. Thanks to the latency-insensitive methodology, we can correct the final layout by introducing relay stations to ensure that each wire's delay is less than the desired clock period. This insertion corresponds to incrementing the weight of each illegal arc a by the quantity $\Delta w(a) = l(a) - 1 - w(a)$.

Although recycling is an easy way to correct the system's final implementation and satisfy the timing constraints imposed by the clock, it does have a cost. In fact, augmenting the weights of some arcs of G could increase

the graph's maximum cycle mean $\lambda(G)$, defined as $\max_C \in G \{[w(C) + |C|] / |C|\}$, where C is a cycle of G and $w(C)$ the sum of the weights of the arcs of C . Increasing $\lambda(G)$ corresponds to increasing the cycle time of the system modeled by G and, symmetrically, decreasing its throughput $\vartheta(G)$.⁶ Throughput always decreases if any arc a with augmented weight $w(a)$ belongs to the set of critical cycles of G , that is, those cycles whose mean coincides with the maximum cycle mean. Increasing the weights of some arcs might make a noncritical cycle of G become a critical cycle of recycled graph G' . In any case, after completing the recycling transformation, we can 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 $\vartheta(G') = 1/[\lambda(G')]$.
- After establishing set A of cycles having at least one arc with an augmented weight, increment the cycle mean of each element C of A by the quantity $1/|C| \times \sum_i \Delta w(a_i)$, where a_i are the arcs of C that have been corrected. Let λ^* be the maximum among all these cycle means, then

$$\Delta\vartheta(G, G') = 0 \text{ if } \lambda^* \leq \lambda(G) \text{ or } [\lambda^* - \lambda(G)] / [\lambda(G) \times \lambda^*] \text{ otherwise.}$$

The critical cycle dictates overall throughput because the rest of the system must slow down to wait for it and avoid packet loss. In general, if G contains more than one strongly connected component, you must decompose the recycling transformation in two steps:

1. Legalization. Legalize G by augmenting the weights of the wires by the appropriate quantity.
2. Equalization. Compute maximum throughput $\vartheta(S_k) = a_k/b_k \in]0, 1]$ that 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 quantity $n_k \in \mathbb{Z}^*$ to the denominator of each $\vartheta(S_k)$. This corresponds to augmenting by quantity n_k the weight of the critical cycle $C_k \in S_k$, that is, to dis-

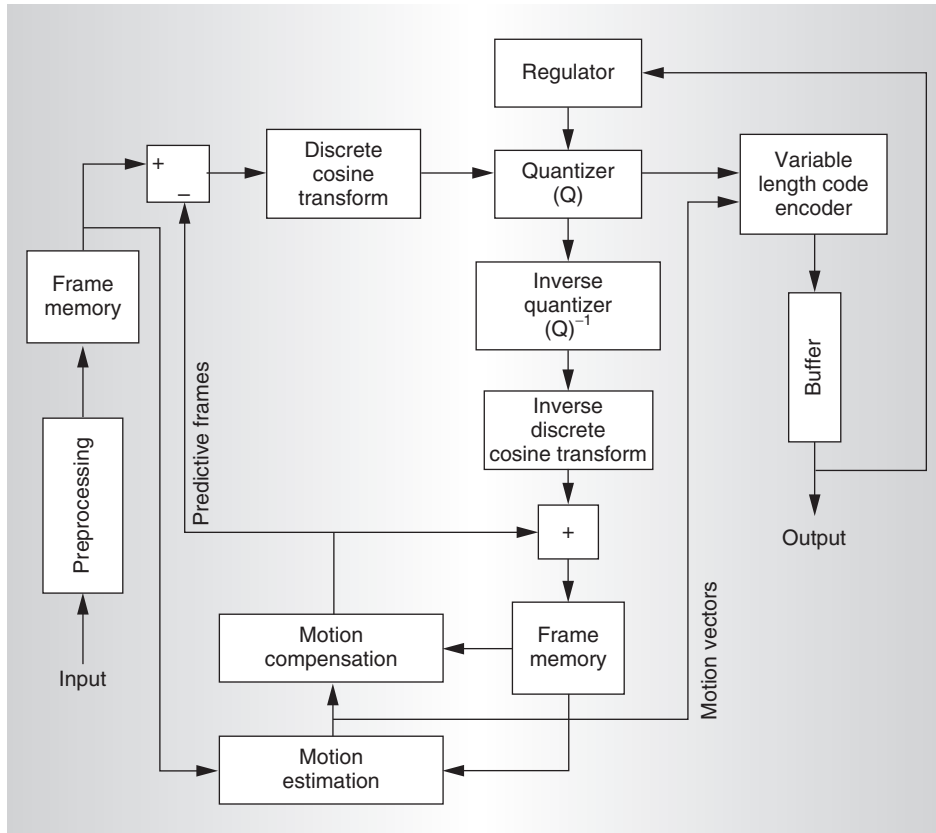


Figure 5. Block diagram of an MPEG-2 video encoder.

tribute n_k extra relay stations among the corresponding paths. Solving an optimization problem gives the quantities n_k .⁶

One key to avoiding large performance losses during recycling is to avoid augmenting the weights of those arcs belonging to critical cycle C of G . Furthermore, from the definition of maximum cycle mean, we have that for the same $w(C)$, the smaller the cycle's cardinality $|C|$, the greater the loss in throughput for G . The worst case is clearly represented by self-loops. Designers must keep these considerations in mind while partitioning the system functionality into tasks assigned to different IP cores. It's true that the latency-insensitive methodology guarantees that no matter how poor the final system implementation (in terms of wire lengths in the communication architecture), it's always possible to fix the system by adding relay stations. Still, to achieve acceptable performance, designers should adopt a design strategy based on the following guidelines:

- All modules should put comparable timing constraints on the global clock (that is, delays of the longest combinatorial paths inside each module should be similar).
- Modules whose corresponding vertices belong to the same cycle should be kept close to each other while deriving the final implementation.

Figure 5 illustrates the functional diagram of an MPEG-2 video encoder whose corresponding graph, reported in Figure 6 (next page), contains six distinct cycles:

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

Most arcs are common to more than one cycle, for example, a_9 is part of C_1 , C_2 , C_4 , C_5 , and C_6 , but others are contained only in one

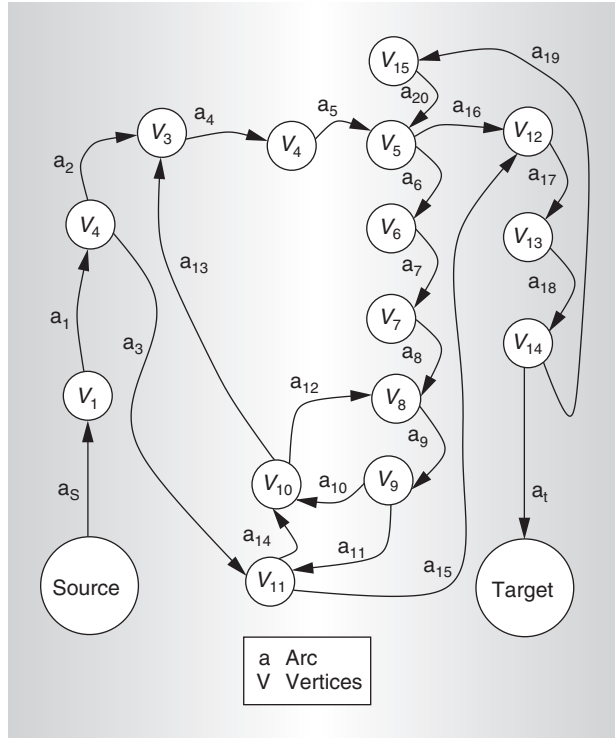


Figure 6. Latency-insensitive design graph of the MPEG-2 video encoder shown in Figure 5. The vertices of the graph correspond to blocks in the MPEG diagram. Source and Target represent respectively the input and output of the MPEG.

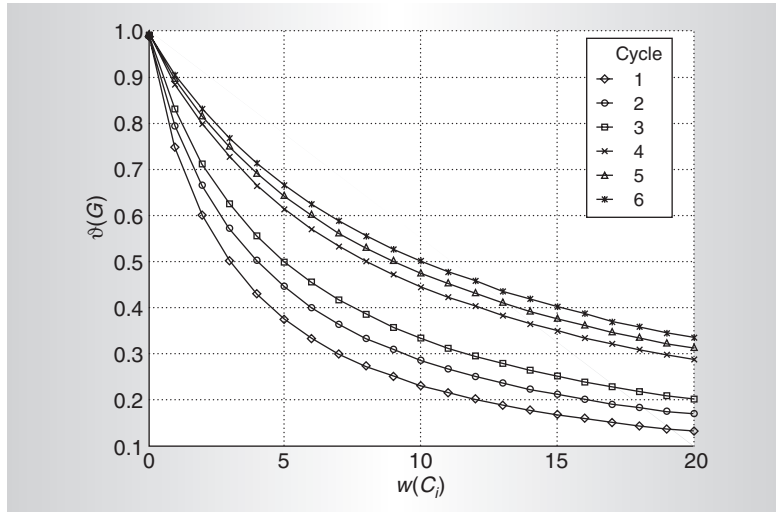


Figure 7. Analysis of throughput degradation for the MPEG-2 encoder.

cycle; for example, a_{15} is only part 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 sys-

tem performance. However, can we compute performance degradation in advance for the arcs belonging to one or more cycles? Figure 7 shows the results of an analysis based on the recycling approach. The six curves in the chart represent the previously mentioned cycles. Each point of curve C_i shows the degradation in system throughput detected after setting total sum $w(C_i)$ of the weights of C_i 's arcs equal to integer x , with $x \in [0, 20]$. Obviously, the underlying assumption is that C_i is a critical cycle of G , limiting the choice of those arcs of C_i with augmentable weights. 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 $\lambda(C_2) = (5 + 4)/4 = 9/4 = 2.250$. 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 $\lambda(C_1) = (4 + 3)/3 = 7/3 = 2.333$.

As Figure 7 confirms, the best way to avoid losing performance is to increase the weights of those arcs that belong to longer cycles. Performing the recycling transformation this way might or might not be possible, for example, if the length of arc a_{10} is large, cycle C_1 will ultimately dictate the system throughput. However, in general the latency-insensitive methodology lets us move around relay stations without redesigning any module. This capability could be useful, for example, in reducing the length of an arc such as a_9 , which belongs to both large and small cycles, while in exchange increasing the length of a_{15} , which is only part of C_6 . This example assumes that, before rebalancing, $\ell(a_9) = 3$ and $\ell(a_{15}) = 1$, where, after rebalancing, they become respectively 1 and 3. It also assumes that all other arcs have unit lengths, giving a final throughput of $10 / (10 + 2) = 0.833$ instead of $3 / (3 + 2) = 0.6$, a 38 percent improvement.

Deep-submicron technologies suffer—and in the foreseeable future will continue to suffer—from delays on long wires that often force costly redesigns. Latency among various SOC components will vary considerably and in a manner that is difficult to predict. Latency-insensitive design is the foundation of a correct-by-construction methodology that lets us increase the robustness of a design implementation. This approach enables recovering the

arbitrary delay variations of the wires by changing their latency, leaving overall system functionality unaffected. A promising avenue for further research is the application of these concepts to the optimization of the computation/communication tradeoffs that arise while designing software compilers for machines having a communication architecture with variable latency.

MICRO

References

1. K. Keutzer et al., "System Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, Dec. 2000, pp. 1523-1543.
2. P. Glaskowski, "Pentium 4 (Partially) Previewed," *Microprocessor Report*, vol. 14, no. 8, Aug. 2000, pp. 10-13.
3. L.P. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli, "Theory of Latency-Insensitive Design," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 9, Sept. 2001, pp. 1059-1076.
4. L.P. Carloni et al., "A Methodology for 'Correct-by-Construction' Latency Insensitive Design," *Proc. Int'l Conf. Computer-Aided Design (ICCAD 99)*, IEEE Press, Piscataway, N.J., 1999, pp. 309-315.
5. T. Chelcea and S. Novick, "Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols," *Proc. 38th Design Automation Conf. (DAC 01)*, ACM Press, New York, 2001, pp. 21-26.
6. L.P. Carloni and A.L. Sangiovanni-Vincentelli, "Performance Analysis and Optimization of Latency Insensitive Systems," *Proc. 37th Design Automation Conf. (DAC 00)*, IEEE Press, Piscataway, N.J., 2000, pp. 361-367.


Luca P. Carloni is a PhD candidate in the Electrical Engineering and Computer Science Department of the University of California, Berkeley. His research interests include computer-aided design and design methodologies for electronic systems, embedded system design, logic synthesis, and combinatorial optimization. He has a Laurea in electrical engineering from the University of Bologna, Italy, and an MS in electrical engineering and computer science from the University of California, Berkeley.

Alberto L. Sangiovanni-Vincentelli holds the Buttner Chair in electrical engineering and computer science at the University of California, Berkeley. His research interests include all aspects of computer-aided design of electronic systems, hybrid systems and control, and embedded system design. He cofounded Cadence and Synopsys and received the Kaufman Award from the Electronic Design Automation Consortium for pioneering contributions to EDA. He is a fellow of the IEEE and a member of the National Academy of Engineering.

Direct questions and comments about this article to Luca P. Carloni, 211-10 Cory Hall, University of California at Berkeley, Berkeley CA 94720; lcarloni@ic.eecs.berkeley.edu.

For further information on this or any other computing topic, visit our Digital Library at <http://computer.org/publications/dlib>.

Good news for your in-box.



Sign Up Today

for the IEEE

Computer


Society's

e-News

Be alerted to

- articles and special issues
- conference news
- submission and registration deadlines
- interactive forums

Available for FREE to members.



computer.org/e-News