# Exploiting Local Logic Structures to Optimize Multi-Core SoC Floorplanning

Cheng–Hong Li, Sampada Sonalkar, and Luca P. Carloni

Department of Computer Science - Columbia University in the City of New York

*Abstract*—We present a throughput-driven partitioning algorithm and a throughput-preserving merging algorithm for the high-level physical synthesis of latency-insensitive (LI) systems. These two algorithms are integrated along with a published floorplanner [5] in a new iterative physical synthesis flow to optimize system throughput and reduce area occupation. The partitioning algorithm performs bottom-up clustering of the internal logic of a given IP core to divide it into smaller ones, each of which has no combinational path from input to output and thus is legal for LI-interface encapsulation. Applying this algorithm to cores on critical feedback loops optimizes their length and in turn enables throughput optimization via the subsequent floorplanning. The merging algorithm reduces the number of cores on non-critical loops, lowering the overall area taken by LI interfaces without hurting the system throughput. Experimental results on a large system-on-chip design show a 16.7% speedup in system throughput and a 2.1% reduction in area occupation.

## I. INTRODUCTION

Latency-insensitive design (LID) has been proposed as a correct-by-construction design methodology for synchronous SoCs [2], [4]. LID provides a sound way to help designers cope with the fact that in nanometer technologies SoCs are increasingly becoming distributed systems due to the impact of global communication delays [12]: LID enables the automatic application of *wire pipelining* while preserving the system behavior, simplifies reuse of IP cores, and can be extended to handle not only communication- but also computation-latency variations [8]. Hence, it facilitates the design space exploration of micro-architectures [14] and helps to bridge the gap between system-level design and physical synthesis [27]. These advantages are a result of the flexibility that LID adds to the register-transfer level (RTL) abstraction through the separation of communication and computation, a form of orthogonalization of concerns [16]. In a latency-insensitive (LI) system each core (which can be a complex FSM, a pipelined datapath, an SRAM. . .) is encapsulated by a simple interface circuit called *shell*. Shell-core pairs exchange data via communication channels that are governed by a latency-insensitive protocol. The protocol decouples the implementation of the channels from the implementation of the cores. In particular, at later stages of the design process, the timing exceptions that may arise due to the presence of long wires implementing the channels can be fixed by pipelining them with the insertion of special sequential repeaters called *relay stations* (RS), without the need of changing any core implementation [2].

In this paper we present the first work that addresses the combined optimization of shell encapsulation at RTL and relay station insertion at physical-synthesis level for LID. We do so by exploiting the *logical structure* of the cores and the *physical information* on the core locations provided by floorplanning.

**A motivational example.** Fig. 1(a) shows a LI system with four shell-core pairs connected by point-to-point, unidirectional channels. Each core can be an arbitrarily-complex sequential module as long as it satisfies the requirement that it can be *clock gated*. The shell dynamically controls the operations of the core by deciding whether to stall it or
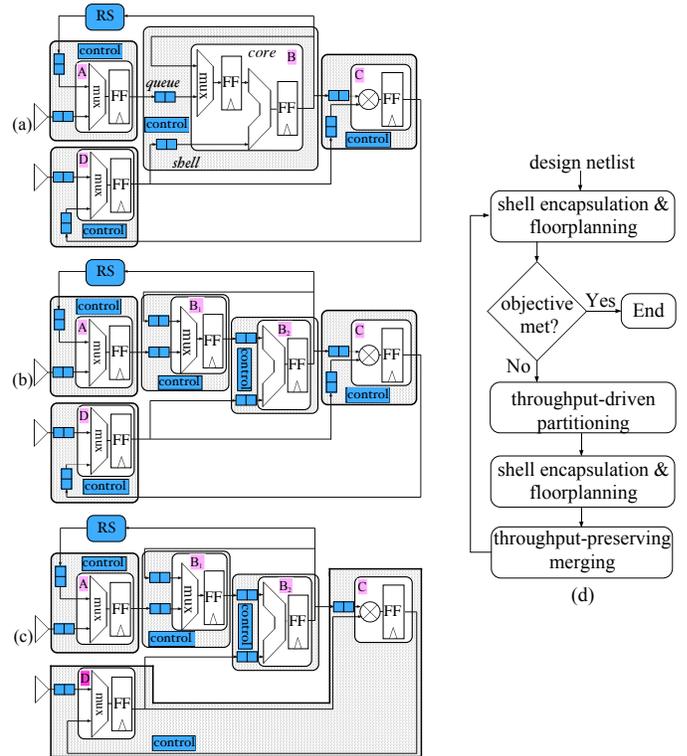


Fig. 1. (a) A simple LI system with four cores and one RS; its throughput is $2/3 = 0.67$. (b) The same system with a finer-grained shell encapsulation where core $B$ is split in two smaller cores $B_1$ and $B_2$; throughput is improved to $3/4 = 0.75$. (c) The LI system maintains the same throughput after core $C$ and $D$ are merged. (d) The proposed high-level physical synthesis flow.

*fire* it based on the value of the flow-control signals on the input/output channels. Data communicated over a channel is labeled by a bit signal indicating whether the current data is valid or not. At each clock cycle the shell fires the core if and only if each input channel presents a new valid data token (*AND-firing semantics*). Otherwise, it *stalls* the core through clock gating while storing valid data having arrived in its input queues (for future processing) and putting void data on each output channel. Since the queues have limited storage capability, a *stop* bit signal is transmitted backward on each channel whenever a downlink shell needs to request an uplink shell to slow down the production of good data (*backpressure*).

At the implementation stage, the wires of a channel with delay longer than the target clock period can be pipelined by one or more RSs (as in Fig 1(a)). An RS is a clocked buffer with unit latency, two-fold storage capacity, and simple flow-control logic. By processing the void and stop bit signals according to the latency-insensitive protocol, the flow-control logic of the shells and RSs can accommodate any variations of delay on *inter-core* wires while guaranteeing that the functional behavior of the original synchronous system is preserved without the need of changing any part of the *intra-core* logic design (*semantics preservation*) [2].

However the AND-firing semantics and the wire-pipelining by RS insertions may negatively affect the system performance in terms of the *data processing throughput* (the average number of valid data tokens processed per unit time) [3], [22].

Since RSs are memory elements added after RTL design is finished, each of them must be initialized with a void data token (a "bubble") so that the system behavior is preserved. If an RS is inserted on a channel which is on a feedback loop, the void data circulates indefinitely in the loop (due the AND-firing semantics), thus stalling the cores periodically.

The data processing throughput of a LI system depends on the locations of RSs and the system-level topology determined by the cores and the communication channels [3], [22]. Each feedback loop in the system imposes an upper bound on the data processing throughput of the entire system. If $S$ and $R$ are the number of cores and RSs of a loop respectively, the upper bound is $\frac{S}{S+R}$. The minimum of such bounds across all loops is the *system throughput*, and can be determined by efficient algorithms [10], [15]. For example, in Fig. 1(a) loop $A \rightarrow B \rightarrow RS \rightarrow A$ is a *critical* loop because it imposes the lowest bound on the throughput. Since it has 2 cores and 1 RS, the throughput is degraded to $2/(2+1) = 0.67$.

The throughput degradation, however, can be mitigated if the *concurrency* of the critical loop is increased by a finer-grained shell encapsulation of the cores' logic. For example, Fig. 1(b) shows that the logic within core $B$ can be partitioned into two smaller cores $B_1$ and $B_2$, each of which has its own shell. This *partitioning* of core $B$ raises the system throughput by 13% to 3/4 = 0.75. The improvement results from the fact that in Fig. 1(a) the computation of the logic within $B_1$ and $B_2$ will be stalled if $B$'s shell receives a void data, while in Fig. 1(b) $B_1$ and $B_2$ are never stalled at the same time.

Further, instead of increasing concurrency, to judiciously decrease the granularity of shell encapsulation on non-critical loops can reduce the shell's area overhead without hurting the system throughput. For example, since loop $C \rightarrow D \rightarrow C$ is not critical, cores $C$ and $D$ can be *merged* so that they get encapsulated by one shell (instead of two) as shown in Fig. 1(c), where there are two less queues and one less control block than in Fig. 1(b), while the throughput is still $0.75$.

**Contributions.** We propose the idea of combining core partitioning and merging to optimize a latency-insensitive system and an iterative high-level physical synthesis flow that relies on novel partitioning and merging algorithms to automatically do so. The proposed flow targets the two factors affecting the data processing throughput: the locations of the RSs, which is usually decided by the physical floorplanning of the cores, and the system-level topology. Unlike the classical LID, which treats each core as a *blackbox*, the proposed approach assumes that some cores are provided as *whiteboxes* whose RTL implementation can be analyzed by the partitioning and merging algorithms. Notice that while the cores' logic can be partitioned or merged for shell encapsulation purpose, the RTL design is not modified. The partitioning algorithm leverages the *local* logic structure of individual cores to improve the *global* topology of the system by identifying finer-grained logical boundaries for shell encapsulation. The iterative nature of the proposed flow allows such improvement in the system topology to be further exploited by a floorplanner for throughput optimization. Dually, the core merging algorithm decreases the granularity of shell encapsulation so that the number of shells and shell queues is reduced for area saving.

## II. THE PHYSICAL SYNTHESIS FLOW

Fig. 1(d) shows the proposed physical synthesis flow. Multiple floorplanning runs are interleaved with runs of the new partitioning and merging algorithms to meet the design target, which can be system throughput, chip area, or both. It consists of three main steps:

**Step 1.** Shell encapsulation is applied based on an initial partitioning of the design. After floorplanning, RSs are inserted to pipeline long channels that have caused timing violations.

**Step 2.** If the insertion of RSs made throughput lower than the design target, then the partitioning algorithm divides one core on the critical loop into new smaller cores that are guaranteed not to have combinational path from their inputs to outputs (Section IV). New shells are generated for these cores. The finer-grained shell encapsulation results in a more concurrent system-level topology of the latency-insensitive system.

**Step 3.** First a new floorplan is derived from the new system topology, then the throughput-preserving merging algorithm is applied to merge cores which are not part of the critical loops (Section V). Merging decreases the granularity of shell encapsulation, thus reducing their aggregate area. Since it also opens opportunities for new optimization, the entire process is repeated from Step 1 until the design target is met.

While this synthesis flow can use any floorplanner, a throughput-driven floorplanner [5], [28] is most suited to our approach. In particular, for the experiments of Section VI we used the floorplanner proposed by Casu and Macchiarulo [5].

## III. DEFINITIONS

In this section we define important concepts that we used to present the proposed algorithms in the following sections.

A design is modeled at the RTL level as a *netlist*, a directed graph $G = \{V, E\}$ where a node $v \in V$ may represents a combinational gate, a latch, a primary input, or a primary output. An edge $(u, v) \in E$ represents a wire connecting node $u$ to $v$. The *immediate fanout* of $u \in V$ is $\mathrm{FO}(u) \equiv \{v \mid (u, v) \in E\}$ and the *immediate fanin* of $v \in V$ is $\mathrm{FI}(v) \equiv \{u \mid (u, v) \in E\}$. Let $s \overset{\leadsto}{} t$ be a path connecting node $s$ to $t$. The *sequential distance* of a simple path of $G$ is the number of latches on the path. Define $\mu(s \overset{*}{\leadsto} t)$ as the shortest sequential distance from node $s$ to $t$.

The $i$-stage *transitive fanout* of $u \in V$ is $\mathrm{TFO}_i(u) \equiv \{v \mid u \overset{*}{\leadsto} v, \mu(u \overset{*}{\leadsto} v) \leq i\}$. The $i$-stage *transitive fanin* of $v \in V$ is $\mathrm{TFI}_i(v) \equiv \{t \mid t \overset{*}{\leadsto} v, \mu(t \overset{*}{\leadsto} v) \leq i\}$. The intersection $\mathrm{TFI}_1(\ell) \cap \mathrm{TFO}_1(\ell)$ of the 1-stage transitive fanin and fanout nodes of a latch $\ell$ is the set of combinational nodes on all of the feedback paths of $\ell$. The $i$-stage transitive fanout latches of a combinational gate $g$ is $\mathrm{TLO}_i(g) \equiv \{\ell \mid \ell$ is a latch, $g \overset{*}{\leadsto} \ell, \mu(g \overset{*}{\leadsto} \ell) \leq i\}$. Similarly, $g$'s $i$-stage transitive fanin latches is $\mathrm{TLI}_i(g) \equiv \{\ell \mid \ell$ is a latch, $\ell \overset{*}{\leadsto} g, \mu(\ell \overset{*}{\leadsto} g) \leq i\}$.

A *partitioning* $P \equiv \{C_1 = \{V_1, E_1\}, \ldots, C_n = \{V_n, E_n\}\}$ of $G = \{V, E\}$ is a set of non-overlapping subgraphs of $G$ such that $\{V_1, \ldots, V_n\}$ forms a partition of $V$. Subgraph $C_i = \{V_i, E_i\}$ is called a *core*; its edge set $E_i = \{(u, v) \mid (u, v) \in E$ and $u, v \in V_i\}$ are the edges whose source and destination nodes are all in $V_i$. Given a partitioning $P$, the core containing node $u \in V$ is denoted as $core_P[u]$. A *merging* operation on a set of cores merges these cores into a single one.

Given a partitioning $\{C_1, \ldots, C_n\}$ of $G = \{V, E\}$, a node $u$ is called an *input/output boundary* if at least one of $u$'s immediate fanin/fanout is in a different core. All the edges from core $C_i$ to $C_j$ form a *channel* $(C_i, C_j) \equiv \{(s,t) \mid (s,t) \in E, s \in C_i$ and $t \in C_j, C_i \neq C_j\}$. The *channel width* $W(i,j)$ is the number of edges on the channel.

A core is *sequential* if the shortest sequential distance from the core's input boundaries to output boundaries is at least one. That is, the core has no combinational path from its inputs to its outputs. A partitioning of a sequential core may contain non-sequential cores; in contrast, merging a set of sequential cores always gives a sequential one. A partitioning in which all cores are sequential is a *sequential partitioning*.

Sequential cores are required for the application of LID. Because the minimum forward latency of the void signal passing through the shell is one [17], the presence of a combinational path between the core's input and output may result in a mismatch between the validity of the core's output data and the void signal generated by the shell.[1]

## IV. Throughput-Driven Partitioning Algorithm

The goal of the partitioning algorithm is to increase the concurrency of the system topology for throughput optimization. Before presenting the algorithm, we introduce some facts on which it is based.[2] The partitioning algorithm needs to ensure that the resulting partitioning of the core's logic is *sequential*. Let $P = \{C_{i,1}, \ldots, C_{i,n}\}$ be a partitioning of core $C_i$. The following two lemmas provide conditions to maintain the sequential property for all cores in $P$:

**Lemma 1 (Feedback)** *Let* $\ell \in C_i$ *be a latch. If partitioning* $P$ *is sequential, any combinational gate* $g \in (\text{TFI}_1(\ell) \cap \text{TFO}_1(\ell))$ *on the feedback path of* $\ell$ *satisfies* $core_P[g] = core_P[\ell]$.

**Lemma 2 (Transitive Fanin and Fanout)** [3] *Partitioning* $P$ *is sequential if and only if one of the following two conditions is true for any combinational gate* $g \in C_i$:

*(1)* $\forall h \in \text{TFI}_0(g) \cup \text{TLI}_1(g), core_P[g] = core_P[h]$, *or*
*(2)* $\forall h \in \text{TFO}_0(g) \cup \text{TLO}_1(g), core_P[g] = core_P[h]$.

Let $C_j$ be a sequential core on a critical loop with $S$ cores and $R$ RSs. Hence, the system throughput is $\frac{S}{S+R}$ (see Section I). Without loss of generality, let $I$ be the input boundary set of $C_j$ induced by channel $(C_i, C_j)$ and $O$ be the output boundary set induced by channel $(C_j, C_k)$, where all three cores $\{C_i, C_j, C_k\}$ are part of the critical loop. The next lemma states that the system throughput improvement that can be obtained by partitioning $C_j$ is limited by the shortest sequential distance between inputs and outputs of $C_j$.

**Lemma 3 (Dominance of the Shortest Sequential Distance)** *Let* $(D+1)$ *with* $D \geq 0$ *be the shortest sequential distance from any* $s \in I$ *to any* $t \in O$ *of core* $C_j$. *For any sequential partitioning* $P$ *of* $C_j$, *let* $\vartheta_P$ *be the throughput of the system in which* $C_j$ *is replaced by* $P$. *Then* $\vartheta_P \leq \frac{S+D}{S+D+R}$.

---

[1] In fact, an incoming void data could pass through the core's combinational path to change (and thus corrupt) the core's output value at the same clock cycle while the shell's output void signal would not reflect the invalidated output data until the next cycle.

[2] Proofs of Lemmas 1-3 are given in [18] and omitted here for space reasons.

[3] Lemma 1 is actually a special case of Lemma 2.

---

LI-PARTITIONING($G = \{V, E\}, \mathcal{I} = \{I_1, \ldots, I_N\}$)
1  CLUSTER-LATCHES-BY-SEQ-DIST($G, \mathcal{I}$)
2  CLUSTER-LATCHES-WITH-SHARED-FEEDBACK($G$)
3  CLUSTER-BY-MIN-COVERING($G$)
4  CLUSTER-BY-MIN-CUT($G$)
5  COST-RECOVERING($G$)

Fig. 2.  The partitioning algorithm at the top-level.

Fig. 2 shows the partitioning algorithm that takes as input a core $C_j = \{V_j, E_j\}$ and subsets of its input boundaries $\mathcal{I} = \{I_1, \ldots, I_n\}$. Suppose core $C_j$ has $n$ input channels $(C_1, C_j), \ldots, (C_n, C_j)$, $I_i \in \mathcal{I}$ is the set of input boundaries induced by the input channel $(C_i, C_j)$. The algorithm clusters the logic elements of the core into a number of smaller sequential cores and returns the size and the connectivity information of these smaller cores. The procedures are described below:

• Procedure CLUSTER-LATCHES-BY-SEQ-DIST clusters latches based on their shortest sequential distances to the input boundary nodes $\mathcal{I} = \{I_1, \ldots, I_n\}$. For each latch $\ell \in V$, the procedure computes the shortest sequential distance across nodes in each set of the input boundary nodes to $\ell$. The computation associates to each latch $\ell$ an $n$-tuple of shortest sequential distances $\langle \mu_1, \ldots, \mu_n \rangle$, which establish an equivalence relation among all latches. The latches having the same $\langle \mu_1, \ldots, \mu_n \rangle$ are grouped together as a *seed kernel*. These seed kernels will be "grown" into cores containing combinational gates in the following steps.

• Procedure CLUSTER-LATCHES-WITH-SHARED-FEEDBACK clusters combinational logic gates on feedback paths of latches. To satisfy Lemma 1, the procedure merges certain seed kernels if necessary. For each latch $\ell$ the procedure computes the intersection of $\text{TFI}_1(\ell)$ and $\text{TFO}_1(\ell)$, which is the set of the combinational gates on the feedback paths of $\ell$. These combinational gates are grouped into the seed kernel which contains $\ell$. If a combinational gate is on the feedback paths shared by multiple latches, all of the seed kernels of these latches are merged into one seed kernel.

• Procedure CLUSTER-BY-MIN-COVERING merges selected seed kernels of latches so that Lemma 2 is satisfied. For each combinational gate $g$ not yet grouped into any seed kernel, let $P_I(g) = \{C_{I,1}(g), \ldots, C_{I,m}(g)\}$ be the set of the seed kernels of latches in $\text{TLI}_1(g)$, and $P_O(g) = \{C_{O,1}(g), \ldots, C_{O,n}(g)\}$ be the set of the seed kernels of latches in $\text{TLO}_1(g)$. By Lemma 2, one of the two sets of seed kernels, $P_I(g)$ or $P_O(g)$, needs to be merged.

Since the goal of the algorithm is to maximize the number of partitions of $C_j$ between $C_j$'s input and output channels, the selection of merging either $P_I(g)$ or $P_O(g)$ is formulated as a minimum-cost covering problem. The cost is the number of seed kernels to be merged. In the covering matrix, each row represents a combinational gate $g$ that satisfies $|P_I(g)| > 1$ and $|P_O(g)| > 1$, and is "covered" by two columns: one represents $P_I(g)$ with cost $|P_I(g)|$, and the other represents $P_O(g)$ with cost $|P_O(g)|$. The seed kernels that correspond to a selected column in the solution of the covering problem are merged. Then, some combinational gates can be grouped into these merged seed kernels. For combinational gate $g$, if $|P_I(g)| = 1$ but $|P_O(g)| > 1$, $g$ is put into the only core in $P_I(g)$, and vice versa for the case of $|P_I(g)| > 1$ and $|P_O(g)| = 1$. The case of $|P_I(g)| = 1$ and $|P_O(g)| = 1$ is handled by the next step.

• Procedure CLUSTER-BY-MIN-CUT clusters the remaining combinational gates, which are not clustered to any seed kernel yet and satisfy $|P_I(g)| = 1$ and $|P_O(g)| = 1$. We call them *free* gates because by Lemma 2 they can be either put into the core of $P_I(g)$ or $P_O(g)$. In fact, the combinational nodes not yet clustered in $\text{TFI}_0(g)$ and in $\text{TFO}_0(g)$ are also free gates and enjoy the same degree of freedom in clustering. The clustering of the free gates in $\text{TFI}_0(g)$ and $\text{TFO}_0(g)$ decides the final boundary between $P_I(g)$ and $P_O(g)$. It is desirable to minimize the number of edges crossing the two connected cores $P_I(g)$ and $P_O(g)$ (this number is called *cut size*), because these edges eventually become channel signals and their number determines the area of the queues in the receiver's shell. To minimize the cut size CLUSTER-BY-MIN-CUT computes the so called *unidirectional* [7] minimum cut using a maximum flow algorithm. The unidirectional constraint forces all cut edges to be of the same direction and thus avoids combinational paths. The flow network is constructed based on the subgraph induced by the gates and edges that are both in $\big(\text{TFI}_0(g) \cup \text{TLI}_1(g)\big)$ and $\big(\text{TFO}_0(g) \cup \text{TLO}_1(g)\big)$. Each induced edge in the flow network has unit capacity. In addition, to maintain the unidirectional property, for each induced edge a "reverse" edge having infinity capacity running in opposite direction is added. We also add a *source* node and edges of infinite capacity connecting it to all of the latches in $\text{TLI}_1(g)$. Similarly, we add a *sink* node and edges of infinite capacity connecting all of the latches in $\text{TLO}_1(g)$ to it. The min-cut of the flow network is computed by solving the max-flow problem. Nodes in the same cut set are clustered together.

## V. THROUGHPUT-PRESERVING MERGING ALGORITHM

Following the throughput-driven partitioning a new floorplan is derived. Based on the new floorplan it is possible to reduce shell area overhead while maintaining the data processing throughput by merging cores on non-critical loops. The area saving comes from the fact that the merging of two cores connected by a channel allows the queue storing data of the channel in the downlink shell to be removed. The area of a shell is linear to the total width of the input channels of the shell. Thus minimizing the shell area overhead is equivalent to minimizing the total channel width by core merging.

Iteratively the throughput-preserving merging algorithm selects cores eligible for merging. The algorithm is as follows:

1) Select pairs of cores for merging such that the following four conditions are satisfied: (i) a core can only be merged with at most one of its neighbors; (ii) the merging of the cores does not change the data processing throughput; (iii) the channel connecting any selected pair of cores is not pipelined by any RS; and (iv) the total area of each pair of cores cannot exceed the maximum allowable area $\overline{A}$ to avoid long intra-core wires.

2) Merge the selected cores and update the system topology. Repeat Step 1 if at least one pair of cores is selected and merged, otherwise stop.

The selection of the cores that are eligible for merging is done by solving a mixed integer linear program (MILP). The objective is to minimize the total channel width after merging.

**MILP 1 (Core Merging MILP)** *Let* $1/\text{T}$ *be the data processing throughput of a LI system with $n$ cores $\{C_1, \ldots, C_n\}$,*



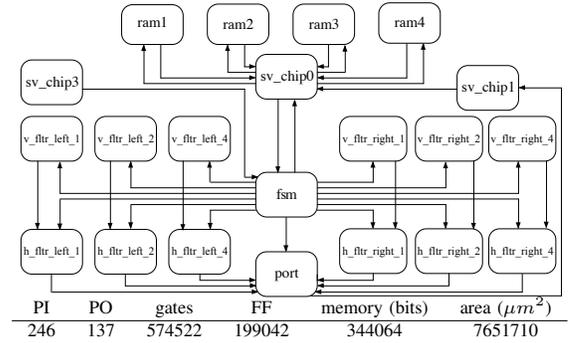| PI | PO | gates | FF | memory (bits) | area ($\mu m^2$) |
|----|----|-------|-----|---------------|------------------|
| 246 | 137 | 574522 | 199042 | 344064 | 7651710 |

Fig. 3. The top-level block diagram of `stereo_vision` based on its functional partition. Primary inputs and outputs are not shown.

$\text{A}(i)$ *be the area of core $C_i$, $\overline{A}$ be the maximally allowable core area, $\text{R}(i, j)$ be the number of relay stations inserted on channel $(C_i, C_j)$, $\text{W}(i, j)$ be the width of channel $(C_i, C_j)$, and M be a large constant s.t. $\text{M} \gg 2 \times \sum_{i,j} \text{R}(i, j)$.*

**Variables:** *$\{\pi_1, \ldots, \pi_n\}$ are variables of real values assigned for each core. For channel $(C_i, C_j)$ connecting core $C_i$ and $C_j$ a binary variable $m_{i,j} \in \{0, 1\}$ is used to encode whether core $C_i$ and $C_j$ are to be merged ($m_{i,j} = 0$ if merged).*

**Objective:** *$\min \sum_{(i,j)} m_{i,j} \text{W}(C_i, C_j)$.*

**Constraints:** *For channel $(C_i, C_j) \in E$, the following constraints are used:*

$$\pi_i - \pi_j + \text{T} \times m_{i,j} \geq (\text{R}(i, j) + 1)m_{i,j} \qquad (1)$$
$$\pi_j - \pi_i + (\text{T} \cdot \text{M}) \times m_{i,j} \geq (\text{R}(i, j) + 1)m_{i,j}, \qquad (2)$$
$$m_{i,j} = 1 \text{ if } \text{R}(i, j) \geq 1 \qquad (3)$$
$$m_{i,j} = 1 \text{ if } \text{A}(i) + \text{A}(j) \geq \overline{A} \qquad (4)$$

Eq. (1)–(2) are used to maintain the throughput $1/\text{T}$. For each channel $(i, j)$, if $(m_{i,j} = 0)$ in the solution of the MILP then core $C_i$ and $C_j$ are merged as a new core $C_k$ whose area is $\text{A}(k) = \text{A}(i) + \text{A}(j)$. For any core $C_\ell$ feeding both core $C_i$ and $C_j$, channels $(C_\ell, C_i)$ and $(C_\ell, C_j)$ are replaced by a new channel $(C_\ell, C_k)$ with $\text{W}(\ell, k) = \text{W}(\ell, i) + \text{W}(\ell, j)$ and $\text{R}(\ell, k) = \max(\text{R}(\ell, i), \text{R}(\ell, j))$. Similar rules apply to the case in which core $C_\ell$ is fed by both $C_i$ and $C_j$.

## VI. EXPERIMENTAL RESULTS

To evaluate the proposed high-level physical synthesis flow we completed a case study with `stereo_vision`, a real SoC design that measures stereo depth [9]. It consists of 16 instanced IP blocks for a total of over half a million gates and about 200,000 flip-flops. The top-level block diagram of this SoC and its characteristics after synthesis and technology mapping are reported in Fig. 3. We synthesized two versions of `stereo_vision` with Synopsys DesignCompiler [25] and a $90nm$ industrial standard cell library[4]: the original ("strict") system implementation and a latency-insensitive one.[5]

**Applicability.** The first experiment analyzes the potential improvements of data processing throughput resulted from our throughput-driven partitioning algorithm. The potential gain of throughput depends on the increase of the number of cores on a critical loop after a core on the loop is partitioned.

---

[4]We re-targeted the design from FPGA to ASIC platforms. We replace all FPGA-specific IP cores used in the design with equivalent ones from Synopsys DesignWare [26]. The SRAMs are generated by a register file generator.

[5]The queue capacity of all shells in the LI implementation is set to two.

| core | minimum length increase | | | | | partitioning run time (sec.) |
| | min | $Q_1$ | median | $Q_3$ | max | |
|---|---|---|---|---|---|---|
| sv_chip0 | 2 | 2 | 4 | $\infty$ | $\infty$ | 17.26 |
| sv_chip1 | 4 | 5 | 5 | $\infty$ | $\infty$ | 52.46 |
| sv_chip3 | 2 | 2 | $\infty$ | $\infty$ | $\infty$ | 0.06 |
| h_fltr ($\times 5$) | 5 | 11 | 11 | 11 | $\infty$ | 2.83 |
| v_fltr_226 ($\times 2$) | 1 | 4 | 232 | $\infty$ | $\infty$ | 10.15 |
| v_fltr_316 ($\times 2$) | 1 | 4 | 322 | $\infty$ | $\infty$ | 17.30 |
| v_fltr_496 ($\times 2$) | 1 | 4 | 502 | $\infty$ | $\infty$ | 41.93 |
| fsm | 1 | 1 | 1 | 3 | $\infty$ | 0.21 |
| port | 3 | 3 | 3 | 3 | $\infty$ | 0.20 |

Fig. 4. The five-number summaries of the length increases after partitioning.

Fig. 4 reports the *five-number summaries* of all potential length increases caused by partitioning the top-level cores of `stereo_vision`, and the run time of the partitioning on Intel Core 2 Duo with 2GB memory. For a core with $n$ input and $m$ output channels, there are potentially $n \times m$ loops passing through the core in the system whose lengths may increase after partitioning. The maximum, the first and third quartiles, the median, and the minimum (the five numbers) of these $n \times m$ length changes are listed in each row. Some of the paths affected by the core's partitioning become *disconnected* (indicated by the $\infty$ symbols). The disconnection implies that if the original path is part of a loop, the loop will disappear after the core's partitioning. Further all but one of the medians of length increase are greater than 1, indicating the chances of throughput improvement are high.

**Comparative Analysis.** In the second experiment we floorplanned three latency-insensitive implementations of `stereo_vision` obtained with our physical synthesis flow. As a reference point, we also floorplanned the strict implementation with PARQUET [1]. The results are summarized in Fig. 5. Each row reports cell area (broken down into core and shell area), channel width, total number and width of relay stations, floorplan area, and data processing throughput. Row *"strict"* shows the results of floorplanning the strict implementation: since this is *not* latency-insensitive, the shell area is zero and throughput is one. The next three rows report the floorplan results after each of the three steps of our synthesis flow (see Section II). The corresponding floorplans are shown in Fig. 6. Row *"starting floorplan"* shows the results of the traditional flow which applies throughput-driven floorplanning [5] to the latency-insensitive implementation based on the original SoC organization, i.e. without using our partitioning and merging algorithms to reorganize the logic across the cores. Row *"post-partitioning"* shows the results after throughput-driven partitioning is applied to core `sv_chip0` to divide it into smaller cores. Row *"post-merging"* gives the floorplanning results after the throughput-preserving merging is performed.

All floorplans are required to fit into a fixed outline of unit aspect ratio with $15\%$ white space. The best in terms of throughput of 50 different floorplanning tries is used. The minimum half-perimeter of the largest core is set as the *critical length*: channels longer than this length need pipelining.[6] The area of the largest core is used as the maximum allowable area in the merging algorithm. For each core the allowable area is set $10\%$ larger than the core's aggregate cell area, anticipating the extra room required in the later placement stage.

With respect to the traditional flow, which returns a LI implementation with a throughput of $0.83$ and an area overhead

of $5\%$ compared to the strict implementation, our flow not only improves the system throughput but also the floorplan area. After applying the partitioning algorithm and re-floorplanning the design, the throughput is improved to $1.0$, the ideal value. Although the area overhead is increased after partitioning due to shell encapsulation ($12\%$ in cell and $16\%$ in floorplan area, compared to the strict system), it is fully recouped by the throughput-preserving merging algorithm:[7] merging not only retains the ideal throughput, but also reduces the area overhead to $3\%$. In summary, after one iteration of our synthesis flow, the throughput of the LI implementation of `stereo_vision` is improved by $16.7\%$ while the floorplan area overhead, compared to the strict system, is reduced from $5\%$ to $3\%$.

## VII. RELATED WORK

Partitioning and floorplanning are important physical synthesis techniques. However many of the earlier works precede LID and do not consider its optimization goals and constraints. Classic partitioning algorithms aim for the minimization of the number of nets crossing different partitions, called "cut". These algorithms usually do not avoid combinational paths between inputs and outputs of the resulting partitions, and thus are not suitable for LID. Cut-minimization partitioning was recently combined with floorplanning and placement [6], [24]. On the other hand, several partitioning algorithms avoid the combinational paths in the resulting partitions [7], [13], [20], [29], but none of them considers system topology for throughput optimization of LID. As to floorplanning, classic optimization goals include area and total wire length, while more recent works optimize the instruction execution rate of microprocessors in the presence of multi-cycle communication latencies between computation units [11], [21], [23].

Lin et al. in [19] identify the upper bound of the "data processing rate" or equivalently the effective clock frequency of a latency-insensitive system as the minimum ratio of the number of latches to the total delay across all loops in the system. In [19] a clustering algorithm is proposed to optimize this *upper bound*. Their algorithm allows gate duplications and assigns a constant delay to each of the inter-core wires. In contrast, our approach does not change the RTL implementation of the original design, and thus is more suitable to large SoCs. Further our integrated flow estimates inter-core wire delays from the system-level floorplan, which is more accurate than the constant-delay assumption in [19].

Two floorplanning approaches specifically designed for performance optimization of latency-insensitive systems have been recently proposed. In [5] a fast algorithm approximating the data processing throughput is used as part of the cost calculation of PARQUET [1]. In [28], instead, the floorplanner computes the exact value of the aforementioned upper-bound as part of its main cost function. Differently from these works on LID floorplanning, we perform combined optimizations of floorplan with RTL shell encapsulation by analyzing the logical structures and physical locations of the cores.

## VIII. CONCLUSION

The proposed physical synthesis flow combines a throughput-driven partitioning and a throughput-preserving

---

[6]This length is about $67\%$ of the width of the chip, or $2255\mu m$.

[7]The merging algorithm finishes in 5 seconds.

| implementation | cell area ($\mu m^2$) | | | | channel width | RS | RS width | floorplan area | | throughput |
|---|---|---|---|---|---|---|---|---|---|---|
| | core | shell | total | overhead | (bits) | (number) | (bits) | ($\mu m^2$) | overhead | |
| strict | 7651710 | 0 | 7651710 | 1.00 | – | – | – | 8741760 | 1.00 | 1.00 |
| a) LI: starting floorplan | 7651710 | 225871 | 7877581 | 1.03 | 3124 | 9 | 378 | 9200440 | 1.05 | 0.83 |
| b) LI: post-partitioning | 7651710 | 939332 | 8591042 | 1.12 | 12685 | 19 | 1075 | 10151900 | 1.16 | 1.00 |
| c) LI: post-merging | 7651710 | 190812 | 7842522 | 1.03 | 2625 | 0 | 0 | 9005140 | 1.03 | 1.00 |

Fig. 5.   Results of floorplanning `stereo_vision`: strict version and LI versions (after each step of the proposed physical synthesis flow.)
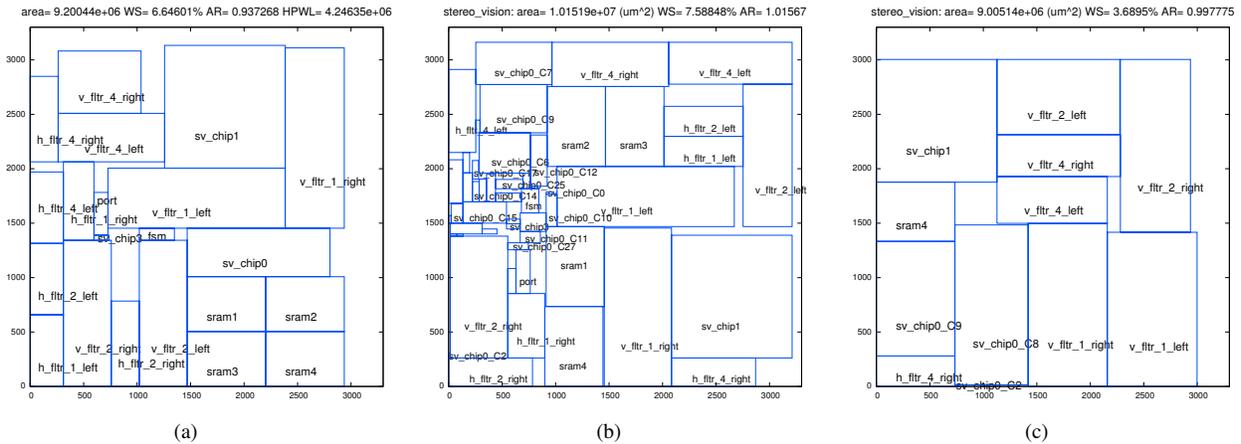


Fig. 6.   Fixed-outline floorplans of `stereo_vision` after each of the steps of the proposed flow: (a) starting floorplan; (b) post-partitioning; (c) post-merging.

merging algorithm with a published floorplanner to optimize the global floorplan of latency-insensitive implementation of a multi-core SoC by analyzing and leveraging the local, intra-core logic structure of its individual cores. Experimental results on a large SoC design shows a $16.7\%$-speedup in data processing throughput and a $2.1\%$-reduction in area occupation, confirming the effectiveness of the proposed approach.

## ACKNOWLEDGMENTS

## REFERENCES

[1] S. N. Adya and I. L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. on Very Large Scale Integr. Syst.*, 11(6):1120–1135, Dec. 2003.

[2] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 20(9):1059–1076, Sept. 2001.

[3] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency insensitive systems. In *Proc. of the Design Automation Conf. (DAC)*, pages 361–367, June 2000.

[4] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Coping with latency in SOC design. *IEEE Micro*, 22(5):24–35, Sep-Oct 2002.

[5] M. R. Casu and L. Macchiarulo. Throughput-driven floorplanning with wire pipelining. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 24-6:663–675, May 2005.

[6] T.-C. Chen, Y.-W. Chang, and S.-C. Lin. A new multilevel framework for large-scale interconnect-driven floorplanning. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 27:286–294, Feb. 2008.

[7] J. Cong, Z. Li, and R. Bagrodia. Acyclic multi-way partitioning of boolean networks. In *Proc. of the Design Automation Conf. (DAC)*, pages 670–675, 1994.

[8] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. of the Design Automation Conf. (DAC)*, pages 657–662, 2006.

[9] A. Darabiha, J. Rose, and W. J. MacLean. Video-rate stereo depth measurement on programmable hardware. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 203–210, June 2003.

[10] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 17:889–899, Oct. 1998.

[11] M. Ekpanyapong, J. R. Minz, T. Watewai, H.-H. S. Lee, and S. K. Lim. Profile-guided microarchitectural floorplanning for deep submicron processor design. In *Proc. of the Design Automation Conf. (DAC)*, pages 634–639, 2004.

[12] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *IEEE Proc.*, 89(4):490–504, Apr. 2001.

[13] A. B. Kahng and X. Xu. Local unidirectional bias for smooth cutsize-delay tradeoff in performance-driven bipartitioning. In *Proc. of the Intl. Symposium on Physical Design*, pages 81–86, 2003.

[14] T. Kam, M. Kishinevsky, J. Cortadella, and M. Galceran-Oms. Correct-by-construction microarchitectural pipelining. In *Proc. of the Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 434–441, 2008.

[15] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[16] K. Keutzer, S. Malik, A. R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 19(12):1523–1543, December 2000.

[17] C.-H. Li, R. L. Collins, S. Sonalkar, and L. P. Carloni. Design, implementation, and validation of a new class of interface circuits for latency-insensitive design. In *Intl. Conf. on Formal Methods and Models for Codesign*, pages 13–22, 2007.

[18] C.-H. Li, S. Sonalkar, and L. P. Caroni. Exploiting local logic structures to optimize multi-core soc floorplanning. Technical Report cucs-052-09, Department of Computer Science, Columbia University, 2009.

[19] C. Lin, J. Wang, and H. Zhou. Clustering for processing rate optimization. *IEEE Trans. on Very Large Scale Integr. Syst.*, 14(11):1264–1275, Nov. 2006.

[20] H. Liu and D. F. Wong. Network flow based circuit partitioning for time-multiplexed FPGAs. In *Proc. of the Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 497–504, 1998.

[21] C. Long, L. J. Simonson, W. Liao, and L. He. Microarchitecture configurations and floorplanning co-optimization. *IEEE Trans. on Very Large Scale Integr. Syst.*, 15(7):830–841, 2007.

[22] R. Lu and C.-K. Koh. Performance analysis of latency-insensitive systems. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 25(3):469–483, Mar. 2006.

[23] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar. Microarchitecture-aware floorplanning using a statistical design of experiments approach. In *Proc. of the Design Automation Conf. (DAC)*, pages 579–584, 2005.

[24] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut floorplacement. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 25(7):1313–1326, July 2006.

[25] Synopsys, Inc. *Design Compiler User Guide*.

[26] Synopsys, Inc. *DesignWare User Guide*.

[27] M. Vijayaraghavan and Arvind. Bounded dataflow networks and latency-insensitive circuits. In *Intl. Conf. on Formal Methods and Models for Codesign*, July 2009.

[28] J. Wang, P.-C. Wu, and H. Zhou. Processing rate optimization by sequential system floorplanning. In *Proc. Intl. Symposium on Quality Electronic Design*, pages 340–345, Mar. 2006.

[29] G.-M. Wu, J.-M. Lin, and Y.-W. Chang. Generic ILP-based approaches for time-multiplexed FPGA partitioning. *IEEE Tran. on Computer-Aided Design of Integr. Circuits and Syst.*, 20(10):1266–1274, 2001.