

# Generalized Latency-Insensitive Systems for GALS Architectures

Montek Singh<sup>1</sup> and Michael Theobald<sup>2</sup>

<sup>1</sup> Department of Computer Science  
UNC Chapel Hill, NC, USA  
`montek@cs.unc.edu`

<sup>2</sup> Computer Science Department  
Carnegie Mellon University, Pittsburgh, PA, USA  
`theobald@cs.cmu.edu`

**Abstract.** *Latency-insensitive systems* were recently proposed by Carloni et al. for the design of single-clock systems-on-a-chip (SoC's) using predesigned IP blocks. The goal of this paper is to extend and generalize latency-insensitive systems in such a way that they can be applied to GALS architectures with multiple clocks. In particular, we propose two extensions. The first extension allows each synchronous module to treat its input and output channels in a much more flexible manner (*i.e.*, greater decoupling). As a result, significant improvement in throughput as well as power consumption may be obtained. The second extension generalizes inter-module communication from point-to-point channels to more complex networks of arbitrary topologies.

## 1 Introduction

Globally-asynchronous locally-synchronous (GALS) architectures have recently witnessed a surge of interest from designers of large-scale systems-on-a-chip (SoC). One of the key reasons for such a trend is that GALS design offers a good compromise between single-clock synchronous design—which is likely to become ever harder with technology scaling—and completely clockless design, which suffers from lack of tool support.

Latency-insensitive systems were originally proposed by Carloni et al. [1–4] for the design of single-clock SoC's. Their goal was to obtain greater modularity of design by using components that can readily adapt to their environments. In particular, a synchronous module is said to be latency-insensitive if it can operate correctly in the presence of arbitrary delays on its input and output channels. As a result, latency-insensitive IP blocks are easily composable, offering ease of design reuse. If an IP block is not latency-insensitive, their approach provides simple “wrapper” circuits that make it latency-insensitive.

While Carloni et al.'s latency-insensitive approach has been successfully used in the design of single-clock SoC's, it needs significant extensions before it can be successfully used for the design of GALS systems with multiple clocks. In particular, their approach assumes identical data rates on all input and output channels of a synchronous module—an entirely reasonable assumption for the case of a single-clock system. However, when multiple synchronous modules with different clock frequencies are connected together, unnecessary stalls are introduced, thereby lowering system performance. In fact, this paper shows that the system throughput will be limited to the throughput of its slowest synchronous module. In addition, their approach only considers point-to-point interconnects. As this paper shows, this limitation can also cause a loss of performance in the context of a GALS system.

This paper extends and generalizes Carloni et al.’s approach to latency-insensitive systems, in order to make them useful for the design of GALS systems. Two specific generalizations are proposed. The first allows each synchronous module to treat its input and output channels in a more flexible manner, allowing a great decoupling of their operation. One benefit of this extension is that modules with different clock speeds can be combined more efficiently. The second extension allows the communication networks to be implemented using arbitrary topologies, instead of only using point-to-point channels. The net impact is that our proposed approach offers improved throughput, reduced power consumption, and greater flexibility in design.

This paper is organized as follows. First, Section 2 discusses some of the issues in the design of GALS systems. Then, Section 3 provides a survey of relevant approaches and their limitations. Section 4 presents our new approach. A few interesting ideas for future exploration are outlined in Section 5. Finally, Section 6 gives conclusions.

## 2 Background: Issues in the Design of GALS Systems

There are several challenges in the design of GALS systems. These challenges can be grouped into two categories: (i) issues in the design of the locally synchronous modules, and (ii) issues in the design of the asynchronous communication network. This section briefly reviews some of these challenges.

### 2.1 Design of Locally Synchronous Modules

The very nature of an asynchronous interconnection network between locally synchronous modules imposes certain requirements on the behavior of those synchronous modules. In particular, the synchronous modules must typically be flexible enough to operate “correctly” in the presence of arbitrary delays on the interconnects between those modules.

The particular notion of “correctness” referred to here is one of *latency-insensitivity*, as defined by Carloni et al. [1]. A synchronous module is said to be latency-insensitive if arbitrary delays on its input channels only cause delays in generation of output values on its output channels (with “bubbles” or “spacers” in-between); the actual output values produced are otherwise unchanged. That is, input streams that differ only in the number and location of spacers produce output streams that differ only in the number and location of spacers.

This property of latency-insensitivity is highly desirable for the design of GALS systems, because it allows the separation of computation from communication. In particular, since communication latencies are a function of the latter steps in physical design (*e.g.*, placement and routing), the use of latency-insensitive synchronous modules allows ease of composition, as well as robustness of operation.

### 2.2 Design of Asynchronous Communication Network

The asynchronous interconnection network, which glues together all of the synchronous modules in a GALS system, must also satisfy certain properties in order to ensure correct, reliable operation. In particular, the communication network must adequately resolve lower-level electrical issues such as *metastability* [5], as well as higher-level issues such as *flow control* (*i.e.*, prevent data overruns).

## 3 Current Approaches and their Limitations

This section reviews some of recent work in several areas related to GALS systems, and discusses how some of that work can be applied towards the design of GALS systems. In particular, the

shortcomings of some recent approaches, which prevent them from being directly used for GALS design, are pointed out. The design of synchronous modules is considered first, followed by approaches for asynchronous inter-module communication.

### 3.1 The Design of Latency-Insensitive Synchronous Modules

We discuss three different approaches to providing synchronous modules with the capability to tolerate delays on their communication channels: (i) pausable clocking, (ii) module stalling via clock gating, and (iii) fine-grain synchronous handshaking.

**Pausible Clocking (Yun et al. [6]).** This approach, also called stretchable or stoppable clocking, is currently by far the most commonly used approach for designing GALS systems [7, 6, 8–11]. It enables the design of synchronous modules that are tolerant of arbitrary delays in communication. The key idea is to make the clock of each module pausable: when any needed communication channel is not ready, the clock of the synchronous module is paused by stretching the clock’s inactive phase. Each pausable clock is generated using a ring oscillator, with a control input for stopping and (re-)starting the clock.

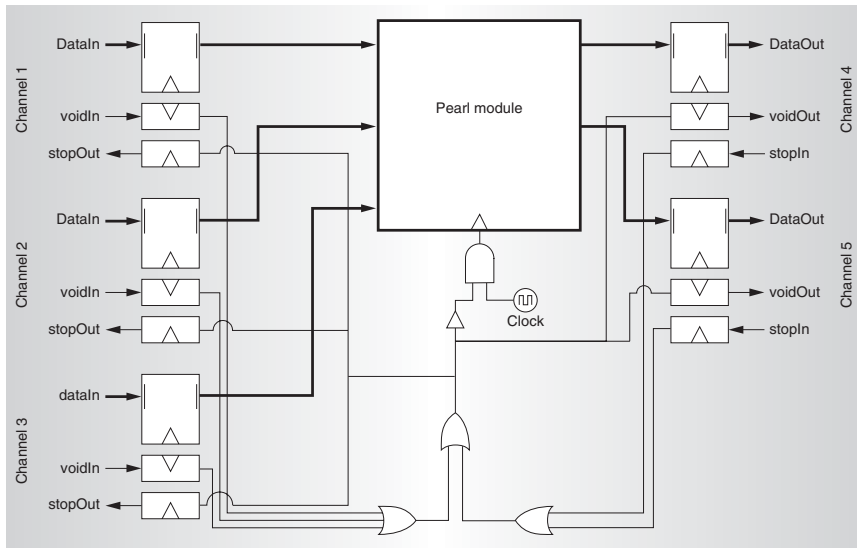
A significant disadvantage of the pausable clocking approach is the use of ring oscillators for clock generation, as opposed to crystal oscillators. While crystal oscillators can maintain clock frequency stability to a few parts per million, ring oscillators are prone to significant amounts of jitter as well as clock frequency variation. In particular, the pausing and restarting of the ring oscillator, as well as noise, can cause appreciable amounts of jitter [12, 13]. Further, temperature and voltage variations can cause significant variation in clock frequency. As a result, the performance of synchronous IP blocks may be severely degraded, since stable low-jitter clocks are key to modern high-performance synchronous design.

Due to the above disadvantages, we feel that pausable clocking currently only has a limited potential for use in high-performance GALS design. Therefore, our approach currently focuses on clock gating instead. We now discuss some of the existing latency-insensitive approaches that are based on clock gating.

**Module Stalling Via Clock Gating (Carloni et al. [1]).** This approach obtains latency-insensitive operation of synchronous modules by carefully gating their clocks [1–4]. The key idea is to decouple the computation performed by synchronous modules from inter-module communication. This decoupling is achieved by encapsulating the synchronous modules inside wrapper circuits, in order to make them latency-insensitive, thereby enabling greater ease of composition. After encapsulation, each synchronous module has two key properties: (i) it is stallable, *i.e.*, its clock can be gated when appropriate, and (ii) its behavior is insensitive to arbitrary delays on its input and output communication channels.

While this approach has been successfully used in the design of single-clock SoC’s, it needs significant extensions before it can be successfully used in multiple-clock systems. In particular, there are two specific features of their approach which make it unsuitable for direct use in GALS architectures, as discussed below.

First, their approach makes certain simple assumptions regarding the input-output behavior of synchronous modules. These assumptions may cause unnecessary stalls in GALS architectures, leading to loss of throughput. In particular, in their approach, on each clock tick, a given synchronous module consumes a data item from each of its input channels, and generates a data item for each of its output channels. If any input channel is not ready with valid data, or if any output channel is not



**Fig. 1.** Carloni et al.’s approach to latency-insensitivity (from [14])

ready to accept data, the synchronous module is stalled by gating its clock, as shown in Figure 1. This assumption about a module’s communication is overly restrictive in the GALS context: *e.g.*, a synchronous DSP core may need only a subset of its inputs and may generate data for only a subset of its output channels. Thus, their approach may cause a significant loss of throughput by generating more stalls than may be necessary.

Second, their approach only considers a simple architecture for inter-module communication: point-to-point channels composed of linear FIFO’s. Such a simple approach is overly restrictive for GALS design because of two reasons: (i) it prevents the designer from specifying more sophisticated interconnection topologies, and (ii) it may result in a significant loss of performance if the throughputs of the synchronous modules are not perfectly matched. More details are provided in Section 4.

**Fine-Grain Synchronous Handshaking.** Another approach to latency-insensitive synchronous design is called *synchronous handshaking*, which adapts asynchronous handshaking for use with clocking [15–17]. This approach pushes latency-insensitivity to the granularity of individual latches within a synchronous module. That is, clock gating is performed on a per-latch basis, instead of for the entire synchronous module.

Synchronous handshaking has the advantage that clock gating is achieved at a very fine granularity, thereby maximizing the savings in power consumption. In addition, it maximizes the *elasticity* of the synchronous system, *i.e.*, the ability to handle variability in input–output rates.

In the context of GALS design, however, synchronous handshaking approaches have certain limitations. First, these approaches can only be used during the design of IP blocks; they cannot be applied to predesigned IP modules that are not already latency-insensitive. Second, these approaches can potentially be area-expensive, since clock-gating structures must be added for every latch in the design. Third, these approaches cannot currently be applied to varying levels of granularity in IP design; they are targeted to only the finest granularities. Finally, unlike Carloni et al.’s approach,

no formal approach is provided to model, analyze, and compose individual synchronous modules designed using these approaches.

### 3.2 Asynchronous Communication Network

Several approaches have been proposed for interconnecting distinct synchronous modules running on different clocks. Two approaches are particularly relevant for GALS systems: one by Chelcea and Nowick [18], and another by Chakraborty and Greenstreet [13]. Both of the approaches provide robust communication between distinct synchronous domains.

However, both of the above approaches have their limitations. In particular, these approaches are only targeted toward point-to-point communication. As discussed above, and in Section 4.1, point-to-point communication is suboptimal for the design of general GALS architectures. A more general communication scheme is needed, which allows arbitrary interconnection topologies, *e.g.*, those with forks, joins, conditionals etc.

Finally, we would like to mention that some alternative approaches based on simple topologies—*e.g.*, shared buses and rings [19], or simple forks and joins [20]—have been recently reported. While these are steps in the right direction, they are nevertheless point solutions. Therefore, a general framework is still needed for the formal specification and synthesis of arbitrary communication networks.

## 4 New Approach

This section presents our new contribution. First, two examples are presented in Section 4.1 to show the limitations of existing approaches, and provide motivation for the new approach. Next, Sections 4.2 and 4.3 present our proposed approach in two parts: (i) a generalization of latency-insensitive synchronous systems, and (ii) a generalization of inter-module communication networks. Finally, Section 4.4 briefly presents some initial ideas towards developing a formal approach for modeling GALS architectures.

### 4.1 Motivation

We present two examples that motivate the new approach.

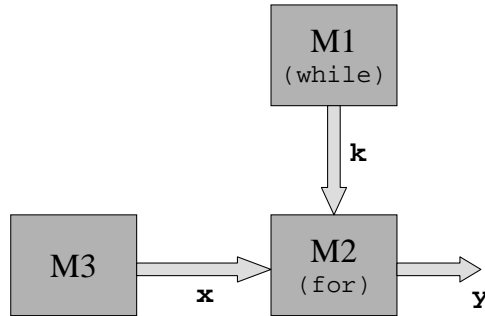
**Example 1.** This example motivates the need, in the context of GALS systems, for a generalization of the model of latency-insensitive synchronous modules proposed by Carloni et al. [1]. Their approach uses a simplifying assumption that every input channel, as well as every output channel, is exercised by a synchronous module on every clock tick. While this assumption works fine for a single-clock system, this example shows that the assumption can be overly restrictive for the more general case of GALS systems.

Consider the following program fragment describing a simple system:

```

while (true) do
  input k;
  for (i=0; i<k; i++) do
    input x;
    compute y=f(x);
    output y;
  end for
end while

```



**Fig. 2.** A possible implementation of the nested loops of Example 1.

Figure 2 shows one possible hardware implementation of this specification, using a decomposition of the system into three distinct synchronous modules:  $M1$ ,  $M2$  and  $M3$ . Module  $M1$  corresponds to the `while` loop and provides the input  $k$  to  $M2$ .  $M2$  corresponds to the `for` loop, and  $M3$  provides the input  $x$  to  $M2$ .

There are two key features of this system that make it difficult to implement it using Carloni et al.’s latency-insensitive approach. First, for every input item that  $M2$  receives from  $M1$ ,  $k$  data items are read by  $M2$  from  $M3$ . Thus, after a data item is received by  $M1$  from  $M2$ , no further communication is needed between those modules for the next  $k-1$  clock cycles. Thus, Carloni et al.’s latency-insensitive modules cannot be directly used here because they exercise every input and output channel on every clock tick. Therefore, it becomes necessary to modify their approach in order to allow the handling of situations where only a subset of input channels must be read. Likewise, writing to only a subset of output channels must also be handled.

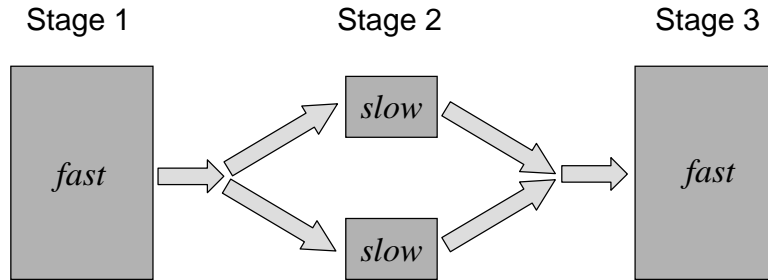
The second key feature of this system is data-dependent completion of module  $M2$ ’s operation. In particular, the number of iterations of the inner loop is controlled by a data input,  $k$ , provided by  $M1$ . Since Carloni et al.’s approach does not consider data-dependent completion times, a new approach is needed to model data-dependency in inter-module communication.

**Example 2.** This example shows that point-to-point communication channels may be inadequate for the design of GALS systems, especially when the clock rates of the synchronous modules are different. Consider the architecture of a three-stage stream processor, shown in Figure 3. Assume that fast modules (*i.e.*, IP blocks) are available for the first and third stages, but only half-rate modules are available for the second processing stage. In order to achieve full throughput, two modules must be used for the second stage, necessitating a fork-join type of interconnection. If only simple point-to-point communication channels were allowed, the designer may be forced to use a single module for the second processing stage, resulting in a 50% loss of throughput.

In general, we can state the following for any GALS system composed of Carloni et al.-style synchronous modules connected by point-to-point communication channels:

*System throughput will be limited by the throughput of the slowest synchronous module.*

If the GALS system—when treated as an undirected graph, with each module represented as a node, and each channel as an edge—is composed of more than one connected component, then the above statement applies to each connected component.



**Fig. 3.** Illustrating the need for more sophisticated communication networks, *e.g.*, those involving forks and joins.

The validity of the claim is obvious if one notes that the synchronous modules of Carloni et al. [1] are required to stall if even a single input channel is not ready with valid new data, or if even a single output channel is not ready to receive new data.

## 4.2 Generalized Latency-Insensitive Synchronous Modules

We now propose a generalization of the notion of latency-insensitivity to handle the case of GALS systems. In particular, we propose modifications of the wrappers that make synchronous modules latency-insensitive, in order to handle more general input–output behavior. This modification directly addresses the issues raised in Example 1 (Figure 2) above, *i.e.*, avoids unnecessary stalling when unavailable inputs are not actually needed.

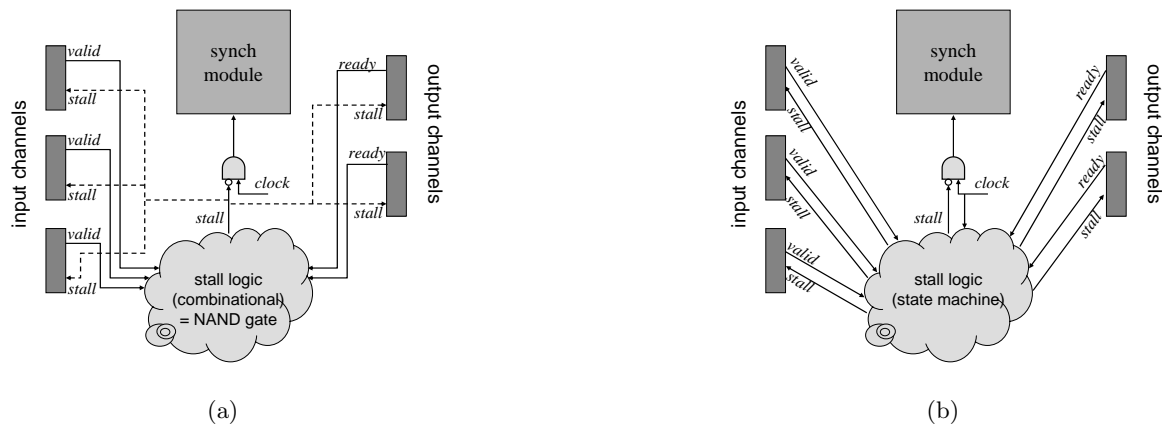
Our modification applies to the stall generation circuitry inside the wrappers circuits. In particular, the combinational logic that generates stalls is now replaced by a more sophisticated finite-state machine. The inputs to this state machine are the *valid* and *ready* signals from input and output channels, respectively.<sup>1</sup> The outputs of the state machine are: (i) the stall signal, used to gate the module’s clock, and (ii) handshake signals for the input and output channels. Figure 4 illustrates this modification.

The generalization of stall logic to a state machine allows us to treat input and output channels in a more flexible manner. For example, for a given state, only a particular subset of the input channels may need to be read, and only a subset of the output channels may need to be written. Using terminology borrowed from asynchronous control synthesis, these subsets of channels can be referred to as *input bursts* and *output bursts* (cf. burst-mode asynchronous controllers [21, 22]). The remaining channels, which are not read or written, are ignored for the next clock tick; unavailability of these channels does not generate stalls.

Going a step further, each state of the finite-state machine can be associated with multiple input bursts, much like in burst-mode asynchronous controllers. The availability of any one of these input bursts allows the synchronous module to operate on the next clock tick, and produce an output burst. If no input burst is available, the module is stalled. If determinism is required, more than one input burst should never be available simultaneously; as a corollary, no input burst should be a proper subset of another input burst.

For convenience of modeling, the state machine could be represented using the notation used for asynchronous burst-mode machines [21, 22]. However, it must be borne in mind that the state

<sup>1</sup> These signals are simply the negations of the `voidIn` and `stopIn` signals of Carloni et al. (see Figure 1).



**Fig. 4.** Generalization of the stall logic from (a) simple combinational gate, to (b) finite-state machine.

machine is actually synchronous, so its implementation is much easier than that of asynchronous controllers.

The generalization presented here has two key benefits. First, a significant reduction in unnecessary stalls may be obtained, since stalls are no longer caused by unavailability of channels that are not needed for a particular operation. Second, the need for “busy waiting” is greatly reduced. That is, modules that are not producing valid data can be shut down, without fear of stalling their neighbors. As a result, significant savings in power consumption may be obtained.

### 4.3 Generalized Asynchronous Communication Network

As our second contribution, we present ideas for the first approach to a formal framework for the specification and synthesis of arbitrary topologies for GALS communication networks. This approach easily addresses the issues raised in Example 2 above.

Our generalized approach, much like Carloni et al.’s, has the benefit of not requiring any change in how the individual synchronous modules are designed. Thus, synchronous designers can continue to design IP blocks exactly how they are used to designing them. The only changes needed either apply to the wrapper circuits, or to the communication network. For the former, as described in the previous subsection, well-known synchronous techniques can be used. For the latter, several mature asynchronous techniques exist, which are now briefly discussed.

Asynchronous technology can be applied to the design of the communication network at different levels of abstraction. At a lower level, a structural approach can be used to combine predefined structures—such as forks, joins, muxes, demuxes, etc.—to generate complex network topologies. In order to ensure robustness against metastability, the structures immediately interfacing with the synchronous modules could borrow ideas from the mixed-timing approaches of Chelcea and Nowick [18], and Chakraborty and Greenstreet [13]. At a higher level, a behavioral approach can take a high-level description of the communication network, and translate it into a circuit. The translation can be either syntax-directed using predefined blocks (*e.g.*, Tangram [23]), or be synthesis-driven (*e.g.*, Balsa-CUBE [24, 25]). Alternatively, graph-based synthesis approaches can also be used (*e.g.*, MINIMALIST [26] and Petrify [27]). Finally, it is interesting to note that the communication network

can even be implemented as a *synchronous* network [14]; even so, a good degree of asynchrony will be inherent in the latency-insensitive nature of the overall system.

The net impact of the proposed generalization of the communication network is two-fold. First, a significantly greater degree of expressivity is offered for the specification of inter-module communication. Second, the designer is offered much greater freedom to “mix-’n-match” modules of different speeds and different types of interfaces. For example, the designer is able to use multiple instances of a slower module to interface with a faster module, using fork-join structures, as shown in Figure 3. As a result, better overall hardware utilization is achieved, thereby obtaining higher system throughput.

Finally, we would like to mention that some alternatives to point-to-point channels have already been reported. These approaches are based on simple topologies such as shared buses and rings [19], or simple forks and joins [20]. While these are steps in the right direction, they are nevertheless point solutions. Our proposed approach represents an important first step towards providing a formal framework for the specification and synthesis of arbitrary communication networks.

#### 4.4 Formal Modeling of GALS Architectures

We are currently exploring ideas for formal modeling of GALS architectures. One preliminary idea is to use Petri nets for specification of GALS systems. In such a Petri net, an entire synchronous module is denoted by one transition. A firing of that transition represents one operation of its associated synchronous module; no firing means the module is stalled. The arcs of such a Petri net represent communication channels in the GALS system. We hope to provide more details on these ideas once we have explored them fully.

## 5 Other Ideas

In this section, we briefly outline a few ideas that seem interesting to explore in the context of GALS systems.

**Cortadella et al.’s Desynchronization Approach.** Recently, Cortadella et al. [28] proposed an approach for *desynchronization* of synchronous designs. Their contribution is an automated approach that converts a synchronous design into an asynchronous one. An interesting idea is to come up with a similar approach that can convert a synchronous system into a GALS design.

**Automated Partitioning into GALS Architectures.** The problem of partitioning a monolithic synchronous system into a GALS system is an interesting one. One idea for an approach that accomplishes this decomposition is to leverage off of current clock-tree synthesis tools. Using these tools, first a clock tree is generated for the design. Then, the synthesized clock tree can be used to guide the partitioning of the system into distinct synchronous modules for mapping onto a GALS architecture.

**Network-on-a-chip (NoC).** Recently, it has been proposed that on-chip communication in SoC’s is likely to evolve in the direction of full-scale networks on chips [29, 30]. These networks will carry data packets, instead of individual bits or words, and include sophisticated protocols for routing, flow control, and recovery from transmission errors. Our approach lies somewhat between the NoC approach, and the traditional approach of regarding wires as simply carrying electrical signals. It would be interesting to study the pros and cons of using our approach vs. NoC’s.

## 6 Conclusions

This paper presented some initial ideas on a new approach to the design of GALS systems. In particular, two generalizations were proposed to extend the notion of latency-insensitive systems as originally proposed by Carloni et al. The first extension allows much greater flexibility in interfacing a synchronous module with its input and output channels, thereby allowing higher overall system throughput through elimination of unnecessary stalls. The second extension proposes more general communication network topologies than the currently popular point-to-point interconnects. Finally, initial ideas on formally modeling GALS architectures using Petri nets were provided.

## References

1. Carloni, L.P., McMillan, K.L., Saldanha, A., Sangiovanni-Vincentelli, A.L.: A methodology for correct-by-construction latency insensitive design. In: Proc. International Conf. Computer-Aided Design (ICCAD). (1999) 309–315
2. Carloni, L.P., McMillan, K.L., Saldanha, A., Sangiovanni-Vincentelli, A.L.: A methodology for correct-by-construction latency insensitive design. In: Proc. International Conf. Computer-Aided Design (ICCAD). (1999) 309–315
3. Carloni, L.P., Sangiovanni-Vincentelli, A.L.: Performance analysis and optimization of latency insensitive systems. In: Proc. ACM/IEEE Design Automation Conference. (2000) 361–367
4. Carloni, L., McMillan, K., Sangiovanni-Vincentelli, A.: The theory of latency insensitive design. *IEEE Transactions on Computer-Aided Design* **20** (2001)
5. Seitz, C.L.: System timing. In Mead, C.A., Conway, L.A., eds.: *Introduction to VLSI Systems*. Addison-Wesley (1980)
6. Yun, K.Y., Donohue, R.P.: Pausible clocking: A first step toward heterogeneous systems. In: Proc. International Conf. Computer Design (ICCD). (1996)
7. Chapiro, D.M.: Globally-Asynchronous Locally-Synchronous Systems. PhD thesis, Stanford University (1984)
8. Muttersbach, J., Villiger, T., Fichtner, W.: Practical design of globally-asynchronous locally-synchronous systems. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2000) 52–59
9. Sjogren, A.E., Myers, C.J.: Interfacing synchronous and asynchronous modules within a high-speed pipeline. *IEEE Transactions on VLSI Systems* **8** (2000) 573–583
10. Moore, S., Taylor, G., Mullins, R., Robinson, P.: Point to point GALS interconnect. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2002) 69–75
11. Kessels, J., Peeters, A., Wielage, P., Kim, S.J.: Clock synchronization through handshake signalling. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2002) 59–68
12. Winstanley, A.J., Garivier, A., Greenstreet, M.R.: An event spacing experiment. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2002) 47–56
13. Chakraborty, A., Greenstreet, M.R.: Efficient self-timed interfaces for crossing clock domains. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2003)
14. Carloni, L., Sangiovanni-Vincentelli, A.: Coping with latency in SoC design. *IEEE Micro, Special Issue on Systems on Chip* **22** (2002)
15. O’Leary, J., Brown, G.: Synchronous emulation of asynchronous circuits. *IEEE Transactions on Computer-Aided Design* **16** (1997) 205–209
16. Peeters, A., van Berkel, K.: Synchronous handshake circuits. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press (2001) 86–95
17. Jacobson, H.M., Kudva, P.N., Bose, P., Cook, P.W., Schuster, S.E., Mercer, E.G., Myers, C.J.: Synchronous interlocked pipelines. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2002) 3–12

18. Chelcea, T., Nowick, S.M.: Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In: Proc. ACM/IEEE Design Automation Conference. (2001)
19. Villiger, T., Käslin, H., Gürkaynak, F.K., Oetiker, S., Fichtner, W.: Self-timed ring for globally-asynchronous locally-synchronous systems. In: Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. (2003)
20. Zhuang, S., Li, W., Carlsson, J., Palmkvist, K., Wanhammar, L.: Asynchronous data communication with low power for gals systems. In: IEEE International Conference on Electronics, Circuits and Systems. (2002)
21. Nowick, S.M.: Automatic Synthesis of Burst-Mode Asynchronous Controllers. PhD thesis, Stanford University, Department of Computer Science (1993)
22. Nowick, S.M., Coates, B.: Automated design of high-performance asynchronous state machines. In: Proc. International Conf. Computer Design (ICCD), IEEE Computer Society Press (1994)
23. Berkel, K.v., Kessels, J., Roncken, M., Saeijs, R., Schalijs, F.: The VLSI-programming language Tangram and its translation into handshake circuits. In: Proc. European Conference on Design Automation (EDAC). (1991) 384–389
24. Chelcea, T., Bardsley, A., Edwards, D., Nowick, S.M.: A burst-mode oriented back-end for the Balsa synthesis system. In: Proc. Design, Automation and Test in Europe (DATE). (2002) 330–337
25. Chelcea, T., Nowick, S.M.: Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In: Proc. ACM/IEEE Design Automation Conference. (2002)
26. Fuhrer, R.M., Nowick, S.M., Theobald, M., Jha, N.K., Lin, B., Plana, L.: Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines. Technical Report TR CUCS-020-99, Columbia University, NY (1999)
27. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Transactions on Information and Systems **E80-D** (1997) 315–325
28. Cortadella, J., Kondratyev, A., Lavagno, L., Sotiriou, C.P.: A concurrent model for de-synchronization. In: Proc. International Workshop on Logic Synthesis. (2003)
29. Dally, W.J., Towles, B.: Route packets, not wires: On-chip interconnection networks. In: Proc. ACM/IEEE Design Automation Conference. (2001)
30. Benini, L., DeMicheli, G.: Networks on chips: A new SoC paradigm. IEEE Computer **35** (2002) 70–78