Floorplanning with wire pipelining in adaptive communication channels

[6] P. Chen, D. A. Kirkpatrick, and K. Keutzer, *Static Crosstalk-Noise Analysis for Deep Sub-Micron Digital Designs*. New York: Springer-Verlag, 2004.
[7] A. Deutsch *et al.*, "On-chip wiring design challenges for gigahertz operation," *Proc. IEEE*, vol. 89, no. 4, pp. 529–555, Apr. 2001.
[8] H. Hidaka *et al.*, "Twisted bit-line architectures for multi-megabit DRAMs," *IEEE J. Solid-State Circuits*, vol. 24, no. 1, pp. 21–27, Feb. 1989.
[9] K. Itoh, *VLSI Memory Chip Design*. Berlin, Germany: Springer-Verlag, 2001.
[10] Y. Konishi *et al.*, "Analysis of coupling noise between adjacent bit lines in megabit DRAMs," *IEEE J. Solid-State Circuits*, vol. 24, no. 1, pp. 35–42, Feb. 1989.
[11] D.-S. Min and D. W. Langer, "Multiple twisted dataline techniques for multigigabit DRAMs," *IEEE J. Solid-State Circuits*, vol. 34, no. 6, pp. 856–865, Jun. 1999.
[12] M. Redeker, B. F. Cockburn, and D. G. Elliott, "An investigation into crosstalk noise in DRAM structures," in *Proc. IEEE Int. Workshop Memory Technol., Des. and Testing*, 2002, pp. 123–129.
[13] I. Schanstra and A. J. van de Goor, "Consequences of RAM bitline twisting for test coverage," in *Proc. Design, Autom. and Test Eur.*, 2003, pp. 1176–1177.

Floorplanning With Wire Pipelining in Adaptive Communication Channels

Mario R. Casu and Luca Macchiarulo

***Abstract*—The recent shift toward wire pipelining (WP) mandated by technological factors has attracted attention toward latency-controlled floorplanning. However, no systematic study has been published so far that takes into account block and logic-delay limitations. This paper aims at filling the gap by showing that block delay can limit and possibly prevent any real gain WP might promise. In this paper, the authors also show how a modified adaptive WP scheme, on the other hand, allows relevant gains. They built a SoC floorplanner based on the use of adaptive and nonadaptive WP, which optimizes the data rate, taking block delay into account. The results of new and old WP techniques applied on benchmarks and on an MPEG decoder are compared to the optimal results obtained when no WP is employed.**

***Index Terms*—Floorplanning, systems-on-chip, wire pipelining (WP).**

I. Introduction

The different scalability of logic and local interconnects on the one hand, and global wires on the other hand, is a serious concern that might jeopardize the advantage of CMOS technology scaling [1]. Wire pipelining (WP) helps facing this problem. In addition to buffer insertion, wires may be segmented by intermediate flip-flops or latches. The throughput of the connection increases as well as the latency of the connections, thus possibly jeopardizing the frequency increase.

This new technique has attracted attention toward latency-controlled floorplanning (see [2]–[6], or the latest [7], for example). However, to the authors' knowledge, no systematic study has been published so far that takes into account block and logic-delay limitations. In this paper, based on the work we presented at the International Symposium on Physical Design [8], we aim at partially filling the gap. We show that block delay can limit and possibly prevent any real performance gain the WP technique, if implemented in a too conservative way, might promise. We also show how a different WP technique exploiting the locality of computation of some blocks, which from time to time do not read data from inputs with added wire latency, can be the key to fulfill the promise of high data rate of WP. To this aim, we adopted a floorplan strategy that optimizes the data rate of systems (DR) based on standard and modified WP using suitable cost functions. We took into account various models of block delay and compared them to the optimal results obtained when no WP is employed.

In Section II, we briefly recall the characteristics of a standard WP technique. Its limits are highlighted in Section III through a mathematical derivation that takes into account the effect of intellectual property (IP) block delays. In Section IV, the performance results obtained with our new floorplanning strategy employing standard WP, with and without limitations imposed by the blocks delays, are reported. The foundations of the alternative WP methodology are described in Section V, while the results of its application using a modified floorplanner are outlined in Section VI. The effectiveness of the new technique is highlighted by means of a realistic case study, an MPEG decoder. Finally, conclusions are drawn in Section VII.

II. Promise of WP

The increased bandwidth of wires guaranteed by WP comes at the cost of increased wire latency. Such change of paradigm not only opens way to a number of new opportunities (see, for example, [9] and [10]) but also gives rise to relevant issues that range from functional and architectural to physical-design aspects (see for a review [3], [11], and references therein).

Even if from an abstract point of view, it might be possible to run wires at any desired frequency by simply increasing the number of pipeline stages, we need to investigate when this is in fact beneficial to the overall performance of the system. We restrict our analysis to the case (as that of systems-on-chip) in which designers connect IP taken from libraries and are not allowed to modify their internal structure, and no architectural features are present to render their IP independently "latency-tolerant." A class of communication protocols that allow transparent insertion of arbitrary latency is the so-called Latency Insensitive Protocols (LIP), which, by means of special wrappers around the blocks and pipeline elements along the wires, make a system with added latency functionally equivalent to a zero-latency system [9]. The equivalence is obtained by making the blocks "patient," i.e., suspending their operation by suitably gating their clock—if at least one of their inputs is not ready to be processed because of the latency.

Latency insensitiveness guarantees a strong form of equivalence between the original and the pipelined system: all the execution traces differ in "non-valid" time slots where no computation is allowed. This means that a one-to-one mapping between the traces is enforced. This is different from usual functional-level pipelining where there is no concept of nonvalid slots, and two traces might depend substantially on the pipelining depth.

To introduce performance measures in such cases, we first give some definitions.

1) A block is a computational logic element. In our analysis, a block behaves as a synchronous element.
2) A system is a collection of interconnected blocks. Formally, it can be represented with a directed graph $G = \{V, E\}$ whose vertices V are the blocks, and connections are edges E. The system is synchronous to a clock frequency F.

Manuscript received June 26, 2005; revised September 17, 2005 and December 18, 2005. This paper was recommended by Associate Editor P. H. Madden.

M. R. Casu is with the Dipartimento di Elletronica, Politecnico di Torino, I-10129 Turin, Italy (e-mail: mario.casu@polito.it).

L. Macchiarulo is with the Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822 USA.

Digital Object Identifier 10.1109/TCAD.2006.882590

3) The throughput of a block is its average number of computations per clock cycle.
4) The throughput of a system (TH) is the worst throughput of its blocks.
5) The DR is the product of its throughput TH and frequency F, that is, the average number of computations per unit time.

According to this set of definitions, the throughput of a non-WP system is TH $= 1$ because every block executes exactly one valid computation per clock cycle.[1] DR and F have the same meaning for these non-WP systems due to the unitary value of TH. On the contrary, since WP systems may be "patient" and suspended from operating for some clock cycles, their actual throughput is in general < 1. Therefore, the correct metric for the performance evaluation is the product of TH and F, i.e., the DR. In the WP case, differently from the non-WP one, DR and F have not the same meaning because of the possible throughput reduction.

Now, it is clear from the previous discussion that if a system with WP allows a frequency increase of $N\times$, in order to have a significant DR improvement, and so to make WP meaningful, TH should not be degraded more than $1/N\times$. It can be shown that the throughput limitation of systems using this transparent WP technique arises from the presence of loops in the system graph [12]. A loop of m blocks with a total of n pipeline stages along the loop has the following throughput:

$$\text{TH} = \frac{m}{m+n} = \frac{m}{m+\sum_{k=1}^{m} n_k} \tag{1}$$

where n_k is the number of delays added to the kth branch of the worst loop. It can also be shown that the system throughput is bounded by the worst loop, which is the netlist loop having the minimum ratio $m/(m+n)$ (for alternative proofs of these properties, see [3] and [12]). n depends mostly on the physical design, i.e., on how short we have been able to keep the long wires by an intelligent floorplan. It is thus possible to insert this cost in a throughput-driven floorplanning and obtain better TH figures than using wirelength, area, or a combination of the two cost functions [3].

III. Mathematical Framework

Each of the system graph edges $e_{ij} \in E$ is annotated with the worst delay T_{ij} of the connections between block i and block j.[2] T_{ij} includes two contributions.

1) The interconnect delay l_{ij} refers to the global wire between the two blocks and is related to the wire length measured as the Manhattan distance between two pins.
2) The internal delay d_{ij} includes (combinational) logic and local wire delay of the I/O stages of blocks i and j.

It is well known that if a line of length l is optimally buffered with standard repeaters, its delay becomes proportional to l [13]. We thus normalize to one (here and in the rest of this paper) this proportionality coefficient so that "delay" and "length" become synonymous. The name we gave to the interconnect delay l_{ij} derives from this assumption.

Besides edge delays, each vertex, i.e., each block, is characterized by an internal critical path T_i that does not depend on its external connections. In a non-WP system, the largest delay among T_i and $T_{ij}, \forall(i,j)$, is the system critical path T_{scp} that dictates the system frequency $F = 1/T_{\text{scp}}$.

It is obvious that WP can be effectively applied to a non-WP system only if the system frequency is limited by a slow wire and not by an internal critical path. In formulas, $\exists(i,j) : T_{ij} = T_{\text{scp}}$. Otherwise, no matter how small are made T_{ij}, the system clock will be limited by a block's critical path T_i.

Suppose then to start with a non-WP system, which does not meet a frequency constraint, $F_{\min} = 1/T_{\max}$ because of some global interconnect limitation; in formulas, $\exists(i,j) : T_{ij} = T_{\text{scp}} > T_{\max}$. Suppose also that $l_{\max}$ is the maximum distance allowed between two pipeline elements, e.g., two flip-flops, so as to respect the frequency constraint. Since delays and lengths are synonymous in our derivation, it turns out that $l_{\max} = 1/F_{\min} = T_{\max}$. For now, suppose that the internal delays of each connection are negligible, that is, $d_{ij} \simeq 0, \forall(i,j)$, and so $T_{ij} \simeq l_{ij}$. Accordingly, there is at least one connection longer than $l_{\max}$. Formally, $\exists(i,j) : l_{ij} > l_{\max}$.

Suppose now to insert pipeline stages in each connection of the system at stake. Each inserted pipeline element increments the connection latency of one sequential delay. The number of delays n_{ij} in a single edge of the graph will be given by

$$n_{ij} = \lfloor l_{ij}/l_{\max} \rfloor. \tag{2}$$

As stated in Section II, the system throughput TH depends on the total number of delay elements in loops. With reference to our graph model, a loop is a set of m consecutive edges that start and end in the same vertex crossing m nodes. If we enumerate the traversed edges and related length $l_k, k \in [1,m]$, each edge will contain n_k delays calculated according to (2). The throughput of the loop will be given by (1) and the system throughput by the worst loop.

Given that the WP system can be clocked at $F_{\min} = 1/l_{\max}$, the DR will be given by

$$\text{DR} = \frac{m}{m+\sum_{k=1}^{m} \left\lfloor \frac{l_k}{l_{\max}} \right\rfloor} \cdot \frac{1}{l_{\max}}. \tag{3}$$

By using the basic property of the floor function $x - 1 \le \lfloor x \rfloor \le x$, we can easily derive a DR upper bound

$$\text{DR}_{\text{UB}} = \frac{m}{m+\sum_{k=1}^{m}\left(\frac{l_k}{l_{\max}} - 1\right)} \cdot \frac{1}{l_{\max}} = \frac{1}{\frac{1}{m}\sum_{k=1}^{m} l_k} = \frac{1}{l_m} \tag{4}$$

where l_m is the average branch length of the loop, and a lower bound

$$\text{DR}_{\text{LB}} = \frac{m}{m+\sum_{k=1}^{m} \frac{l_k}{l_{\max}}} \cdot \frac{1}{l_{\max}} = \frac{1}{l_{\max}+l_m}. \tag{5}$$

The DR without WP DR_0 is simply given by

$$\text{DR}_0 = \text{TH}_0 \cdot F_0 = 1.0 \cdot \frac{1}{L} \tag{6}$$

where $\text{TH}_0 = 1$ is the unitary throughput for a system without WP, and F_0 is the maximum allowed frequency. As long as the delays d_{ij} are negligible, F_0 is limited by L, which is the longest interconnect delay in the system, and $L = \max_{i,j}(l_{ij})$. Using WP for this kind of systems is convenient only if the speedup, i.e., the ratio DR/DR_0, is significantly higher than one, in order to compensate for the increasing problems of high-frequency clocks and related power dissipation and

[1] We define valid computations also as those cycles where a gated clock is possibly used, considering them as "no-operation (NOP)."

[2] To be general, T_{ij} should also include delays between the system I/O terminals and blocks. If we consider I/O pins homologous to blocks, the definition also holds true for these cases.

timing issues. Using (4)–(6), we obtain that the speed-up SU lays between two bounds

$$\frac{L}{l_m + l_{\max}} \leq \text{SU} \leq \frac{L}{l_m}. \tag{7}$$

Interestingly, increasing the frequency in the WP system, and so augmenting the wire-pipe depth, reduces $l_{\max}$ but cannot further increase the speedup over L/l_m, which merely depends upon the floorplan strategy used.

If the floorplanner is able to keep close the blocks belonging to the loops, the average length l_m would be reduced, and the effectiveness of using WP is maximized. Only the cases in which l_m is substantially lower than L are worth, considering the overheads arising from the increased clock frequency.

A. Effect of Block Delay

So far, we supposed the blocks ideal and attributed all frequency limits to the interconnects. Let us admit now that each edge in the system graph is characterized by a nonnegligible delay d_{ij} and infer the corresponding system DR, with and without WP.

For a zero-latency system, the maximum DR becomes

$$\text{DR}_0 = 1.0 \cdot F_{\max} = \frac{1}{\max_{i,j}(l_{ij} + d_{ij})} \tag{8}$$

where, in general, $\max_{i,j}(l_{ij} + d_{ij})$ is different from $L + d_{\max}$, where L was introduced before, and $d_{\max}$ is the maximum delay, $d_{\max} = \max_{i,j}(d_{ij})$.

For this case, too, the use of WP makes sense only if the frequency is limited by global interconnects. We are then supposed to set a constraint $F_{\min}$, and that the lower bound is due to interconnect limits, viz. $F_{\min} = 1/l_{\max}$. The throughput is

$$\text{TH} = \frac{m}{m + \sum_{k=1}^{m} \left\lfloor \frac{l_k + d_k}{l_{\max}} \right\rfloor} \tag{9}$$

and, after a number of easy algebraic steps, it turns out that DR is bounded as follows:

$$\frac{1}{l_m + d_m + l_{\max}} \leq \text{DR} \leq \frac{1}{l_m + d_m} \tag{10}$$

where d_m is the average delay of various edges (i.e., of the d_{ij} type) belonging to the worst loop

$$d_m = \frac{1}{m} \sum_{k=1}^{m} d_k. \tag{11}$$

Consequently, upper and lower bounds to speed up SU are

$$\frac{\max_{i,j}(l_{ij} + d_{ij})}{l_m + d_m + l_{\max}} \leq \text{SU} \leq \frac{\max_{i,j}(l_{ij} + d_{ij})}{l_m + d_m}. \tag{12}$$

However, (12) holds true as long as $l_{\max} > d_{\max}$. The deeper the WP, the faster is the interconnect, but due to the slowest block, the frequency limit is $1/d_{\max}$. Therefore, even though we reduce the WP flip-flop distance to $l_{\max} = 0$, the maximum speedup would be bounded by

$$\frac{\max_{i,j}(l_{ij} + d_{ij})}{l_m + d_m + d_{\max}} \leq \text{SU} \leq \frac{\max_{i,j}(l_{ij} + d_{ij})}{l_m + d_m}. \tag{13}$$

In the general case, whether the WP system with delays is worse than the previous ideal case or not depends on the d_{ij} values. However, it is likely that the real case is worse. Suppose for instance the ideal speedup is $\text{SU} = L/l_m = 1.5$ according to (7). Suppose also, for the sake of simplicity, that delays and average length l_m are all equal to $d_{\max}$. By evaluating (13), we get $0.833 \leq \text{SU} \leq 1.25$, which, in the worst case (0.833), loses 44% compared to the ideal case (1.5) and 25% in the best case (1.25).

It is clear from the previous discussion that keeping d_{ij} as small as possible is mandatory. Designers are used to break possible critical paths crossing the boundary between two blocks by preplacing boundary registers at the blocks I/O ports. This practice already limits the logic gate delays in d_{ij}. On the other hand, the blocks' internal interconnects still may play an important role. In Section IV, we present some DR results obtained using a throughput-driven floorplanner [3], with and without considering the blocks' delays.

TABLE I
GSRC AND MCNC BENCHMARKS. IDEAL CASE WITHOUT BLOCK'S DELAY

bench.	#blocks	DR	DR_0	$L(\%)$	$SU(\%)$	$l_m(\%)$
n10	10	0.961	0.852	117	+13	104
n30	30	0.979	0.727	138	+35	102
n50	50	0.793	0.617	162	+29	126
n100	100	1.114	0.555	180	+100	90
apte	9	0.705	0.699	143	+1	142
xerox	10	0.613	0.565	177	+9	163
hp	11	0.660	0.511	196	+29	151
ami33	33	1.106	1.039	96	+6	90
ami49	49	1.047	0.774	129	+35	96

IV. TH-DRIVEN FLOORPLAN

We now report some results obtained running the floorplanner [3] over some of the Gigascale Systems Research Center (GSRC) and Microelectronics Center of North Carolina (MCNC) benchmarks [14]. Our floorplanner is derived from the publicly available PARQUET. We refer the reader to [16] and [17] for further details on the floorplanner and to [3] for the modifications to the internal cost functions required for taking WP into account (but see the brief comparison with the new floorplanner described in Section V).

For the nonpipelined systems, we ran floorplanning experiments until we found the shortest possible $l_{\max}$ compatibly with $n = 0$ and so $\text{TH} = 1$ in (1) that is a nonpipelined optimum. For the WP system, we repeated the experiments allowing pipelining and identifying the maximum DR for higher frequency values. Table I summarizes the results of throughput and DR for both systems, with WP (DR) and without (DR_0), obtained in the ideal case of **no block delay**. With respect to our previous work [8], these results have been improved due to a better search of the maximum DR.

Lengths are in percentage of the die edge, computed as the square root of the sum of the blocks' area. We normalized the die edge to 17 mm, according to the ITRS'03 estimations for the high-performance dies [1]. As a result, **all benchmarks have the same total area**, regardless of the blocks' number and size.

Based on L and speed-up SU columns, we can compute the average length l_m of the worst loop branch for the LIP case, according to (7). For the benchmarks considered, the values, in percentage of the die edge, are also reported in Table I. They can be suitably compared to the maximum length L of the case without WP for a better understanding of the speed-up figure.

It is interesting to notice that if we do not set any constraint on the minimum delay in the pin-to-pin connections, i.e., $d_{ij} = 0$, as we did to obtain the results reported in Table I, the maximum DR found by the successive runs of floorplanner is for $l_{\max} \to 0$ ($F_{\min} \to \infty$), which is in strict accordance with the upper bound of (7). According to (2), $n_{ij} \to \infty$ as $l_{\max} \to 0$, and consequently $\text{TH} \to 0$ as of (1). In spite

TABLE II
BLOCK DELAY PROPORTIONAL TO BLOCK'S EDGE

bench	DR	TH	l_{max}	DR_0	L	SU
n10	0.764	0.333	43.6%	0.687	145%	+11%
n30	0.715	0.206	28.8%	0.516	193%	+39%
n50	0.717	0.223	31%	0.626	160%	+15%
n100	0.926	0.167	18%	0.534	187%	+73%
apte	0.541	0.2	37%	0.450	222%	+20%
hp	0.530	0.25	47%	0.530	189%	0%
xerox	0.469	0.25	53.3%	0.469	213%	0%
ami33	0.846	0.333	39.4%	0.846	118%	0%
ami49	0.676	0.333	49.1%	0.613	163%	+10%

TABLE III
BLOCK DELAY EQUAL TO 13FO4

bench	DR	TH	l_{max}	DR_0	L	SU
n10	0.521	0.5	96%	0.521	192%	0%
n30	0.473	0.308	65%	0.408	245%	+16%
n50	0.513	0.333	65%	0.469	213%	+9%
n100	0.5	0.4	80%	0.423	236%	+18%
apte	0.465	0.333	71.9%	0.465	215%	0%
hp	0.478	0.333	69.9%	0.478	209%	0%
xerox	0.397	0.333	84.0%	0.397	252%	0%
ami33	0.559	0.5	89.5%	0.531	188%	+5%
ami49	0.535	0.5	93.4%	0.535	187%	0%

of the throughput degradation, the DR increases, like the SU column in Table I proves. This is an interesting result because it shows that the floorplanner is able to keep short the average interconnect l_m of blocks in loops. As a result, DR increases because the throughput degradation is more than compensated by the clock-frequency increase.

The previous case was that of a system wherein all the blocks have registered inputs and outputs that, although not uncommon, cannot represent the entire spectrum of applications (especially for high-performance systems). We then considered two extreme approaches to take block delays into account. We considered two types of delays related to internal wires or logic. In the first case, we supposed the delay barely proportional to the block edge calculated as the square root of the rectangular block area (i.e., the geometrical average of the two edges). We assumed the proportionality factor equal to 1.0 to represent a mild dependence on the block size. The second choice consisted in choosing a "constant" delay, i.e., independent of the block size. The rationale is that of attributing all the delay to logic stages whose depth is independent of the block size, and it only relies upon the designer's choices. We chose a delay equal to 13FO4, that is, 13 times the delay of an inverter loaded with four identical inverters (fan out of four). This is the value used in the ITRS for computing the estimated clock frequency of high-performance microprocessors [1]. We took the values of logic and interconnect delay from the roadmap in order to define a proportionality factor between the two. Again, we supposed that the wire delays increase linearly with wire length, i.e., the case of buffered wires. It turned out that given the ITRS predicted die area and die edge, and defined 100% the unitary delay of a global buffered wire spanning a die edge, a single FO4 delay equals 5% of that wire delay. Therefore, 13FO4 corresponds to $l_{\max} = 13 \cdot 5 = 65\%$ (buffered-wire delay). In other words, 13FO4-stages logic delay equals the propagation delay of a buffered line of length 65% of a die edge.

Tables II and III report the results for these two choices. From Table II, we observe that the delay proportional to the block's edge leads to results that are not univocal for all benchmarks. While the GSRC suite (n10, n30, n50, and n100) shows a similar if worse behavior than the ideal case, some MCNC benchmarks do not experience any improvement. From Table III, we observe that the impact of a constant block delay is such that for almost all benchmarks, there is no more evidence of the effectiveness of using wire pipes.

We observe a trend toward a generalized reduction of the speedup when the delay becomes relevant, even if there are exceptions for given values of FO4 or block/edge proportion. From this set of experiments, we can draw the conclusion that increasing the DR for this type of transparent WP systems, with respect to a generalized slowdown solution without added wire latency, is certainly not an easy task. If the block's delay is negligible compared with the maximum delay between pipeline elements $l_{\max}$, the speedup is, in general, > 1, but the amount of actual DR increase depends on the circuit geometric and netlist characteristics. For higher delay values, obtaining a significant speedup may be a hopeless task.

So far, we have analyzed a class of systems that use WP in a totally transparent way. Blocks are encapsulated within wrappers that make them patient when input signals are not ready yet due to the wire latency. This is the very reason of the throughput reduction expressed by (1) and in the end of the bad speed-up results reported above for the blocks' delay case. In the next section, we introduce a new class of WP systems where a modification to the protocol aims at breaking this TH limitation. The degradation shown in Tables II and III, of course, will not prevent the system from being employed in cases where all blocks have registered inputs and outputs (as previously mentioned). The advantages of a modification of the protocol will be even greater in such cases.

V. DYNAMICALLY ADAPTIVE COMMUNICATIONS

The underlying assumption of our previous derivation and of the results obtained after floorplanning was that every block reads all its inputs in every clock cycle. However, it is likely that the computation within the blocks has some degree of locality because there certainly exist states where the blocks do not read one or all of their inputs and execute some local computation. This is certainly peculiar to blocks of a relevant complexity. As an example, a microprocessor with a local L1 cache accesses an L2 memory only in case of a miss. In other words, the content of some input connections is ignored from time to time. This property can be exploited in order to break the throughput limitation of (1), as noticed in [15].

It can be shown that an implementation of an extended version of the latency-insensitive protocol is possible that allows this form of adaptiveness. Although still falling short of the flexibility of architectured pipelines as those found in a microprocessor, such protocol has the advantage of maintaining the same computational traces and, therefore, the communication profile as the unpipelined system that can be evaluated up front and conveniently used in the optimization process.

In order to update (1) to fit this new situation, we first need some definitions. Suppose that a given task of the blocks belonging to a loop requires N computations, of which αN is done with "closed" loop and $(1-\alpha)N$ with "open" loop $(\alpha \leq 1)$. Each computation takes one clock cycle when the loop is open and 1/TH clock cycles when closed. The number of clock cycles required to finish the computation is $M = (1-\alpha)N + \alpha N/\mathrm{TH}$. The effective throughput, that is, the average number of computations per clock cycle, will be given by $\mathrm{TH}_e = N/M$ that turns out to be

$$\mathrm{TH}_e = \frac{N}{M} = \frac{1}{1-\alpha+\frac{\alpha}{\mathrm{TH}}}. \tag{14}$$

When $\alpha = 1$, we reobtain $\mathrm{TH}_e = \mathrm{TH}$ as of (1). When instead $\alpha < 1$, we get $\mathrm{TH}_e > \mathrm{TH}$. This might mean that, for a period of a few system's clock cycles, a potentially limiting loop is actually open, so that the

overall performance of the system is related to the most critical **active** loop rather than the most critical loop altogether. This increases the potentiality for higher overall performance (that, as underlined in the previous sections, needs to be computed in terms of DR rather than pure clock cycle and/or throughput).

It is possible to show, although not in the scope of this paper, that a modified WP could allow such a flexibility at a minimal cost. We implemented a suitable protocol in a VHDL model, which proved to be completely compliant with the adaptive requirements.

We try here to develop a physical-design system that takes into account of minimal information in this respect, namely, the average channel occupation, to increase the possible throughput of a heavily WP system beyond the theoretical limits described and analyzed in the previous sections. We will proceed to deduce some basic cost functions in this section.

Equation (14) evaluates the effective throughput of a single loop. A system may entail more than one loop. The case of adaptive communications differs from the case where all loops are active at all time, and in which the effective throughput is statically determined by the worst loop. In the case at stake, the worst loop cannot be statically determined because it may vary from time to time, depending on the activation of the channels that connect the various blocks. However, providing a physical-design system with cost functions based on full profile information and details of operation is not practical. In order to derive simpler cost functions, we need some simplifications. A simple formula can be derived on the basis of an assumption.

Assumption 1: The blocks communicate in such a way that no physical time is lost in switching between the condition in which a channel is used and that in which the channel is idle.

The practical scheme we developed shows that, while the context switching does not entail any overhead for the local node (the wrappers are able to adapt immediately to the change in their inputs), information has to propagate throughout the loops thus generating a potential overhead that is related to the loop's length. However, if the communication occurs in bursts, the potential overhead of context switching will be amortized and made negligible. We will work under this hypothesis and show in the experimental section that is conducive to relatively predictable behavior.

Under the assumption, the system in each moment will work at a maximum throughput given by **the most critical active loop present**, which is the most critical loop whose branches are all active at the same time.

Bearing in mind the fact that the adaptive protocol will not modify the computation details, but only their timing, we can define the logical time as the number of clock cycles needed to perform a given computation in the system without pipelining, while the physical time takes into account possible nonvalid time slots introduced by the protocol.

In order to quantify the performance of the system, let us suppose that W_i represents the logical period of time in which the ith loop, with associated throughput TH_i, dominates the system's throughput. Then, the physical period corresponding to W_i computations is given by the formula W_i/TH_i, because each data calculation in such conditions implies a number of clock periods equal to $1/\mathrm{TH}_i$. Therefore, the overall physical time spent by the system to complete its computations is obtained by summing the physical periods of alternatively dominating loops, that is, $\sum_i W_i/\mathrm{TH}_i$, while the number of logical time steps is just $\Omega = \sum_i W_i$. This gives a value for the overall throughput of the computation of

$$\frac{\sum_i W_i}{\sum_i \frac{W_i}{\mathrm{TH}_i}} = \left(\sum_i \frac{W_i}{\Omega \cdot \mathrm{TH}_i}\right)^{-1} = \left(\sum_i \frac{w_i}{\mathrm{TH}_i}\right)^{-1} \tag{15}$$

where $w_i = W_i/\Omega$ is the time fraction in which the most critical throughput is TH_i. Even if this formulation allows us to estimate the performance of an adaptive system, it is still impossible to use it as a cost function for any optimization purpose because of the exponential loop dependence and the exponential number of loops.

In order to render the analysis of such a system manageable, we propose to define a quantity that defines each channel's activity independently of the rest of the system, the **channel activation ratio**, which represents the logical time fraction in which a channel is active. It is impossible to know such quantity without any detail of the system's behavior, and an exact evaluation still requires a complete profiling of the overall system, but the great advantage with respect to loop quantities is that their number is bounded by the system graph-edge size. Furthermore, the logical time in which a channel is used remains the same no matter how many pipeline stages we introduce and, therefore, can be assessed through a single profiling experiment (or better, averaging significant profiling) and not repeated for potentially, each and every different pipelining configuration, as implied by the techniques described in [4]–[6].

In order to exploit such quantities in a computation, we need to make the additional assumption.

Assumption 2: The channels' activation ratios can be considered statistically independent.

This assumption is of course far from being true in real cases, as communication channels tend to be strictly correlated from the functional point of view. However, the independence needed for our purposes does not imply a complete functional independence, but rather a single cycle decoupling, which is normally true at the hierarchical level we are considering. Blocks with high complexity will normally execute complex tasks internally before communicating with each other. An example of such cycle-level independence will be shown in the case of an MPEG decoder whose communication infrastructure is analyzed as a case study in Section VI.

We also studied cases somewhat between the spectrum of systems with complete uncorrelated signals (out MCNC and GSRC cases) and strictly deterministic behavior, such as that of a simple processor core, that behaves acceptably from this point of view (we cannot report such simulations for space reasons).

Under the independence assumption, together with assumption 1, which is enforced by the burst nature of most of the communications, it is easy to compute the values w_i for each loop. In fact, given a loop with n edges e_j, its weight w_i is simply the product of the single edge's activation α_{ji}, that is, $w_i = \Pi_j \alpha_{ji}$. This property is valid also under the less restrictive set of assumptions that allow for nonburst communication. Wrapping up the previous results

$$\mathrm{TH}_{\mathrm{tot}} = \left(\sum_i \frac{\prod_j \alpha_{ji}}{\mathrm{TH}_i}\right)^{-1}. \tag{16}$$

The formulation is clear, but it allows the computation of the overall throughput only by analyzing an exponential number of loops.

Given the previous discussion, the necessity for an heuristic computation to embed in existing physical-design environments is more pressing than ever; in order to perform some technology explorations, we tried to introduce an approximate calculation of the throughput that could be easily embedded in a floorplanning environment. We decided to use the function described in [3], modified in order to include the effect of channel activity, as detailed in the following steps.

1) For each pin to pin connection, we evaluate the Manhattan distance between the pins.
2) The distance is divided by the maximum length admissible between clocked elements.

TABLE IV
BLOCK DELAY EQUAL TO 13FO4, ADAPTIVE CASE. TH $E.$ AND $T.$ = ESTIMATED AND TRUE TH, ϵ = PERCENT ERROR, SU = PERCENT SPEEDUP

Bench	DR_0	f_1				f_2				f_3				f_4			
		th e.	th t.	ϵ(%)	su(%)	th e.	th t.	ϵ(%)	su(%)	th e.	th t.	ϵ(%)	su(%)	th e.	th t.	ϵ(%)	su(%)
n10	0.521	0.5	0.359	+39	**+6**	0.5	0.359	+39	**+6**	0.5	0.359	+39	**+6**	0.5	0.333	+50	**-2**
n30	0.408	0.322	0.343	-6	**+29**	0.29	0.296	-2	**+12**	0.318	0.314	+1	**+18**	0.351	0.359	-2	**+35**
n50	0.469	0.313	0.341	-8	**+12**	0.36	0.36	0	**+18**	0.347	0.338	+3	**+11**	0.346	0.346	0	**+13**
n100	0.423	0.345	0.357	-3	**+30**	0. 357	0.362	-1	**+32**	0.364	0.369	-1	**+34**	0.348	0.342	+2	**+24**
ami33	0.465	0.343	0.325	+6	**+8**	0. 376	0.339	+11	**+12**	0.362	0.348	+4	**+15**	0.344	0.313	+10	**+4**
ami49	0.465	0.339	0.333	+2	**+10**	0.266	0.269	-1	**-11**	0.303	0.269	+13	**-11**	0.303	0.282	+7	**-7**
xerox	0.478	0.302	0.269	+12	**-13**	0.270	0. 254	+6	**-18**	0.254	0.258	-2	**-17**	0.252	0.282	-11	**-9**
apte	0.397	0.286	0.286	0	**+11**	0.266	0.254	+5	**-2**	0.237	0.236	+5	**-9**	0.303	0.281	+8	**+9**
hp	0.531	0.396	0.355	+12	**+3**	0.354	0.318	+11	**-8**	0.353	0.316	+12	**-8**	0.353	0.336	+5	**-3**
AVG	0.462	0.350	0.330	6.0	**+10**	0.342	0.313	7.5	**+4.6**	0.338	0.311	5.5	**+4.4**	0.344	0.324	+6.1	**+7.3**

3) This last number is divided by the length of the smallest loop to which the connection belongs.
4) The number is multiplied by a weight related to the activity of the channel.
5) All such values are summed.

Incidentally, the only difference of this cost function with respect to the one described in [3] consists in the fourth step, here, in boldface: the algorithm used in [3] is equivalent to this last one when weights are all set to one (or in general equal to each other). As for the weight used to take into account the channel activity within the floorplan cost function, we tested four different strategies. The simplest consisted in multiplying for the activity ratio of the ith channel, $\times\alpha_i$ (function f_1 in the following). Another function was $\times 1/(2-\alpha_i)$, which does not underestimate the smaller activities (function f_2). Another was that of multiplying for the maximum activation ratio among the channels belonging to the loop under consideration $\times \max_i(\alpha_i)$ (function f_3). The last tried function (f_4) was rather different. We compute up front, before optimization, another heuristic multiplicative cost of passing through a node that takes into account the activities of the connections. In practice, instead of dividing by the length of the smallest loop each edge belongs to, as of point 3 of the procedure outlined in the previous section, we use a cost that is the effective throughput in (14).

In addition to the floorplan optimization, the tool output consists also of a system throughput estimation by means of a worst loop approximation. In practice, the tool finds the worst loop according to (14), not differently from [3] where (1) was used. As for the evaluation of the loop activation α in (14), we tested different solutions like for the cost functions used within the floorplan engine. We found that a better fit with VHDL simulations is obtained by searching the loop having minimum TH_e, calculated using $\alpha = \max_i(\alpha_i)$ in (14), with α_i being the activities of the edges that belong to such worst loop. The reason why such choice gives better results has still to be investigated.

VI. RESULTS AND COMPARISONS

In order to obtain some meaningful figures, we followed a twofold experimental evaluation: We first considered the MCNC and GSRC benchmarks of the rest of the paper and try to settle the question as to whether adaptive WP can provide acceptable advantages under the most adverse delay model (13FO4) discussed previously, then, we analyzed the performance of a well-defined system, an MPEG decoder, using real profiling information to simulate the systems.

A. GSRC Benchmarks

The experiments described here use the same benchmarks of Section IV, with a substantial difference: We introduced a number describing the activity of each channel between blocks, so as to make it possible to take advantage of adaptive schemes. As no functional information is given with the benchmarks, we decided to describe the channels as being used in a burst mode with lengths and relative phases uniformly distributed (coherently with the assumptions of the previous section). With respect to our previous work [8], we ran new experiments with more stringent conditions, i.e., with higher channel activities, in order not to bias too favorably the results.

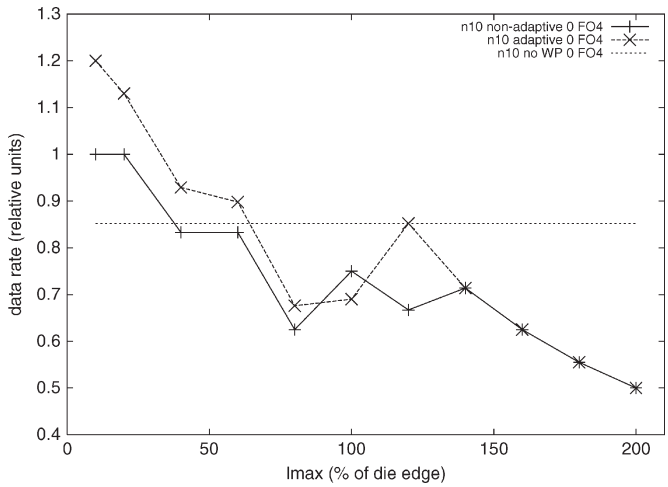

Fig. 1. n10: Optimization of DR with no block delay.

After generating (in a suitable format) the "new" benchmarks, we proceeded to compute two values: an optimized non-WP solution and an adaptive one. The first is generated in a similar way to Section IV. The adaptive solution is generated by employing a floorplanner that optimizes the heuristic cost function described in previous section (we modified PARQUET in such a way that loop computation and channel activity annotation are possible). The result is then automatically translated into a VHDL netlist, which mimics the behavior of the real system by allowing the adaptive communication between blocks. Each block functionality is simply that of a counter, which allows tracking the evolution of the system without the necessity of a full-scale simulation. Of course, each channel is represented by the appropriate pipelining delays derived from the floorplanner. The behavior of the channel is emulated by a simple function, which generates a bit of information per block input using the activity values derived from their description. The detailed implementation of the adaptive protocol is outside the scope of this paper; it is important to note, however, that it represents a real RTL design, which can be easily turned into a synthesizable description. An incidental result is therefore that of giving an opportunity to validate such an implementation. This VHDL model is then simulated and its real performance compared to the nonadaptive design. Various choices for the systems' frequencies were simulated in order to obtain a good approximation of the optimum. However, due to space reasons, we report only the results obtained for $l_{\max} = 13\mathrm{FO4}$

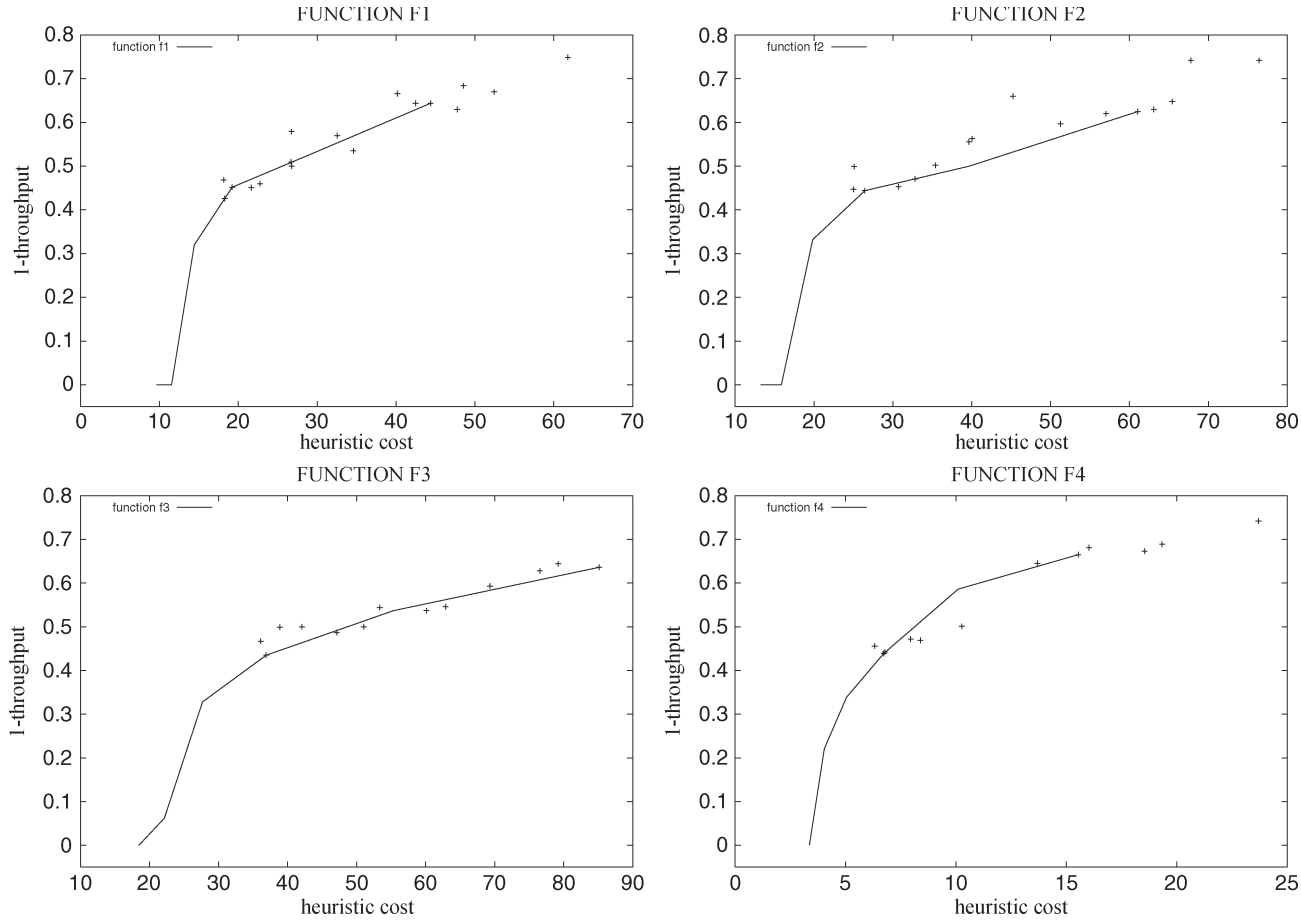

Fig. 2. GSRC n100 heuristic cost f1–f4 versus VHDL simulated cost during and after floorplan optimization.

which, as stated in Section IV, proved to be untractable—at least for the technological parameters we considered—by the nonadaptive pipelining shown before.

The results are shown in Table IV. The true DR, as outcome of the VHDL simulations, is reported along with the estimates from the floorplanner. The different results obtained using the different weighting functions (f_1–f_4) are reported. Since $l_{\max} = 65\%$ for all cases in the table, we did not report DR results, but only throughput, being the first readily obtained by the second divided by $l_{\max}$. As the table shows, but for few cases, the speedup is positive and in some cases also significantly high. It has to be reminded that, contrarily to Table III where the maximum DR has been reported varying $l_{\max}$, here, all results refer to the case $l_{\max} = 65\%$. Thus, it is likely that these are not optimal results in terms of speedup. Best SU results, in average, are obtained with function f_1 even though the differences are small and unfortunately do not help giving a definitive answer about which function is preferable. The error between tool estimates and simulated true values of throughput is small and proves the goodness of the TH estimator described before.

The particularly bad estimation performance in the case of n10 could be explained by the size of the benchmark: The estimation takes advantage from the averaging out effect of the many loops, which in this case is simply insufficient.

For a single case n10, we report in Fig. 1 the results obtained using cost function f_4 and varying $l_{\max}$. Two-DR curves for the nonadaptive and adaptive WP cases with 0FO4 block delays are reported. The horizontal line is the DR of the nonwire-pipeline case. We observe that the DR curves are not monotonic with respect to $l_{\max}$ or equivalently to the frequency. This implies that DR optimization has to deal with local optimum points. Nevertheless, the DR increases moving toward higher clock frequencies (lower $l_{\max}$). This result was anticipated in [3] for the nonadaptive case and is confirmed now for the new WP case considered in this paper. The reason is that the throughput degradation, both according to (1) and (14), is overcompensated by the increase of clock frequency (decrease of $l_{\max}$).

B. Cost-Function Assessment

We report here some results we obtained with the aim of assessing the quality of our cost functions. Solid lines in Fig. 2 represent heuristic costs f1–f4 versus real cost, viz. 1-throughput, obtained after floorplan optimizations of benchmark n100 varying $l_{\max}$ from 65% to 300%. True costs are estimated by postfloorplan-cycle accurate-VHDL simulations. Unconnected points represent intermediate floorplans before final optimization. Estimated and real costs are positively correlated, as was expected. All curves show a good behavior, although unconnected points seem less aligned in f2.

C. Case Study: MPEG

In this section, we will try to validate the methodology with a real-life example that is the typical (although small) case for which heavy WP might be appropriate: an MPEG decoder. We follow in this example the implementation of the decoder described in [18]. In such implementation, the various blocks of the system communicate with each other in bursts followed by channel idle periods. Owing to a useful timing diagram of activation of the various constituent blocks of the decoder reported in [18], it is possible to quantify exactly the

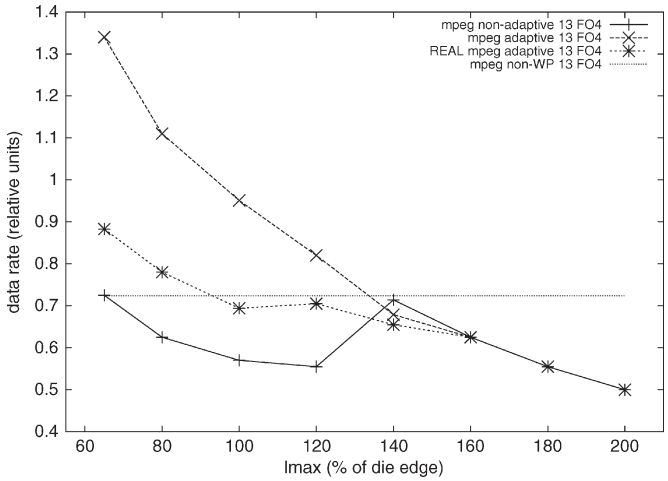

Fig. 3. MPEG: Optimization of DR with 13FO4.

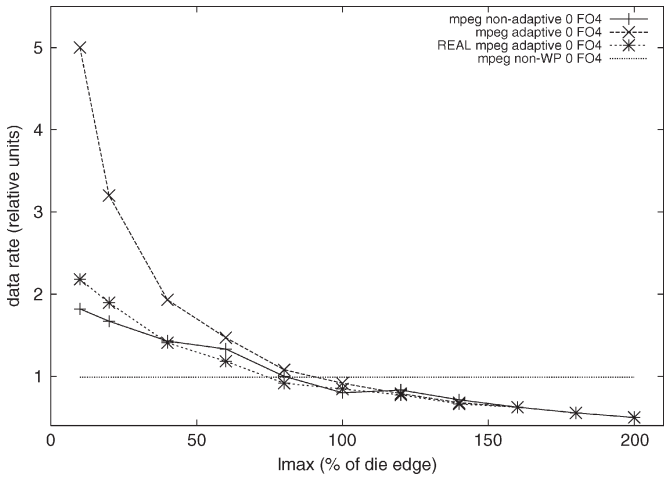

Fig. 4. MPEG: Optimization of DR with no block delay.

duration of the activity periods, like the number of clock cycles to perform a given task, and relate it to the number of cycles needed for a complete MPEG frame decoding. It is thus possible to estimate the channel activation factors and formulate a floorplanning problem, which is compliant with our description as of Section IV. The only difference is that the system enforces a strict data communication dependence that contradicts the randomness hypothesis: However, the experimental results show that the optimization potential of a pipelined solution is substantial.

We first described the system in standard format, together with channel annotations and activities taken from the real behavior. Then, as a first phase, we optimized the system as detailed for the other benchmarks, that is, using randomly generated phases of the channel activities (bursts of real duration but with random beginning). In a second phase, we used the optimal floorplan thus obtained (both case 0 and 13FO4) and simulated the communication channel with the real system timing, rather than random bursts using a VHDL simulator.

Figs. 3 and 4 report the DR curves already reported in Fig. 1 for the n10 benchmark (function f_4), now both for 13 and 0FO4 delays. DR curves obtained with the "real timing" of the MPEG blocks are plotted as well. The impact of adaptive wire pipelines is more effective in increasing the overall performance, especially when the block delays are negligible. In the 13FO4 case, the minimum possible $l_{\max}$ is 65% of the die edge because of the frequency constraint set by the block delays. The gains at $l_{\max} = 65\%$ are 0% for the nonadaptive case and 85% and 22% for the two adaptive cases, "random" and "real" burst phases, respectively. In the no-delay case, the gain depends on how low $l_{\max}$ is kept compatibly with technology and logic block constraints. As for the latter, we assume that an aggressive gate-level pipelining is able to push the block frequencies at the 13FO4 case and possibly more, in such a way that system clock frequency is always limited by the interconnects. For example, at $l_{\max} = 60\%$, the gains are $+35\%$, $+48\%$, and $+19\%$ for the same three cases, while at 10%, we get $2\times$, $5\times$, and $2.2\times$.

VII. Conclusion

In this paper, we focused on the floorplan of systems-on-chip based on the use of IP blocks and WP interconnects to improve the DR. We first introduced a mathematical framework to give a sound basis, which helped understanding the opportunities as well as the limits of this technique. In particular, we analyzed and modeled the detrimental effect of the delays of the IP blocks, including internal logic and interconnect delays. The maths suggested that not keeping such delays as small as possible would jeopardize the increase of performance that WP seems to promise. The experimental results on GSRC and MCNC benchmarks we obtained, by running the floorplanner we presented in [3] in this new context, confirmed the mathematical intuitions.

In the second part of this paper, we considered the use of "adaptive" WP, which tries to benefit of the clock-frequency increase of WP while limiting the reduction of system throughput that is due to the latency of pipeline elements in the loops of the system netlist. We adapted the floorplan tool to this new context by changing the cost function used in the simulated annealing algorithm. Running the experiments over the same benchmarks and in a realistic case study showed that it is possible to lessen the impact of block delays and to recover the speedup that the standard application of WP loses.

In the future, we would like to concentrate on three issues: finding cost functions that best estimate the real performance, both to be used inside the optimization loop and for final evaluation; relaxing the assumptions of this paper, including the analysis of multiclock domain systems; and comparing the performance of the floorplanner with other similar contributions [4]–[6] by addressing the somewhat different problem of microarchitecture planning.

References

[1] The International Technology Roadmap for Semiconductors, SIA. 2003.
[2] M. R. Casu and L. Macchiarulo, "Floorplanning for throughput," in *Proc. ISPD*, Phoenix, AZ, Apr. 2004, pp. 62–69.
[3] ——, "Throughput-driven floorplanning with wire pipelining," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 5, pp. 663–675, May 2005.
[4] M. Ekpanyapong *et al.*, "Profile-guided microarchitectural floorplanning for deep submicron processor design," in *Proc. DAC*, San Diego, CA, Jun. 2004, pp. 634–639.
[5] C. Long *et al.*, "Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects," in *Proc. DAC*, San Diego, CA, Jun. 2004, pp. 640–645.
[6] V. Nookala *et al.*, "Microarchitectural-aware floorplanning using a statistical design of experiments approach," in *Proc. DAC*, Anaheim, CA, Jun. 2005, pp. 579–584.
[7] J. Wang, P. C. Wu, and H. Zhou, "Processing rate optimization by sequential system floorplanning," in *Proc. ISQED*, 2006, pp. 340–345.
[8] M. R. Casu and L. Macchiarulo, "Floorplan assisted data rate enhancement through wire pipelining: A real assessment," in *Proc. ISPD*, San Francisco, CA, Apr. 2005, pp. 121–128.
[9] L. P. Carloni *et al.*, "A methodology for correct-by-construction latency insensitive design," in *Proc. ICCAD*, 1999, pp. 309–315.
[10] M. R. Casu and L. Macchiarulo, "A new approach to latency insensitive design," in *Proc. DAC*, San Diego, CA, Jun. 2004, pp. 576–581.
[11] ——, "On-chip transparent wire pipelining," in *Proc. ICCD*, Oct. 2004, pp. 160–167.

[12] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Performance analysis and optimization of latency insensitive protocols," in *Proc. DAC*, pp. 361–367.
[13] Y. I. Ismail and E. G. Friedman, "Effects of inductance on the propagation delay and repeater insertion in VLSI circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 2, pp. 195–206, Apr. 2000.
[14] W. Dai, L. Wu, and S. Zhang, *GSRC T2 Bookshelf at UCSanta Cruz*. 2003. [Online]. Available: www.cse.ucsc.edu/research/surf/GSRC/progress.html
[15] M. Singh and M. Theobald, "Generalized latency insensitive systems for single-clock and multi-clock architectures," in *Proc. DATE*, Paris, France, 2004, p. 21 008.
[16] S. Adya, H. H. Chan, and I. Markov Parquet, *Fixed-Outline Floorplanner,* (2006). [Online]. Available: http://vlsicad.eecs.umich.edu/BK/parquet/
[17] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.
[18] M. Ikeda *et al.*, "A hardware/software concurrent design for real-time SP@ML MPEG2 video-encoder chip set," in *Proc. Eur. Des. and Test Conf.*, Mar. 1996, pp. 320–326.

An Efficient Data Structure for Maxplus Merge in Dynamic Programming

Ruiming Chen and Hai Zhou

***Abstract*—Dynamic programming is a useful technique to handle slicing floorplan, technology mapping, and buffering problems, where many maxplus merge operations of solution lists are needed. Shi proposed an efficient $O(n \log n)$ time algorithm to speed up the merge operation. Based on balanced binary search trees, his algorithm showed superb performance with the most unbalanced sizes of merging solution lists. The authors propose in this paper a more efficient data structure for the merge operations. With parameters to adjust adaptively, their algorithm works better than Shi's under all cases, unbalanced, balanced, and mix sizes. Their data structure is also simpler.**

***Index Terms*—Data structure, dynamic programming, timing optimization.**

I. Introduction

Dynamic programming is an effective technique to handle slicing floorplan [1], technology mapping [2], and buffering [3] problems. For example, van Ginneken [3] proposed a dynamic programming method to complete buffer insertion in distributed RC-tree networks for minimal Elmore delay, and his method runs in $O(n^2)$ time and space, where n is the number of legal buffer positions. An essential operation in van Ginneken's algorithm is to merge two candidate lists into one list where inferior candidates are pruned. Shi [4] proposed an efficient algorithm that improves Stockmeyer's algorithm [1] for the merge operation in slicing floorplan. Based on it, Shi and Li [5] presented an $O(n \log^2 n)$ algorithm for the optimal buffer insertion problem. In these algorithms, a balanced binary search tree is used to represent a list of solution candidates, and it avoids updating every candidate during the merge of two candidate lists. However, as shown in [4], the merge of two candidate lists based on balanced binary search trees can only speed up the merge of two candidate lists of much different lengths (unbalanced situation), but not the merge of two candidate lists of similar lengths (balanced situation).

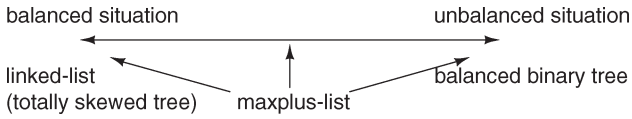

Fig. 1. Flexibility of maxplus-list.

Fig. 1 illustrates the best data structure for maintaining solutions in each of the two extreme cases: the balanced situation requires a linked list that can be viewed as a totally skewed tree or the unbalanced situation requires a balanced binary tree. However, most cases in reality are between these extremes, where neither data structure is the best. As we can see, the most balanced situation requires the most skewed data structure while the most unbalanced situation requires the most balanced data structure. Therefore, we need a data structure that is between a linked list and a balanced binary tree for the cases in the middle. We discovered that a skip-list [6] is such a data structure as it migrates smoothly between a linked list and a balanced tree. In this paper, we propose an efficient data structure called maxplus-list based on the skip-list and corresponding algorithms for merge operations in the dynamic programming. As shown in Fig. 1, we can migrate the maxplus-lists to suitable positions based on how balanced the routing tree is; a maxplus-list becomes a linked-list in balanced situations or it behaves like a balanced binary tree in unbalanced situations. Therefore, the performance of our algorithm is always very good. The maxplus-merge algorithm based on maxplus-list has the same asymptotic time complexity as the merge algorithm used in [4] and [5]. Our experimental results show that it is even faster than the balanced binary search tree in unbalanced situations, and it is much faster in balanced situations. Besides, the maxplus-list data structure is much easier to understand and implement than balanced binary search tree.

The rest of this paper is organized as follows. In Section II, the general problem of merging two candidate lists is formulated, and the skip-list data structure is reviewed. In Section III, the maxplus-list data structure and an efficient algorithm to merge two maxplus-lists are shown. In Section IV, the approach for finding the optimal solutions after the bottom-up merge operations is shown. The experimental results are reported in Section V. Finally, the conclusion and future work are given in Section VI.

II. Preliminary

A. Maxplus Problem

The following three different problems have the similar algorithmic structure, the merge of candidate solution lists.

Given a slicing tree representing a floorplan, the problem of area minimization is to select the size of each module such that the chip area is minimized [1]. The dynamic programming approach [1] builds the solutions bottom up. Each solution (h_v, w_v) at a node v represents a floorplan at v having h_v as the height and w_v as the width. As shown in Fig. 2(a), given the solutions (h_m, w_m), (h_n, w_n) of the two subtrees and a parent node with vertical cut, a candidate solution at the parent node can be constructed as $(\max(h_m, h_n), w_m + w_n)$. The optimal structure of dynamic programming requires that there are no solutions (h_1, w_1) and (h_2, w_2) such that $h_1 \leq h_2$ and $w_1 \leq w_2$ for the same subtree.

Manuscript received April 8, 2005; revised August 19, 2005 and November 23, 2005. This paper was recommended by Associate Editor J. Lillis.

The authors are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 USA (e-mail: haizhou@ece.northwestern.edu).

Digital Object Identifier 10.1109/TCAD.2006.882479