POLITECNICO DI TORINO
Repository ISTITUZIONALE

Issues in Implementing Latency Insensitive Protocols

Original
Issues in Implementing Latency Insensitive Protocols / Casu, M.R., Macchiarulo, L.. - STAMPA. - (2004), pp. 1390-1391. (Design, Automation and Test in Europe, 2004. Paris (F) 16-20 February 2004) [10.1109/DATE.2004.1269102].

Availability:
This version is available at: 11583/1410346 since: 2018-03-26T13:58:44Z

Publisher:
IEEE Computer Society

Published
DOI:10.1109/DATE.2004.1269102

Terms of use:

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

Publisher copyright

(Article begins on next page)

25 June 2026

Issues in Implementing Latency Insensitive Protocols

Mario R. Casu and Luca Macchiarulo
Politecnico di Torino, Dipartimento di Elettronica, C.so Duca degli Abruzzi, 24, I-10129 Torino, Italy
mario.casu@polito.it luca.macchiarulo@polito.it

1 Extended Abstract

The performance of future Systems-on-Chip will be limited by the latency of long interconnects requiring more than one clock cycle for the signals to propagate. To deal with the problem L. Carloni *et alii* proposed the *Latency Insensitive Protocols (LIP)*. A design that works under the assumption of zero-delay connections between functional modules is modified in a *Latency Insensitive Design (LID)* by encapsulating them within wrappers ("shells") and connecting them through internally pipelined blocks ("relay stations") complying with a protocol that guarantees identity of behavior [1]. The wrappers perform:
- **Data Validation**: each output channel signals whether the datum therein present has still to be consumed.
- **Back Pressure**: when the pearl is stopped the shell generates a *stop* signal sent in the opposite direction of inputs;
- **Clock Gating**: a module waiting for new data and/or stopped keeps its present state.
Such a protocol was implemented [2] through the introduction of two new signals per channel, *valid* and *stop*. The introduction of the *stop* led us to important theoretical considerations about the minimum memory requirements for the safe implementation of the protocol that are not apparent from the papers by Carloni et alii. Our general conclusion is that since the stop signal cannot be back propagated indefinitely throughout the shells, at least one memory element to save this signal is needed between two shells. This led to the introduction of the "half relay station" whose difference from the normal ("full") relay station is that it has only one register instead of two. Our shell will be simplified since it does not save the incoming stop signals, but we need to add at least one half or one full relay station between two shells. This will guarantee at the same time that no data are lost and will help to improve the performance when necessary.

We employed a slight variant of the original protocol, aimed at optimizing its implementation. In previous works the stop signal is back-propagated regardless of the signals validity, in our implementation stops on invalid signals are discarded. The overall computation can get a significant speedup, and higher locality of management of void/stop signals is ensured. The details of the RTL implementation of relay stations as FSM's, and of the shells can be found in [3], together with the differences from previous proposals.

Our protocol refinement allows precise calculations of important design parameters, such as *System Throughput* and *Transient Length*. The second figure comes up as an interesting consequence of the protocol: after a number of clock cycles that are dependent on the system each part of it behaves in a periodic fashion.

It is natural to associate a direct, possibly cyclic graph to a system of interconnected synchronous processes. There are representative graph topologies whose performance can be easily derived. The simplest topology is a *tree*. The throughput of each node, i.e. the number of valid data per clock cycle, is 1. However, each relay station must be initialized with non valid outputs that must be eliminated flowing toward the primary outputs[1]. Thus the initial latency for each node before firing at full speed can be as much as the longest path in the tree (transient duration).

Another case is what we called "reconvergent inputs" topology. Its behavior differs from that of trees due to implicit loops created by the introduction of reverse-flowing stop signals. Let's consider the simple example of Fig. 1 and follow the system's evolution. "N"s represent non

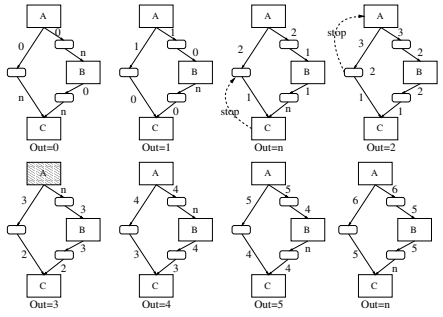

Figure 1. FeedForward Topology Evolution.

valid data, while stops are indicated by dashed arrows and stopped modules by dashed blocks. After the initial transient, the situation becomes periodic, and the output utters

[1] On the other hand, the shells outputs are initialized with valid data.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)
1530-1591/04 $20.00 © 2004 **IEEE**
IEEE COMPUTER SOCIETY

an invalid datum every 5 cycles. The unbalanced number of relay stations in the reconvergent paths forces the longest one to introduce a number of invalid data. A single invalid token gets propagated from top to bottom and then generates a stop signal that goes back on the shortest path every n cycles. The number of invalid data is the difference of relay stations i between the "feedforward" branches. In the present case, $n = 5$, while $i = 1$. The number of valid data every 4 periods is 4 and the throughput is $\frac{4}{5}$. The general formula $T = \frac{m-i}{m}$, where m is the total number of relay station in the loop, plus the number of shells on the path with the highest number of relay station. To get the maximum T from a feedforward arrangement, it is necessary to insert enough spare relay stations to make all converging paths of the same length (**path equalization**).

Graphs containing loops of shells and relay stations as in Fig. 2 are responsible for the worst throughput degradation. The evolution shows a behavior dictated mainly by the ratio between shell and relay station numbers. A maximum of S valid data can be present at a time, out of $S+R$ positions (where R is the number of relay stations in the loop). This justifies the number $\frac{S}{S+R}$ for the maximum throughput. This result is fundamentally the same discussed by Carloni in [5].

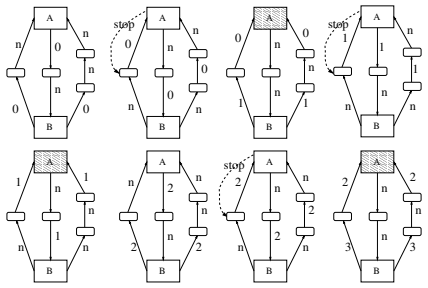

Figure 2. FeedBack Topology Evolution.

The most general topology is a feed-forward combination of self-interacting loops. It is possible to prove that the slowest subtopology (either reconvergent feed-forward or feedback) will force the system to slow down to its speed. The protocol itself will adapt to such a speed without any need for path equalization.

Both to get more insight on the properties of our implementations and to avoid design mistakes, we employed a tool of formal verification in which it was possible to describe our basic blocks at the RT level. We dealt with safety and liveness problems separately.For us a LIP implementation is **safe** iff any composition of blocks will behave in a latency insensitive sense exactly as an equally connected system without and shells and non/pipelined connections. We used the tool SMV [4] to verify that:
- *Any shell elaborates coherent data;*
- *Any shell produces outputs in the correct order;*
- *Any shell does not skip any valid output;*
provided the shell works in an appropriate environment, t.i., all its inputs keep their values on asserted stops. Analogously, for relay stations, we verified:
- *Any relay station produces outputs in the correct order;*
- *Any relay station does not skip any valid output;*
- *Any relay station keeps its output on asserted stops;*
provided the relay station works in an appropriate environment, t.i., all its valid inputs are ordered.

Another issue to be addressed is that of ensuring that deadlock does not occur (*liveness*). Since liveness is topology dependent, we couldn't verify formally the protocol as such. However experiments and some proved results support the following conclusions:
- *Any LID is deadlock free if it has only a feed-forward topology (possibly with reconvergence);*
- *Any LID using only "full" relay stations is deadlock free;*
- *Any LID with full and half relay stations has potential deadlocks iff half relay stations are present in loops;*
The last point is particularly critical. However, there is a possible remedy: in many cases, even though deadlock is not ruled out by general consideration, its injection will never occur. If we simulate the system up to the transient's extinction, either the deadlock will show, or will be forever avoided. And fortunately, the transient length is related to the number of relay stations and shells, and can be predicted upfront. Moreover we are allowed to simulate just the skeleton of the system consisting of stop and valid signals, thus the simulation cost is absolutely negligible. For a relatively limited effort, this strategy can allow high increase in throughput. In our experience, furthermore, the cases that inject deadlocks can be "cured" by low intrusive changes (adding/substituting few relay stations).

To validate our protocol, together with the results exposed in the previous sections, we used many proof-of-concept examples that comprise various combinations of feedforward and feedback topologies. All examples were successfully simulated using a VHDL description of all blocks and an event-driven simulator.

References

[1] L.P. Carloni *et al.*, Theory of Latency-Insensitive Design, IEEE TCAD, vol. 20, No. 9, Sept. 2001, pp. 1059-1076.

[2] L.P. Carloni *et al.*, A Methodology for "Correct-by-Construction" Latency Insensitive Design", Proc. ICCAD 99, pp. 309-315.

[3] M.R. Casu and L. Macchiarulo, A Detailed Implementation of Latency Insensitive Protocols, Proc. FMGALS 2003, Pisa, Italy, Sep. 2003. Available at `http://www.vlsilab.polito.it/~casu`

[4] K.L. McMillan, "Getting Started with SMV," Cadence Berkely Labs, 2001 Addison St., Berkely, CA, March 1999.

[5] L.P. Carloni and A.L. Sangiovanni-Vincentelli, Performance Analysis and Optimization of Latency Insensitive Protocols, Proc. DAC 00, pp. 361-367.

IEEE COMPUTER SOCIETY