Throughput and FIFO Sizing: an Application to Latency-Insensitive Design

Julien Boucaron, Anthony Coadou, Robert De Simone

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Throughput and FIFO Sizing: an Application to Latency-Insensitive Design

Julien Boucaron — Anthony Coadou — Robert de Simone

N° 6919

May 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6919--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Throughput and FIFO Sizing: an Application to Latency-Insensitive Design

Julien Boucaron , Anthony Coadou , Robert de Simone

Thème COM — Systèmes communicants
Équipe-Projet Aoste

Rapport de recherche n° 6919 — May 2009 — 16 pages

Abstract: On-chip communications are a key concern for high end designs. Since latency issues cannot be avoided in deep-submicron technologies, design methodologies need to cope with it. In such a case, precise FIFO sizings are of high interest, to find the right trade-off in between area, power and throughput. This paper provides means to size *optimally* FIFOs while reaching maximum achievable throughput. We apply our algorithms to Latency-Insensitive Designs. Such algorithms can also be used to size FIFOs in other application fields, as for instance Networks-on-Chips. We also revisit the *equalization* process, which introduces as much latencies as possible in the system while preserving global system throughput. This algorithm point out where it is possible to introduce more stage of pipelines while ensuring the maximum throughput of the system. It allows for instance to postpone execution of IP(s) to limit dynamic power peak. We provide a modified algorithm that globally minimizes the number of such introduced latencies.

Key-words: Dimensionnement de FIFO, dimensionnement de buffer, débit, Latency-Insensitive Design, Marked Graphs

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Débit et dimensionnement de FIFO: une application au Latency-Insensitive Design

Résumé : Les communications sur puce sont une limitation importante dans le cadre de design sur puce à grandes performances. Puisque les problèmes de latences ne peuvent être évités dans les technologies de gravure sub-microniques, les méthodologies de design doivent les prendre en compte dès la conception. Dans un tel cadre, le dimensionnement précis des FIFOs est très important, afin de trouver le bon compromis entre la surface occupée, la puissance consommée et le débit.

Ce rapport de recherche introduit comment dimensionner *optimalement* des FIFOs tout en atteignant le débit maximum obtensible. Nous appliquons nos algorithmes au cas du Latency-Insensitive Design. Ces algorithmes peuvent être aussi utilisés dans le cadre de Networks-on-Chips. Nous revisitons aussi le processus *d'égalisation* qui introduit autant de latences que possible dans le système tout en préservant le débit global. Cet algorithme détermine quels sont les endroits où l'on peut rajouter des étages de pipeline tout en assurant les contraintes de débit. Ceci permet par exemple de retarder l'exécution de certains composants afin de limiter les pics de puissance dynamique. Nous proposons une version modifiée de cet algorithme permettant de minimiser globalement le nombre de latences introduites.

Mots-clés : FIFO sizing, Buffer sizing, Throughput, Latency-Insensitive Design, Marked Graphs

1 Introduction

On-chip interconnect is one of the main bottlenecks for high-perfor-mance designs: required bandwidth is growing far away than what standard busses can support. Moreover, interconnect delays exceed the mean clock rate of each IP blocks, and will continue to grow in the future. FIFOs and flow-control (also called *back-pressure*) are needed for high performance ASICs, multi-clock designs and Networks-on-Chips.

Determining FIFO sizes is a key concern in order to maximize design throughput, and to minimize area footprint while maintaining acceptable power and temperature requirements.

This paper describes how we can determine *optimal* FIFO sizes in case of single clock IPs, interconnected together with fixed integer latencies. *Optimal* in our case means to achieve the maximum throughput of the system, while minimizing the sum of FIFO sizes, and take into account the flow-control induced by FIFOs (distributed or atomic ones). We show an application of our results to the specific case of Latency-Insensitive Design. Our results may also apply to different cases, and especially on Networks-on-Chips.

In next section, we briefly describe Latency-Insensitive Design, which is a specific case of synchronous design with non-uniform latencies, re-synchronized through distributed FIFOs (called *relay stations*) and *shell wrappers*.

Then, we establish the link that exists between Latency-Insensitive Design and Marked Graphs with finite capacities. We briefly recall some useful results on static scheduling of Marked Graphs. We introduce an algorithm to compute minimum FIFOs sizes while achieving maximum throughput, and few variants for different set of constraints.

After, we recall the *equalization* process, which is an algorithm able to push as much as possible latencies on a given design while maintaining its maximum throughput; that is to say, to point out where data flows can be slowed down without affecting the global throughput. This provides hints for re-pipelining a design. We present a revised equalization, which is able to *minimize* globally the number of such introduced latencies: showing opportunities of re-pipelining giving the smallest area footprint (and still maintain original throughput).

Finally, we discuss results, experiments and implementations.

2 Latency-Insensitive Design

This section introduces briefly the Latency-Insensitive Design (also known as Latency-Insensitive Systems, Latency-Insensitive Protocols or Synchronous Elastic Flow).

Latency-Insensitive Design (LID) [5] allows to split apart synchronous module functionalities and their communication constraints. It ensures that communication latencies will not interfere with necessary pre-conditions of the synchronous modules. LID is a composition of *patient processes*: a synchronous process where its functionality only depends on signal values, but not on their reception times. Composition of patient processes is itself patient, as shown in Carloni's seminal paper [6]. Since most of IPs do not match this requirement, LID has two kind of basic blocks (cf. Figure 1).

A *shell* wraps each computation block ; the latter is named *pearl*. Shell function is two-fold: 1) inputs synchronization, the pearl is executed as soon as all input data are present; and 2) outputs propagation, computation results are emitted only if downward receivers are able to store them, in order to avoid over-writings and data losses.

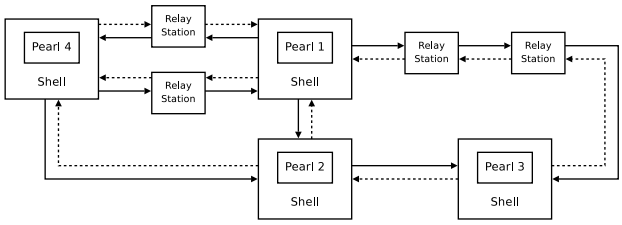

Figure 1: Example of latency-insensitive system.

Signals cannot go through long wires within a single clock cycle in practice. Such long wires are split up in shorter ones using *relay stations*; they are patient processes, ensuring data storage from a clock cycle to the next one. Long wires are segmented in a such way that each relay station is only "distant" from its upward emitter and downward receiver of less than a clock cycle. In order to avoid throughput shrinking, the minimum capacity of a relay station is supposed to be two [6].

Finally, LID relies on two hypotheses: pearls can be stalled within one clock cycle and all latencies are integers.

The design is a composition of modules, communicating through point-to-point channels. This modules are wrapped by shells, used as interfaces between relay stations. Neither blocks placements nor communication latencies are known *a priori*; they have to be estimated and refined during placement and routing. Channels are iteratively split and relay stations inserted until it reaches a fixed point. Carloni *et al.* detail this methodology in [5].

Data flows can be controlled in two ways:

- With dynamic scheduling [4, 5, 8, 11] using flow-control mechanism (called *back-pressure*): when a component cannot handle more data, it sustains a *stop* signal on the upward channel till the register is freed. This solution is flexible enough to prevent data losses, even if a pearl unexpectedly stall. A drawback of the dynamic protocol is that it precisely needs control signals. Such overhead, especially wiring, may be expensive to place and route in an already complex design.

- An alternative is to statically schedule pearls [4, 9]. Execution instants can be computed at compile time, and each component knows exactly on its own when it will receive a new data. But it can also take a lot of silicon area to implement such static scheduled pearls: big shift registers may be required, depending on the schedule sequence to store.

3 Latency-Insensitive Design as Marked Graphs

In the sequel, LID will be modelled as Marked Graphs. We first recall in this section the main definitions and results on Marked Graphs. Then, we show with examples how bounded capacities may shrink the graph throughput.

3.1 Marked Graphs with Finite Capacities

Marked Graphs (also called *Event Graphs*) are a quite useful subset of Petri nets, introduced and first studied in [10, 13]. They were later *timed*, introducing latencies, in [15].

Definition 1 (Timed marked graph)**.**
A Timed Marked Graph *(TMG) is a structure* $\langle \mathcal{N}, \mathcal{P}, M_0, L \rangle$*, such that:*

- $\mathcal{N}$ *is a finite set of* vertices*, or* computation nodes*.*
- $\mathcal{P} \subset \mathcal{N} \times \mathcal{N}$ *is a finite set of* arcs*, or* places*.*
- $M_0 : \mathcal{P} \to \mathbb{N}$ *is the function that assigns an initial marking (quantity of data abstracted as* tokens*) to each place*[1]*.*
- $L : \mathcal{P} \to \mathbb{N}$ *is a function that assigns a weight to each place. In our case, this weight is the latency for a token to go from the input to the output node.*

For each place p, we note $\bullet p$ (resp. $p\bullet$) its tail (resp. its head), such that $\bullet p = \{n \in \mathcal{N} / \exists n' \in \mathcal{N},\ p = (n, n')\}$ (resp. $p\bullet = \{n \in \mathcal{N} / \exists n' \in \mathcal{N},\ p = (n', n)\}$). TMG are bipartite directed graphs, where $\forall p \in \mathcal{P},\ |\bullet p| = |p\bullet| = 1$. In other words, places have a single producer and a single consumer, thus leading to an absence of *conflicts*. This also means that a TMG is *confluent*: for all sets of activated nodes, the firing of any of this nodes does not remove another node from this subset than itself. This means that all firing rules define a partial execution order compatible with the *as soon as possible* (ASAP) firing rule: a node is fired as soon as it is enabled.

We recall some essential results on Marked Graphs given by Commoner *et alii*. Proofs are deferred to [10].

Lemma 1 (Token count)**.**
The token count of a directed circuit does not change by vertex firing.

Theorem 2 (Liveness)**.**
A marking is live if and only if the token count of every circuit is positive.

Theorem 3 (Firings)**.**
If there exists a firing sequence for a graph whose underlying undirected graph is connected, and this sequence leads back to the initial marking, then all nodes have been fired an equal number of times.

Real-life systems have bounded memory capacities. In order to model this constraints, place *capacities* may be added to Definition 1. Then, we define the function $K : \mathcal{P} \to \mathbb{N}^*$, assigning to each place the maximum number of tokens that it can contain at a time.

TMG with capacities can be transposed in an equivalent without capacities, introducing *complementary* places [1, 2]. If $p_1 = (a, b)$ is a place with a capacity $K(p_1)$, its equivalent without capacity $p_2 = (a, b)$ and the corresponding complementary place $\overline{p_2} = (b, a)$ have initial markings such that $M_0(p_2) = M_0(p_1)$, and $M_0(\overline{p_2}) = K(p_1) - M_0(p_1)$, as shown in Figure 2. In the sequel, for any set of places $\mathcal{P}$, we will note $\overline{\mathcal{P}}$ the set of its complementary places.

This means that, for instance, if it exists a circuit in a capacity-bounded TMG whose initial marking saturates its place capacities ($\forall p,\ M_0(p) = K(p)$), then it is equivalent

[1] We recall that $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$.

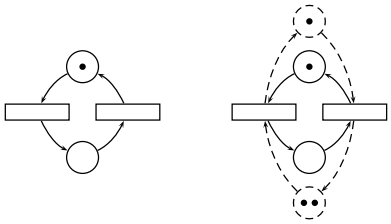

Figure 2: A Marked Graph whose places capacities are 2-bounded (*left*) and its equivalent without capacities (*right*).

one to a complementary circuit with a void initial marking, thus violating Theorem 2. In other words, an overflow (or *livelock*) in a TMG with capacities, is equivalent to a starvation (or *deadlock*) in its equivalent one without capacities.

It is worth noticing that the property of acyclicity loses its sense when talking about a graph with finite capacities: it behaves the same way as its strongly connected equivalent, where complementary places introduce circuits.

3.2 Example-Based Throughput Limitations

A LID can be modeled as a TMG: a shell with its pearl is abstracted as a computation node, and a relay station with its two buffers corresponds to a place of capacity two.

First, we do not take into account place capacities; we consider that places are unbounded channels and recall briefly known results on scheduling of Marked Graphs. Then, we show on two simple examples why finite capacities can slow down throughput of cyclic and acyclic systems.

3.2.1 Infinite-Capacity Places

A lot of interesting scheduling results exists on top of Marked Graphs with unbounded places and ASAP firing rule [2, 16].

Definition 2 (Rate)**.**
We denote the rate *of a circuit the ratio:* $\frac{\#tokens}{\#latencies}$

Theorem 4 (Throughput)**.**
The throughput *of a Marked Graph is its maximum execution speed. The throughput of a strongly connected graph equals the minimum rate among its circuits. The throughput of an acyclic graph is* 1.

Proof. A Marked Graph cannot run faster than such throughput [16]. [2] has shown that Marked Graph executions can reach such throughput. □

Definition 3 (Critical circuit)**.**
A circuit is said critical *if its rate is equal to the graph throughput.*

3.2.2 Finite-Capacity Places

Finite capacities may slow down the graph throughput, compared to a topologically equivalent graph with unbounded buffers. We give an example in Figure 3 of a 2-bounded graph. Plain places belong to the graph with finite capacities, while dashed

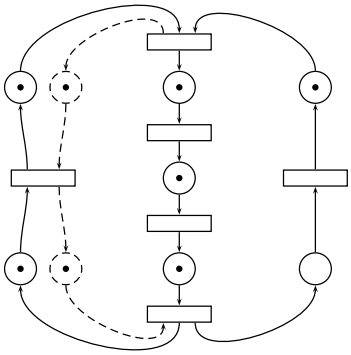

Figure 3: Strongly connected Marked Graph with unitary latencies, whose throughput is limited to $3/4$ due to 2-bounded places.

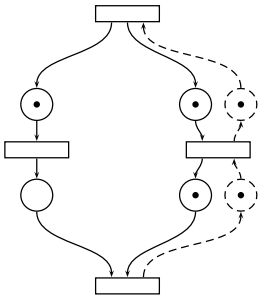

Figure 4: Acyclic 2-bounded graph whose capacities shrink the throughput down to $3/4$.

places are complementary places. We just show two of them (dashed ones), the most significant ones.

We suppose null places latencies. Initially, the graph has two circuits: the left circuit has a throughput of $5/5 = 1$, and the right one has a throughput of $4/5$.

However, if we construct its equivalent graph without capacity, we have to introduce complementary places, such that their initial marking equals the capacities minus the markings of original places: this leads to introduce a circuit whose throughput equals $3/4$, slowing down the system because of a lack of capacity.

In a graph with infinite capacities, each lock or slowdown is due to (temporary) starvation: a node produces data as long as it receives inputs. It stops only when awaiting tokens to process. Then, as soon as we introduce finite capacities, we introduce new constraints in addition to the previous ones: nodes are not only awaiting inputs, they also need available storage for their outputs. Dynamic LID has back-pressure signals to handle this problem; they have to be modeled while abstracting a LID into a Marked Graph.

At first sight, it seems that the throughput of a directed acyclic graph (DAG) is 1, as stated in [7]. However, the introduction of complementary places creates cycles. Then, a DAG may have a throughput lower than 1, according to its initial marking, as illus-

trated in Figure 4. In this example, branches do not have a balanced marking, causing a *bubble* in the left branch to get back in the right one. When we consider the *complemented* graph, there is a circuit with a throughput $3/4$: the one passing through the *back-pressure* dashed path in the right branch, getting back into the left branch. Only trees with capacities of two or more escape this rule, since introducing complementary arcs do not create more simple circuits than an arc and its complementary.

In both [7] and [9], the authors use the *maximum cycle mean* problem [2, 12] to compute the throughput of LID systems. Authors in [3] use costly circuits enumeration to obtain its throughput. However, none of these papers discuss the throughput slowdown due to lack of capacity. In [7], a DAG is supposed to have a throughput of 1, which is false with our definition of throughput as we shown previously on an instance. We need to clarify what is the throughput of a LID system. If we assume that the throughput is computed without taking into account the back-pressure, then the previous statement about the DAG is true. However, since LID has relay-stations with bounded capacity of 2, such definition of throughput is an *overestimation* of what will be the real and effective throughput of system. Lack of place capacities (or relay-stations in LID case) may cause potential slowdown, as we have explained previously.

4 Capacities and throughputs

As said above, the maximum achievable throughput is bounded by the ratio $\frac{\#tokens}{\#latencies}$, but may be slower because of lack of place capacities. In this section, we detail such maximum throughput taking care of place capacities. Next, we provide our algorithm to compute for each place its capacity, in order to achieve maximum throughput while minimizing the global number of capacities introduced.

4.1 Maximum Throughput with Given Capacities

Definition 4 introduces necessary notations. Theorem 5 allows to compute the exact system throughput, with respect to its capacities.

Definition 4 (Complemented graph)**.**
If $\mathcal{G} = \langle \mathcal{N}, \mathcal{P}, M_0, L, K \rangle$ is a connected timed marked graph with finite capacities, the complemented graph *$\mathcal{G}' = \langle \mathcal{N}', \mathcal{P}', M_0', L' \rangle$ is its equivalent with complementary arcs and no capacities such that:*

$$\begin{aligned} \mathcal{N}' &= \mathcal{N} \\ \mathcal{P}' &= \mathcal{P} \cup \overline{\mathcal{P}} \\ \forall\, p \in \mathcal{P},\ M_0'(p) &= M_0(p) \\ \text{and } M_0'(\overline{p}) &= K(p) - M_0(p) \\ \forall\, p \in \mathcal{P},\ L'(p) &= L'(\overline{p}) = L(p) \end{aligned}$$

In order to shorten notations, we write in the sequel $L(\mathcal{C})$ or $L'(\mathcal{C})$ of a circuit $\mathcal{C}$ (resp. $M_0(\mathcal{C})$ or $M_0'(\mathcal{C})$) the sum of its place latencies (resp. initial markings). For instance, $M_0(\mathcal{C}) = \sum_{p \in \mathcal{C}} M_0(p)$.

Theorem 5 (Maximum throughput)**.**
Let $\mathcal{G} = \langle \mathcal{N}, \mathcal{P}, M_0, L, K \rangle$ be a connected timed marked graph with finite capacities,

and $\mathcal{G}' = \langle \mathcal{N}', \mathcal{P}', M'_0, L' \rangle$ *its complemented graph. The maximum reachable throughput* $\theta(\mathcal{G})$ *of* $\mathcal{G}$ *is:*

$$\theta(\mathcal{G}) = \min_{\mathcal{C} \in \mathcal{G}'} \left(\frac{M'_0(\mathcal{C})}{|\mathcal{C}| + L'(\mathcal{C})}, 1 \right)$$

Proof. The rate $\theta(n)$ of a node n is usually defined by:

$$\theta(n) = \lim_{i \to +\infty} \frac{i}{T_n(i)}$$

where $T_n(i)$ is the elapsed time up to the i^{th} firing of n. For each circuit $\mathcal{C}$, each node cannot be fired more than once per instant; that is to say $i \leq T_n(i)$. So the global throughput is at most 1, and we ignore the case where $M'_0(\mathcal{C}) > |\mathcal{C}| + L'(\mathcal{C})$.

We now consider $\mathcal{C}$ with $M'_0(\mathcal{C}) \leq |\mathcal{C}| + L'(\mathcal{C})$. A Marked Graph can be ultimately periodically scheduled with the ASAP firing rule. And as mentioned earlier, all firing rules are compatible with the ASAP one. Then, we can restrict the study on a single period. The minimum time for a token to go round $\mathcal{C}$ is $|\mathcal{C}| + L'(\mathcal{C})$. This sum of all nodes and places latencies in $\mathcal{C}$ is the period length. Over a period, so that each token go round the circuit, each node is fired $M'_0(\mathcal{C})$ times (cf. Lemma 1 and Theorem 3).

To summarize, the rate of a circuit $\mathcal{C}$ is:

$$\theta(\mathcal{C}) = \min \left(\frac{M'_0(\mathcal{C})}{|\mathcal{C}| + L'(\mathcal{C})}, 1 \right)$$

Since $\mathcal{G}'$ is the complemented equivalent of $\mathcal{G}$ and $\mathcal{G}$ is a (possibly acyclic) connected graph, $\mathcal{G}'$ is strongly connected because of complementary places, and $\theta(\mathcal{G}) = \theta(\mathcal{G}')$. The throughput of a strongly connected graph equals the rate of its slowest circuit (cf. Theorem 4). □

4.2 Required Capacities for Optimal Throughput

Additional capacities may be required in order to reach the maximum throughput. Number and positions of such capacities can be computed with the following Integer Linear Program (ILP):

$$\min \sum_{p \in \mathcal{P}} K(p)$$

with for all circuit $\mathcal{C}$ in $\mathcal{G}'$ such that $\mathcal{C}$ contains at least one complementary place:

$$M'_0(\mathcal{C}) \cdot \big(|\mathcal{C}_c| + L'(\mathcal{C}_c)\big) - M'_0(\mathcal{C}_c) \cdot \big(|\mathcal{C}| + L'(\mathcal{C})\big) \geq 0$$

and for all p in $\mathcal{P}$:

$$K(p) \geq 2$$

where $\mathcal{C}_c$ is the "latency-critical" circuit (without taking capacities into account). M'_0 and L' are defined in Theorem 5.

As stated previously, the maximal throughput of the graph is the throughput computed on its complemented equivalent. If there is a slowdown in the system, it is caused by a circuit passing through at least one complementary place, that is to say a place where back-pressure shrinks the effective throughput. Intuitively, the ILP states

that we want to minimize global capacity count for FIFOs under the following set of constraints: for each circuit $\mathcal{C}$ having at least one complementary place, we can add further capacity until we reach the maximal throughput of the graph. We set $K(p)$ greater or equal to two, so that general LIP properties apply [6].

Previous inequations can be slightly modified if we want to minimize the number of capacities to reach (if possible) a specified throughput θ. Then, $M'_0(\mathcal{C}_c)$ and $|\mathcal{C}_c| + L'(\mathcal{C}_c)$ can be replaced by M and L respectively, such that $\theta = M/L$.

Such ILP formulation is also interesting to introduce constraints on capacities. Because of blocks placement and routing, it is not always possible to add more capacities to a single place at a given location. One can add constraints of the form $K(p) \leq K_{max}(p)$, for example.

4.3 Algorithm

Now, we provide the algorithm to compute minimum capacities for each place in order to achieve maximum throughput of the system.

1. Compute maximum throughput on the *not-complemented* graph using for instance Bellman-Ford algorithm (or Yen's algorithm [18] since we do not have any negative circuit).
2. Build *complemented* graph.
3. Enumerate all directed circuits having at least one complementary arc in the *complemented* graph. We use a modification of Johnson's algorithm [14] that saves some memory: to detect circuits this algorithm uses a clever depth-first search algorithm. Once a circuit is found, we check the previous structural criterion.
   We can optimize this further, in case of LID, we remove simple circuits composed of an arc and its complementary one. LID assumes that there is at most one initial token per place, thus such circuits have a throughput of one. The modification of Johnson's algorithm is straightforward, and remove at least $\left|\mathcal{N}\right|$ circuits from potential $\left|\mathcal{N}\right|!$ circuits.
4. Build and solve previous formulation of the ILP.

The LID optimization is due to the fact that places will have to be expanded to match LID requirements. Relay stations are separated by wire latencies of at most one, so places with latencies of two or more will have to be expanded [15]. A general example of expansion is given in Figure 5.

Thus, a place p of latency $L(p)$ will be synthesized as a row of $L(p)$ relay stations, with a total capacity of $2L(p)$, plus the inner registers of wrappers (since node latencies are unitary). A cycle $\mathcal{C}$ composed of a place p and its complementary $\overline{p}$ will have a throughput of:

$$\begin{aligned}\theta(\mathcal{C}) &= \frac{M'_0(\mathcal{C})}{|\mathcal{C}| + L'(\mathcal{C})} = \frac{M_0(p) + K(p) - M_0(p)}{2 + 2L(p)} \\ &= \frac{2 + 2L(p)}{2 + 2L(p)} = 1\end{aligned}$$

That cannot shrink the global throughput which is at most equal to one. Thus, we can remove safely all such simple circuits.

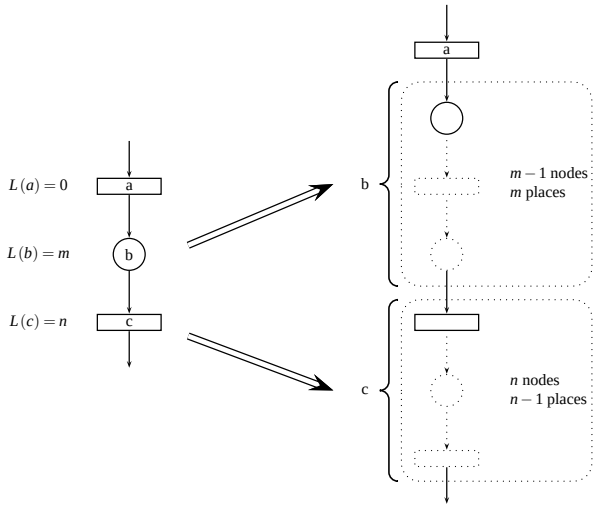

Figure 5: Latency expansion.

5 Equalization

In this section, we recall the *equalization process* introduced in [3], and we present a revised one in order to minimize the number of added places. The equalization process is an algorithm that adds as much as possible places in the graph while maintaining the throughput of the system. The addition of such places gives an hint where there is a positive slack, so that the designer can add more pipeline stage(s) while ensuring the same performance of the whole system. Also, such positive slack can be used to postpone an execution, in order to *smooth* dynamic power and flatten temperature *hot-spots* that are critical for leakage power (leakage power is exponential with temperature).

5.1 Standard Equalization

The equalization process is built using ILP. The algorithm works as follows:

- Compute maximum throughput of the *complemented* graph.
- Enumerate all circuits (using in our case Johnson's algorithm [14]): circuits are used to build constraints between places that can be shared by two or more circuits.
- Build and solve the ILP.

The ILP formulation is of the form:

$$\max \sum_{p \in \mathcal{P}} a(p)$$

with the constraints that for each circuit $\mathcal{C}$ in graph $\mathcal{G}$:

$$(|\mathcal{C}_c| + L(\mathcal{C}_c)).M_0(\mathcal{C}) - M_0(\mathcal{C}_c).\left(|\mathcal{C}| + L(\mathcal{C}) + \sum_{p \in \mathcal{C}} a(p)\right) \geq 0$$

and the following bounds for each place p:

$$a(p) \geq 0$$

where $a(p)$ are latencies added to each place p.

Intuitively, the ILP states that we want to maximize the number of introduced latencies on each place (arc) while ensuring for each circuit that its throughput will be at least as fast as the throughput of the system. Dependency relations exist between circuits through *shared* places (arcs) that is why we need to enumerate all circuits.

We use ILP for its versatility to introduce new constraints to suit a given need: for instance, a designer run the equalization process and there is a set of arcs he/she does not want any additional latency, such modification is done quickly on the ILP formulation.

5.2 Revised Equalization

This revised equalization is a variant of standard equalization that minimizes added latencies, using arcs shared amongst different circuits. The modification is straightforward; it consists in introducing a weight for each arc in the previous formula that we want to maximize.

The function to maximize is:

$$\max \sum_{p \in \mathcal{P}} w(p).a(p)$$

where $w(p)$, the weight associated to the arc, is simply the number of occurrences of circuits it belongs to. The set of constraints stays unchanged.

Correctness criterion: both standard and revised equalization have the same optimum value. The underlying intuition is has follows:

- Let us assume that there is no circuit sharing the same place/arc. Then the weight equals one, we have the same sum to maximize.
- Now, let us assume that there is a set of circuits x sharing a place/arc, and we can add 1 latencies on each circuits. In the standard equalization, we attempt to put at different locations in order to maximize the gain, so we will add x latencies. However, in the revised equalization, if it is possible to add a latency to this shared place/arc, then we will add it. In both cases, we will have the same gain x.

6 Discussion

Johnson's algorithm One of the building block is Johnson's algorithm, enumerating all elementary circuits. This algorithm and different variants are used for all of our algorithms. This algorithm has been implemented using Boost Graph Library [17] and C++. As detailed in Johnson's seminal paper [14], worst case graphs for enumerating all circuits are complete graphs, in such case the number of circuits is huge, up to $|\mathcal{N}|!$. But, in real-life such complete graphs are not usual. In Table 1, we provide experiments for complete graphs, run on an Intel Xeon 3Ghz, 8GB RAM, Linux x64, gcc4.3.

Memory is of course an issue in a such case. We have implemented a variant of the algorithm just to enumerate circuits on-the-fly. We have been able to enumerate around 10^{15} circuits; run time of such experiment is about 5 to 6 days. It is also possible to parallelize the algorithm, using a parallel depth-first search.

Table 1: Enumerating circuits of complete graphs

# circuits	# nodes	Time (s)	Memory (MB)
125673	9	0.170	20
1112083	10	1.730	190
10976184	11	17.0	1600

Table 2: Number of added latencies by standard equalization vs revised one

Examples	Standard (latencies)	Revised (latencies)	%
Simple DAG	9	7	-22
No Share	14	14	0
Shared	14	7	-50
Test4	45	27	-40

Throughput w.r.t. capacities We have structurally characterized the capacity bottleneck, for the maximal theoretical throughput, as circuits containing at least one complementary place. We use this structural hint to implement an efficient variant of Johnson's algorithm that enumerates only circuits having this previous property. As we said, the number of circuits can be huge, up to $|\mathcal{N}|!$; we reduce memory requirements of $|\mathcal{N}|$ circuits in such a case. This helps the ILP solver (in our case lp_solve) to ramp up a bit more. Notice that we used a notation of the form $|\mathcal{C}|+L(\mathcal{C})$ throughout the paper to mention latencies. This is due to the fact that we have supposed that the execution of a pearl takes one clock cycle, in order to match early LID implementations. This can be modified if one suppose combinatorial pearls (null firing latency), or integer latencies as well.

Equalization Both equalization algorithms have been implemented using previous Johnson's algorithm implementation and the ILP engine lp_solve. Table 2 shows results of both algorithms. The worst case for the revised algorithm is when there is no circuits share places. Then both standard and revised algorithm have the same solution. Figure 6 depicts *Test4*. As shown in the right strongly connected component, we have a hierarchy of circuits with shared paths. In this case, the revised equalization algorithm is much more efficient.

7 Conclusion

This papers introduces an algorithm to compute *optimal* FIFO sizes while maintaining maximum achievable system throughput, in case of single clock synchronous systems, interconnected with fixed integer latencies.

We establish the link between such synchronous systems and Marked Graphs (MG) with bounded-capacity places. After, we briefly recall results about throughput of MGs with unbounded capacity places. Then, we recall a simple structural transformation to convert a MG with bounded capacities into a MG with unbounded capacities: for each place, it introduces a new *complementary* place in reverse direction, with an appropriate marking. This creates a circuit between the initial place and the *complementary* one that ensures the capacity constraint of the initial place.

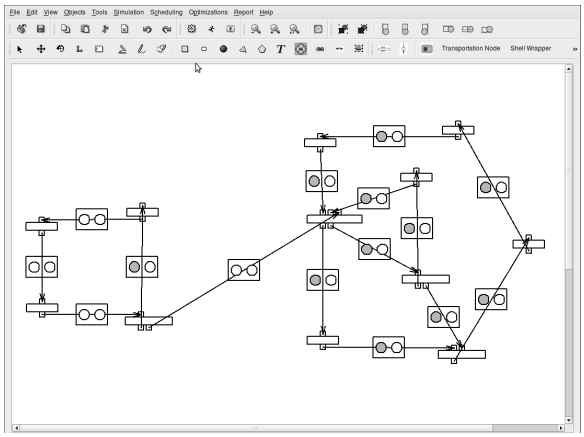

Figure 6: Screenshot of example Test4, designed with our tool.

This graph transformation enables a simple structural criterion to find out circuits causing throughput degradation, because of bounded FIFO capacities: circuits with at least one complementary place. We provide a variant of Johnson's algorithm using this criterion to enumerate only such circuits. After, using previous algorithm, we provide an algorithm using Integer Linear Programming to compute the *optimal* size of each FIFO. Hence, we obtain the maximal throughput while minimizing the sum of FIFOs sizes. We also provide an additional simple structural criterion in case of Latency Insensitive Design (a specific case of MG with bounded capacity places). This allows to remove more unneeded circuits: we describe an algorithm using this additional criterion.

Next, we briefly recall the *equalization* process. This algorithm inserts in the system as much latency as possible, while maintaining its maximum achievable throughput. This provides hints where a designer can add further pipeline stages without affecting the system throughput. It also enables to defer execution of some parts of the system, in order to minimize dynamic power peaks. We provide a revised equalization process that globally minimizes the sum of such latencies. It shows *best* locations where to re-pipeline the design with minimum impact on area.

We apply our results on Latency-Insensitive Design (LID), but such results are not limited to LID and can be used in Network-on-Chips, and other fields.

Contents

References

[1] Charles André. Use of the behaviour equivalence in place-transition net analysis. In *Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets*, pages 241–250, London, UK, 1982. Springer-Verlag.

[2] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity*. John Wiley & Sons Ltd, Chichester, West Sussex, UK, 1992.

[3] Julien Boucaron, Jean-Vivien Millo, and Robert de Simone. Latency-insensitive design and central repetitive scheduling. In *Proceedings of the 4th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE'06)*, pages 175–183, Napa Valley, CA, USA, July 2006. IEEE Press.

[4] Julien Boucaron, Jean-Vivien Millo, and Robert de Simone. Formal methods for scheduling of latency-insensitive designs. *EURASIP Journal on Embedded Systems*, 2007(1), 2007.

[5] Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha, and Alberto L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency-insensitive design. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'99)*, pages 309–315, Piscataway, NJ, USA, November 1999. IEEE.

[6] Luca P. Carloni, Kenneth L. McMillan, and Alberto L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, September 2001.

[7] Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency insensitive systems. In *Proceedings of the 37th Conference on Design Automation (DAC'00)*, pages 361–367, New York, NY, USA, 2000. ACM.

[8] Mario R. Casu and Luca Macchiarulo. A detailed implementation of latency insensitive protocols. In *Proceedings of the 1st Workshop on Globally Asynchronous, Locally Synchronous Design (FMGALS'03)*, pages 94–103, September 2003.

[9] Mario R. Casu and Luca Macchiarulo. A new approach to latency insensitive design. *Proceedings of the 41st Annual Conference on Design Automation (DAC'04)*, pages 576–581, 2004.

[10] Frederic Commoner, Anatol W. Holt, Shimon Even, and Amir Pnueli. Marked directed graph. *Journal of Computer and System Sciences*, 5:511–523, October 1971.

[11] Jordi Cortadella, Mike Kishinevsky, and Bill Grundmann. Synthesis of synchronous elastic architectures. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC'06)*, pages 657–662, New York, NY, USA, 2006. ACM.

[12] Ali Dasdan and Rajesh K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17:889–899, 1998.

[13] Hartmann J. Genrich. *Einfache Nicht-Sequentielle Prozesse.* PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 1970.

[14] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[15] Chander Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1973.

[16] Raymond Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, 1968.

[17] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[18] Jin Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general network. *Quart. Appl. Math.*, 27:526–530, 1970.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399