On-Chip Transparent Wire Pipelining

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

Abstract

Wire pipelining has been proposed as a viable mean to break the discrepancy between decreasing gate delays and increasing wire delays in deep-submicron technologies. Far from being a straightforwardly applicable technique, this methodology requires a number of design modifications in order to insert it seamlessly in the current design flow. In this paper we briefly survey the methods presented by other researchers in the field and then we thoroughly analyze the solutions we recently proposed, ranging from system-level wire pipelining to physical design aspects.

1. Introduction

The increasing complexity of integrated systems has naturally led to the idea of System-on-Chip where designers connect together complex components taken from libraries of Intellectual Properties (IP) like in board-level design. The performance of such systems is communication-constrained instead of computation-constrained as in almost all previous integrated circuit designs. The platform-design philosophy aims at a correct-by-construction integration of IP components. The driving force is the *orthogonalization of concerns* that is the capability of reducing the correlation among the various issues and facing them separately so as to improve the overall quality of the implementation [1]. The idea is that of separating computation from communication in order to avoid the performance being limited by the time required for moving data from one place to another within the chip. Such orthogonalization brings about the necessity of hiding physical/electrical details from higher-level design by making them, so to say, *transparent*. Technology scaling, however, play against such a scenario. In particular, *interconnection delay* threatens the projected *2001 International Technology Roadmap for Semiconductors (ITRS)* [2] evolution, because constant or slightly increasing chips' size and integration of more and more functional blocks cause the length of global wires not to scale down properly. This will make it impossible to respect the projected clock period and at the same time to allow all blocks in a chip to communicate within a single clock cycle. As a result, interconnect architects have been brought to borrow the classic pipelining technique from logic. This technique we call *wire pipelining (WP)* poses a number of issues, the most important of which, we believe, is the alteration of functionality due to the insertion in the logic netlist of *new* flipflops that break interconnect in shorter, and faster, segments.

This problem has been tackled in a number of recent works, ranging from the system-level aspects to the more physical ones. A survey of the literature on this topic is presented in section 2 where we review known solutions to the presented problem and evaluate their "transparency" to the high level designers. After an estimation of the number of pipelining elements needed to meet the clock frequency as indicated by the ITRS roadmap in section 3, we will then concentrate in the following sections on a family of solutions we have been working on in the past years. Finally we make an overall summarization giving also a perspective of the work still needed in the field.

2. A Survey on Wire Pipelining

In the following, past research in the field of wire pipelining will be summarized.

L. Carloni *et al.* proposed the "latency insensitive" (LI) design methodology [3][4][5] that allows the preservation of functionality when a system which is designed to work under zero wire delay constraint is modified with a certain amount of added wire latency. The modified system works by means of a *Latency Insensitive Protocol* (LIP) whose the bases are described in the next sections. We recently demonstrated in [6] another approach to latency insensitive design that does not require the routing of the protocol signals while preserving the performance of the previous approach. It can be shown that these approaches, while guaranteeing that the clock frequency constraint imposed by functional blocks is met, lead to a throughput degradation (i.e. the number of new data produced per clock cycle) that is *solely* related to the blocks netlist and in particular to the existence of loops in the circuit graph. That means that the throughput is independent of the data that block exchange and can be predicted upfront by a netlist graph traversal, in other words the performance is *static*. This normally does not come at the cost of an overall data-rate reduction (i.e. the product of throughput and clock frequency) with respect to a pure slowdown of the clock frequency to fit the global wires delay, but it certainly does not fully respect the throughput increase that WP seems to promise. To overcome this limitation is necessary to break the complete orthogonalization of computation and communication and give the protocol some (minimum) knowledge

Proceedings of the IEEE International Conference on Computer Design (ICCD'04)
1063-6404/04 $ 20.00 IEEE

of when the blocks are willing to transmit newly computed data. This approach was briefly introduced in [7].

While not totally related to the WP problem, we mention the *GALS* approach (Globally-Asynchronous Locally-Synchronous) where a fully-synchronous paradigm is adopted *within* computational blocks and a *handshake* protocol is used for the *intra-blocks* communication. While GALS present several advantages, among them the fact that clock constraints are largely relaxed compared to a fully synchronous approach, the performance may be limited by the propagation of signal across long wires. In order to both pipeline the wires and provide an interface to different clock domains Chelcea and Nowick proposed an asynchronous FIFO approach in [8] while showing the the latency-insensitive approach can be easily adapted to GALS systems. In this case the performance will be substantially limited by the netlist loops like the case of fully synchronous latency insensitive methodologies.

Another approach borders with classical retiming [9][10][11][12]. Wire pipelining is not obtained by adding new flipflops within the long wires but by moving already existing memory elements from logic blocks to interconnects or from blocks' inputs to blocks' outputs. This technique can only be applied if logic blocks are described at least at RT level and the logic description or the post-synthesis netlist are available. That is of course not possible for *hard-IP's*. As for *soft-IP's*, while theoretically possible, the manual rework to account for pipelined global wires at the RT level might severely impact the design costs. Moreover, there is no guarantee that retiming allows long wires to run at an arbitrarily high clock frequency like the real wire pipelining case. The combination of retiming, where its application is allowed, and other techniques like interconnect latency insensitive design, may lead to further improvements as it's been shown in [13].

J. Cong recently proposed [14] a methodology to totally avoid the manual rework at RTL level to include WP. The design is mapped onto a *regular distributed register (RDR)* architecture that supports wire pipelining and efficient sharing of global wires. Since the work deals with architecture-level synthesis, the entry point in the proposed design-flow is a C/VHDL description of the architecture at the RT level. The methodology does not support the integration of hard IPs without RTL description differently from LIP's where this is naturally accounted for.

In order to solve functionality problems deriving from WP, a recent work proposes an algorithm working at the gate level [15]. Compared to LIP, the new method has the advantage of not requiring additional protocol signals but still have one of the limitations of retiming since a gate level netlist is required instead of a collection of IP blocks. In addition, this method preserves functionality at the cost of a throughput reduction that may be generally worse than in the LIP case. In fact the throughput reduction (slowdown in the paper's vocabulary) provoked by the flifplop addition in netlist loops is always *integer* (i.e. a factor of 2 or 3 and so on) while LIP's allow a fractionary throughput reduction that is in general lower for a given number of added flipflops as shown later on.

A relevant number of papers concentrated on how the techniques developed for the routing and buffer insertion can be adapted to flipflop placement for WP. In this sense they can be considered complementary to previous works that address functionality alteration determined by wire pipelines. In [16] and [17] only pin-to-pin connections are considered while [18][19] also consider the case of multipin connections. In a recent paper, D.K. Tong and E.F.Y. Young [20] proposed a method for register placement along global wires, given a retiming solution. All these papers, however, do not address floorplanning issues and how netlist connections translate on final performance achievable from the buffer-flipflop routing after a given placement. As far as we know, these issues were addressed for the first time in our work [21]. We included in a simulated-annealing floorplanning tool a cost function that takes into account the throughput reduction due to the addition of flipflops in the netlist loops with better results compared to standard wirelength or area goals. In two more recent works, that with less generality only address the performance of microprocessors, floorplan tools have been modified so as to consider the cycles per instruction (CPI) metric as a cost function [22][23]. This performance index is in fact affected by the latency of wires connecting together the basic blocks of a superscalar microprocessor. The first paper uses a simulation-based profiling for estimating the impact of wire pipelining on CPI and then to drive a floorplanning algorithm. The second one inserts CPI in the cost function of a simulated-annealing based floorplan tool, after having launched a number of offline microarchitectural simulations.

3. Estimation of WP Elements

The first part of this work aims at better analyzing the interconnect delay evolution through a series of simulations based on data reported in the ITRS, to find out how much on-chip communications will impact performance during next years. The 2001 ITRS reports for the "near-term" years the values of projected chip dimension (area and number of transistors) and clock frequency; the latter is summarized in Table 1, while area is fixed at 280 mm^2 for all technologies, with integrated transistors ranging from about 190 to 770 million.

Assuming the correctness of these predictions, we started considering roadmap clock frequency as a value that in the future will be necessary to reach; then for each of these technology nodes we took an interconnection with variable length from 1 to 34 mm (this last value models a corner-to-corner wire on a 280 mm^2 square chip) by steps of 3 mm, and we calculated its delay using the *Marco GSRC Technology Extrapolation (GTX) System* [24] with *BACPAC* model [25]; some of the BACPAC's formulas were a bit modified to include a more accurate *RLC* line description and *optimal repeaters* insertion, as described in [26]. For each wire, we also made the reasonable hypothesis that we could change its width w inside a set of integer multiples of its minimum value in the used technology. In this way, for a given interconnection we calculated the *best achievable delay*, i.e. supposing all the optimizations, buffering and width sizing, could be practically realized. In particular, comparing $f_{ck_{ITRS}}$ with the results of these simulations we were able to evaluate the *maximum critical distance* for every technology (Table 1), defined as the longest segment of wire that a signal can reach in a single clock cycle. An interconnection longer than $l_{crit,max}$ will be "critical" in any case, because a signal will *surely* require more than one clock period to propagate, thus needing to insert one or more memory elements (ME in the table, i.e. *flip-flops*) along the line (one after at most

Year	Techn.	$f_{ck_{ITRS}}$ [GHz]	$l_{crit,max}$ [mm]	Needed MEs	
				10 mm	34 mm
2001	130 nm	1.684	17.11	0	1
2002	115 nm	2.317	12.17	0	2
2003	100 nm	3.088	8.95	1	3
2004	90 nm	3.990	7.37	1	4
2005	80 nm	5.173	5.28	1	6
2006	70 nm	5.631	4.63	2	7
2007	65 nm	6.739	4.16	2	8

Table 1. 2001 ITRS projected frequencies, with critical lengths and memory elements for a single wire

$l_{crit,max}$). Table 1 shows the minimum required number of them for two wirelengths of 10 mm and 34 mm, as obtained from all our simulations.

Other experiences showed that wire pipelining gives rise to particularly interesting power/delay tradeoffs, by allowing suboptimal wire buffering. After studying delay and power of a single wire, we tried to model on-chip wirelength distribution and thus to get a measure of how many memory elements should be inserted. To do so, we took real IBM circuits' data from the *ISPD98 Circuit Benchmark Suite* [27] and we applied the model presented in [28] to each of these designs, supposing each of them ideally partitioned in 64 IP-blocks. Projections for the 65nm node are that the number of needed FFs is in the order of 20,000, for a maximum area die. This overall analysis shows how important these considerations will be in future designs.

4. Latency Insensitive Protocols

The analysis of the minimum latency of wires of a given length and the prediction of the number of memory elements done so far are independent on how this added flipflops modify the system functionality. One of the methods we surveyed in section 2, makes the new flipflops and the consequent added wire latency totally transparent to the designer. The Latency Insensitive Design (LID) methodology requires that the original modules connected through pipelined wires are *stallable*, i.e. they can be stopped at any time and they are guaranteed to hold their state throughout the interruption. This condition is normally satisfied if modules are synchronously holding their data in internal memory elements, which can be therefore controlled by standard clock-gating techniques. Given the assumption, LID methodology proceeds as a 3-step process:
- each module (a *pearl*) is **encapsulated** within a wrapper (called *shell*) that performs clock-gating and handles interface signals;
- **Physical Layout** is performed, after which long connection that would result in timing violation are identified;
- Critical wires are **segmented** by introducing the appropriate number of *relay stations* which allow for some internal buffering and handle stalling signals of the protocol.

It is worth noticing that the protocol requires the added memory elements along the wires to not be simple flifplops but instead to contain at least two registers, which increase the flipflop count of section 3. The relay station is nothing but another queue whose size is limited to two places; the first register pipelines data in normal operation; the auxiliary register is allowed to store another data when the relay station receives a stall signal and then must maintain the previous data in the first register. The shell may also be provided with input queues for a limited input data storage while the pearl is stalled. These wrappers have to fulfill three tasks:
- **Data Validation**: each output channel has to signal whether the datum therein present is a new one to be consumed by asserting a *valid* bit;
- **Back Pressure**: when the pearl is stopped for some reason and cannot accept more data because it cannot store them, the shell has to generate a *stop* signal sent in the opposite direction of inputs;
- **Clock Gating**: a module waiting for new data and/or stopped by following modules needs to keep its present state.

Such a protocol was implemented in [3] through the introduction of two new signals per channel, alongside with the following interpretation:
- *valid*: accompanies a datum which is actually a new valid packet;
- *stop*: propagates in the opposite direction to indicate that what precedes has to be stopped, because the shell or relay station is not ready to receive new data.

In the papers that first introduced the latency insensitive methodology, shells and relay stations are said to have the "compositional" property, or in other words they can be connected together while guaranteeing the preservation of the protocol and its reliability. It can be shown (details omitted) that, in order for any pair of blocks to communicate within the protocol, some queuing capability is needed at the inputs (or in the communication channels) no matter the necessity of pipelining. We therefore introduce the concept of "half relay station", a relay station which contains a single pipelining register (instead of two) and allows direct connection between input and output. The role of this register seems to be played by the "extended relay station" according to the vocabulary of one of the original papers [5] placed on the wrapper's output. We will re-interpret the compositional property of the shells by imposing the constraint of adding at least one half or full relay station between two of them. This will guarantee at the same time that no data are lost and will help to improve the performance when necessary.

In some previous works ([32], [31]) we described a complete RTL implementation of such a protocol. By way of employing such a description we have been able to experience with the protocol at an implementation level, and experimentally validate the properties that we will discuss in the following.

We employed a tool of formal verification in which it was possible to describe our basic blocks at the same description level of the design, t.i. RTL. We dealt with safety and liveness problems separately. We define an RTL implementation of a LIP **safe** iff any composition of blocks subject to certain constraints and with primary inputs subject to certain assumptions, will behave in a latency insensitive sense exactly as an equally connected system wherein wrappers and shells are substituted by non/pipelined connections. We define the implementation **live** iff no conditions will

occur that will put the system in a deadlock condition, that is, a situation in which no new data is ever produced. Note that a safe system is not necessarily deadlock-free. We proved, using the formal verification tool SMV [29] that our implementation is always safe and live for most common netlist topologies.

5. A priori throughput evaluation

As noticed in section 2, the performance of LIPs can be predicted upfront in a *static* way, based on the circuit blocks netlist only. In this section we will try to summarize a few conclusions we reached by proving basic timing relations dependant on overall circuit topology. The attention will be focused, for reasons that will become apparent in the following, on two figures: *System Throughput* and *Transient Length*. It turns out to be possible to define the second figure due to an interesting consequence of the protocol: after a number of clock cycles that are dependent on the system (and can be indicated with precision) each part of it behaves in a periodic fashion. We baptize the first part as **transient**,

It is natural to associate a direct, possibly cyclic graph to a system of interconnected synchronous processes, where each vertex represents a shell and each edge is a unidirectional channel where data flow. There are representative topologies for such a graph whose performance can be easily derived and that help understanding the behavior of complex systems. Their discussion will make it possible to formulate some rules that are helpful in designing optimum relay station arrangements, when some flexibility is permitted.

The simplest topology is represented by a *tree*, where for each possible couple of vertices (u, v) there is only one path connecting u to v. The throughput of each node in the graph, i.e. the number of valid data per clock cycle is 1 as for the original system. However, each relay station is initialized with non valid outputs that must be eliminated flowing toward the primary outputs[1]. Thus the initial latency for each node before firing at full speed can be as much as the longest path in the tree (transient duration), where the length is defined by the sum of the weights of each traversed edge and the number of vertices in the path.

If we allow one or more couple of vertices in a direct acyclic graph to be connected by two or more paths we obtain what we call the "reconvergent inputs" topology. Its behavior differs from that of trees due to implicit loops created by the introduction of reverse-flowing stop signals. To understand the problem, let's consider the simple example of figure 1. Three blocks are arranged as shown, with a minimum number of relay stations.

Now, let's follow the system's evolution from the legal initial condition that assigns valid values at all initial outputs of the three shells. Such an evolution is shown in figure 1 where "n"s represent data which are not valid, while stops are indicated by dashed arrows and stopped modules by dashed blocks. It is apparent that after the initial transient, the situation becomes periodic, and the output utters an invalid datum every 5 clock periods. This value can be explained as follows: the unbalanced number of relay stations in the two branches introduces an unequal number of relays on every reconvergent path, thus constraining the longest one to introduce a number of invalid data in an otherwise constant throughput. A single invalid token gets propagated from top to bottom on the longest branch, and then generate a stop signal that propagates on the shortest one from the bottom up every n cycles. On the other hand, the number of invalid data that propagate is equal to the difference of relay stations between the feedforward branches. In the present case, $n = 5$, while $i = 2 - 1 = 1$. This means that the number of valid data every 4 periods is $5 - 1 = 4$ and the throughput is equal to $\frac{4}{5}$, as expected. The general formula is as follows: $T = \frac{m-n}{m}$, where m is the total number of relay station in the loop, plus the number of shells on the path with the highest number of relay station (the other branch's shells don't count due to the combinational propagations of the stops), while n is the difference in number of relay stations between the two branches. The general conclusion we can draw from these examples is that, in order to obtain the maximum throughput from a feedforward arrangement, it is necessary to insert enough spare relay stations to make all converging paths of the same length (length being defined on the basis of number of relay station). We call this condition **path equalization**, and note that it is completely different from the equalization described in [4] whose aim is forcing all connected components of the design to work at the same throughput. Even though such equalization can be achieved manually through relay station insertion, it is also possible to obtain it automatically by introducing enough buffering capability on the branches, for example by modifying the shell structure allowing for inclusion of a FIFO.

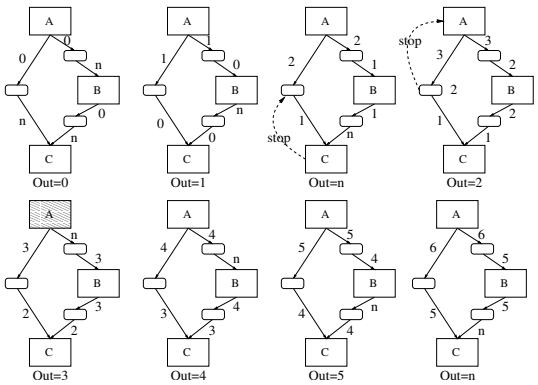

Figure 1. FeedForward Topology Evolution.

Graphs containing loops as in figure 2 are responsible for the worst throughput reduction in latency insensitive protocols when relay stations have to be inserted within the cycle. In this case the evolution from the starting state shows a behavior dictated mainly by the features of the loop with the least ratio between shells and relay stations. In fact, in such a loop, a maximum of S valid data can be present at a time, out of $S + R$ positions (where R is the number of relay stations in the loop). This justifies the number $\frac{S}{S+R}$ for the maximum throughput. This result is fundamentally the same discussed by Carloni in [4]. For the specific case shown in figure 2, two loops that exhibit a "stand-alone" throughput of $\frac{2}{4}$ and $\frac{2}{5}$ respectively interact in such a way that the entire system works at the worst throughput of $\frac{2}{5}$, as shown in figure where each block utters 2 valid and 3 non valid ("n") data every 5 cycles.

1 The shells outputs are instead initialized with valid outputs

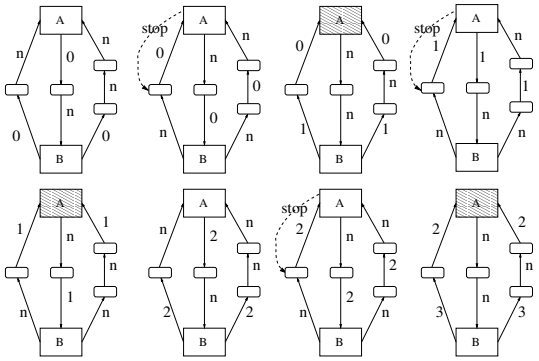

Figure 2. FeedBack Topology Evolution.

The most general topology can be seen as a feed-forward combination of self-interacting loops. It is possible to prove that the slowest subtopology (either reconvergent feed-forward or feedback) will force the system to slow down to its speed. The protocol itself will adapt to such a speed without any need for path equalization, that might help in reducing the transient time and/or increase the throughput (of the feed-forward components only). The transient length, in general, is a linear function of the number of relay stations and shells.

6. Experimental Validation

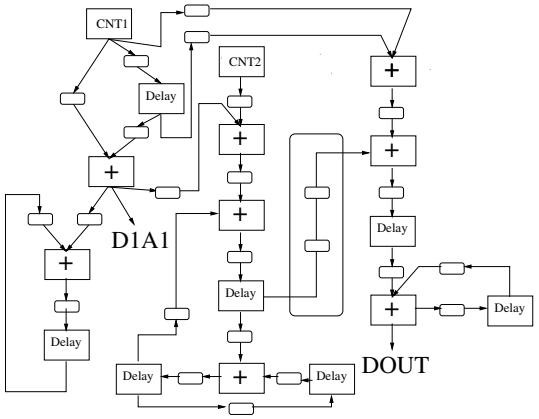

Figure 3. Complex System under Analysis.

To validate our protocol, together with the results exposed in the previous sections, we used a proof-of-concept example that comprises various combinations of feedforward and feedback topologies. The system shown in figure 3, where computational elements are connected through wires pipelined by relay stations, was compared to the corresponding original case without shells and relay stations. The functional blocks are simple up counters, adders, and pure delays, whose choice allows easy identification of functional mismatches between the implementations.

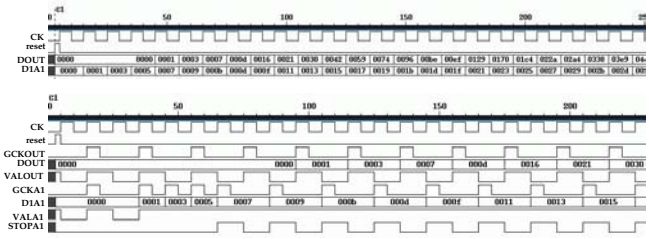

Figure 4. Snapshots of Simulations for system of figure 3.

The snapshots reported in figure 4 show signals in channels labeled as DOUT and D1A1, in both implementations (the latency insensitive version has also the corresponding valid, stop and gated clock signals plotted). The two simulations are identical when compared on the relevant data (as explained in section 3). Furthermore, it is clear that the throughput reduction to $\frac{1}{2}$ (one valid every 2 clock ticks in the latency insensitive versione) justified by the short loop made of the adder generating DOUT and its delay.

7. Static Scheduling Solution

Perhaps the most relevant property of the LIP methodology is its straightforward applicability, the only requirement needed for its implementation being the "stalling" capability of pearls, which may be achieved with latched blocks and standard clock gating techniques. However the requirements of the protocol from the physical design side are relevant. Every communication channel must be provided with a couple of additional signals, such as Carloni's "void" and "stop", or "valid" and "stop" in other works [31], [32]. Such signals sum up to the total wiring requirements increasing the already critical wire congestion. In addition, the insertion of relay stations along the interconnects poses serious area constraints since it requires available spaces in the floorplan for placing at least two registers and a (small) finite state machine for each added relay station [3]. The shell also should be provided with a few logic gates for clock gating, output data validation and input data back pressure that are the protocol relevant operations. These requirements can be alleviated if a clock schedule for the functional blocks is defined. Using a suitably adapted algorithm a schedule can be found which is optimal in the sense that it guarantees the maximum throughput allowed by the structure of the sequential system. In addition the method allows the use of simple flipflops instead of complete relay stations. There is no more need for routing the protocol signals, thus the saving in routing resources is relevant. We point out that the application of the method is at least as general as the original LIP (as described in [3]). In fact, due to *orthogonalisation of concerns*, no info can be drawn from the IPs during their operation: Therefore void and stop signals are generated in a completely data and block-independent fashion (they mainly depend on topology). The only reasonable assumption on IP blocks is that they have the potentiality of react-

ing to each and every input which has been validated by the protocol. Under such assumptions, as we will show, our method has the same range of applications and performance as the classical LIP methodology, though it saves resources.

To preserve the system functionality, we can control the synchronicity by selectively controlling the elements' clocks in such a way they react only to valid signals. This requires an overall clock scheduling that will activate the various units in a coordinated way. The problem can be cast as a maximum time-to-ratio problem [34].

Once and if the schedule for the correct activation of functional modules has been found, the implementation of the latency insensitive system becomes straightforward. As for the periodic schedule, it is sufficient to initialize a shift register for each "pearl" at reset with the sequence of clock enable/disable. The output of the register is used to gate the global clock so as to produce the strobe pulse for activating the functional module within the "shell." The output of the shell is sent directly to another shell or to a flip-flop, depending on the latency of the interconnect. In figure 5, two shells communicate with a unit latency and are activated two times every three clock ticks (two '1's loaded in their shift-registers).

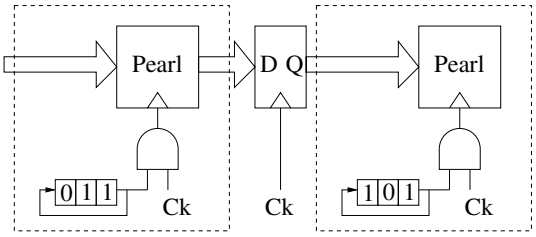

Figure 5. Conceptual scheme for scheduled pearls.

If we compare the system in figure 5 to an analogous latency insensitive system [3], the saving in terms of resources is potentially relevant. The relay station is substituted by a simple flipflop; the shells do not have to elaborate valid and stop signals; the routing is greatly simplified since nothing else but the signals of the original system have to be accounted for (no valid's and stop's).

To complete the picture of the static scheduling methodology, we need a way to compute a valid schedule, given the netlist. Such schedule must guarantee *a priori* safety conditions which are ensured by explicit Latency Insensitive Protocols. As a performance requirement, furthermore, the schedule should ensure an average throughput equal to the maximum attainable with the given latency constraints (interconnect-related memory elements). For these reasons we looked for a systematic method to generate valid (in the sense forementioned) schedules by analyzing the graph (*lis-graph* in Carloni's definition [4]) describing the block connectivity. An extension of the method described by Boyer *et alii* in [33], can be adapted for the issue here at stake. The details are contained in [6]. Here we summarize the basic idea: The netlist is conveniently represented as a graph and labelled accordingly; a max cost to ratio problem is solved on the labelled graph (for example by using Lawvere's method [34]). The solution, if it exists, gives a valid scheduling for the netlist. As a by-product, we obtain the maximum throughput of the system.

As in the original protocol, feedforward reconvergent fanouts can be "cured" by adding appropriate number of FFs. The same problem of reconvergent fanouts is possible inside loops. In this case, though, its solution might turn to be more complicated. In fact, contrary to the feedforward case, it is not possible in general to add flip flops on the faster branch in order to equalize paths, because this might adversely affect the performance on the loop where branches are added. The deep reason of it is that differences in schedules are not necessarily integer numbers (from the above discussion it should be clear that schedules are inherently pseudo-periodic rather than periodic), while insertion of a flip flop always introduces a fixed integer delay. It is possible that the delay between branches does not adversely affect synchronization (thanks to clock gating in specific time), or that the delays are integers (so that pure flip flops or FIFOs are sufficient) but in the most general case a device which is capable of a perfect resynchronization is needed. A periodic schedule (synchronized with the general schedule of the system though different in general) introduces selectively a delay only when needed, by choosing the latched branch of the mux's input.

8. Physical Design Issues

The LIPs work at system level because they guarantee functionality whatever the topology of blocks and delay elements used to pipeline interconnects. As we have seen however, the addition of memory elements comes at the cost of reducing throughput, mainly because of the presence of feedback topologies in the blocks netlist. It is then clear that a suited physical design strategy aiming at reducing the throughput reduction consequence of the flipflop addition is highly desirable. We propose a throughput driven floorplanning where throughput enters in the cost function of a simulated annealing based tool.

It is important to make an observation: The algorithm that uses Lawvere's method (or an equivalent one as that described in [21]) is sufficiently efficient to allow a performance evaluation, but still way too slow to be included in a loop of an iterative method of optimization as one needed in floorplanning. We therefore looked for an heuristic which could approximate the exact throughput cost.

A function that could be easily integrated in a simulated annealing context can be computed as follows:

1. Before entering the annealing iteration, we statically evaluate a weight for each pin to pin net: The inverse of the shortest loop the net belongs to.
2. At each iteration we consider each pin to pin connection and, based on the current position of the blocks and the relative positions of the pins we evaluate the Manhattan distance between the pins.
3. The distance is divided by the maximum length admissible between clocked elements, and the integer part of the result is taken.
4. This last number is multiplied by the weight computed in the first point.
5. All such values are summed.

The loop computation is performed through a simple breadth first search starting from the destination of the edge in question.

Besides, being outside the loop, it represents a small fraction of CPU time. Our experiments showed a pretty high correlation between this cost and the exact throughput (see [21] for more details).

In order to test our optimization technique and gather some statistics on the features of this problem, we implemented the algorithms described in previous sections integrating it in an existing publicly available simulated annealing floorplanner based on the sequence pair representation: PARQUET (see [30]).

The complete results we gathered on a series of benchmarks are reported in the original paper [21]. Here we would like to point out a series of conclusions:
- Throughput optimization really achieves better results in terms of throughput. The gain w.r.t. wirelength minimization is in average 11%, while the gain w.r.t. area is 25% again in average over all the experiments. If we consider only the gain in the case of the longest critical length, though, the two gains are 24% and 64% respectively, thus suggesting the existence of a threshold length about which the gain becomes substantial.
- Wirelength and throughput minimization are goals which are not so correlated as it might be thought.
- Different benchmarks behave differently as long as throughput is concerned, and their difficulty is not a simple measure of other features, let alone their complexity (number of blocks, number of nets).
- The task is in and by itself inherently difficult: there is no chance of getting an acceptable throughput with area optimization, while also wirelength minimization can lead to highly suboptimal results. - Even if it depends on the benchmark considered, there is a value for critical dimension such that below it, no matter what kind of cost function, the throughput cannot be optimized, while above that threshold the three methods differ.

9. Case Study: MPEG

In order to show the effectiveness of our approach we decided to resort to a benchmark that have been already used in the context of Latency Insensitive Protocols [4]. The schematic in figure 6 represents the functional blocks of a Mpeg2 encoder. The small blocks along interconnects represent pipeline flipflops. Their position and number is obtained after a physical design step which produces the floorplan reported in figure 7. In figure 6 the critical loop of the entire system is highlighted with a dashed line. This throughput is 4/6=2/3 because the functional modules along the loop are 4 and there are 2 flipflops (4/(4+2)). As a consequence the scheduling period is 3 clock cycles and there will be two "valid" cycles and one "stop" cycle. Every "shell" will be provided with a 3 bits long shift register. Some additional delays had to be inserted in various points to equalize reconvergent paths as explained in section 4.2. For instance, a flipflop has been added to the connection between Frame Memory and Motion Compensation blocks.

After obtaining the netlist of figure 6 we applied an implementation of the scheduler described in section 4 in order to identify valid schedules and path unbalance.

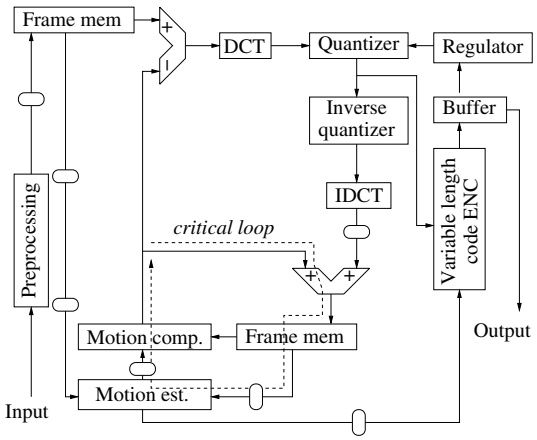

Figure 6. MPEG block diagram with pipelining memory elements.

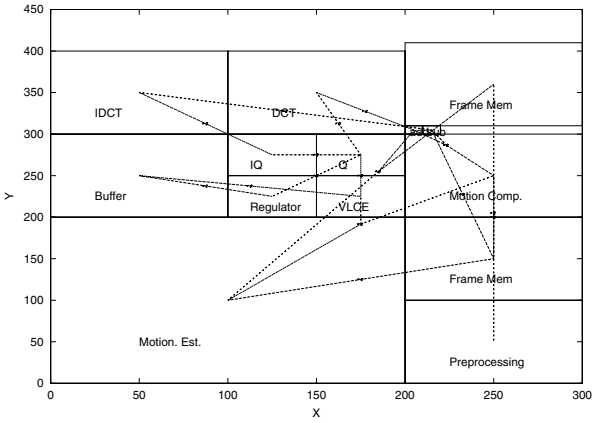

Figure 7. MPEG Floorplanning Example.

The simulation (not reported) shows that 2 new data are produced every 3 clock ticks. The first waveform is the output of Frame Memory and the following are in the same order as the blocks in the loop. The progression of the "stalled" functional module is evident and it is represented by a stretching of the last valid data. We also verified that the fundamental constraints as of section 4 on system safety (no data loss) were satisfied.

10. Conclusions and Future Work

The fact of wire pipelining is likely to hamper the evolution of high-speed designs from the near future on (following the projected trend discussed in section 3): An appropriate design methodology will relieve designers from its nitty-gritty details. The solution outlined here is but one of the possible in the spectrum summarized in section 2. Careful comparison will be needed to assess which one is the best compromise between intrusiveness in the design flow, performance, area and power penalty.

Our vision is that some elements of insight about the design functionality will be the key to break in certain limitations of communication intensive designs.

References

[1] K. Keutzer *et al.*, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design" IEEE Trans. CAD, Vol. 19, No. 12, Dec. 2000.

[2] The international technology roadmap for semiconductors, 2001, SIA.

[3] L.P. Carloni *et alii*, A Methodology for "Correct-by-Construction" Latency Insensitive Design", Proc. ICCAD 99.

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

[5] L.P. Carloni, K.L. McMillan and A.L. Sangiovanni-Vincentelli, Theory of Latency-Insensitive Design, IEEE TCAD, vol. 20, No. 9, Sept. 2001.

[6] M. R. Casu and L. Macchiarulo, "A New Approach to Latency Insensitive Design," *Proc. DAC 04,* June 2004, San Diego CA.

[7] M. Singh and M. Theobald, Generalized Latency Insensitive Systems for Single-Clock and Multi-Clock Architectures, *Proc. DATE 2004*, Paris.

[8] Tiberiu Chelcea and Steven M. Nowick, Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols, *Proc. DAC 01*.

[9] J. Cong and S.K. Lim, "Physical Planning with Retiming," Proc. ICCAD 2000.

[10] R. Lu and C.-K. Koh, "Interconnect Planning with Local Area Constrained Retiming," Proc. DATE 2003.

[11] C. Chu, E.F.Y. Young, D.K.Y. Tong, and S. Dechu, "Retiming with interconnect and gate delay", *Proc. ICCAD 2003*.

[12] C. Lin and H. Zhou, "Retiming for wire pipelining in system-on-chip," in Proc. ICCAD 2003.

[13] L.P. Carloni and A.L. Sangiovanni-Vincentelli, "Combining retiming and recycling to optimize the performance of synchronous circuits Carloni," *Proc. SBCCI03.*

[14] J.Cong *et al.*, "Architecture-Level Synthesis for Automatic Interconnect Pipelining,", *Proc. DAC 04,* June 2004, San Diego CA.

[15] V. Sachin and S. Sapatnekar, "A Method for Correcting the Functionality of a Wire-Pipelined Circuit," *Proc DAC 04,* June 2004, San Diego CA.

[16] R. Lu *et al.* , "Flip-Flop and Repeater Insertion for Early Interconnect Planning," Proc. DATE 2002.

[17] S. Hassoun *et al.* , "Optimal Buffered Routing Path Constructions for Single and Multiple Clock Domain Systems," Proc. ICCAD 2002.

[18] P. Cocchini, "Concurrent Flip-Flop and Repeater Insertion for High Performance Integrated Circuits," Proc. ICCAD 2002.

[19] —-, "A Methodology for Optimal Repeater Insertion in Pipelined Interconnects," *IEEE TCAD,*, Vol. 32, No. 12, Dec. 2003.

[20] D.K. Tong and E.F.Y. Young, "Performance-driven Register Insertion in Placement," *Proc. ISPD,* Phoenix, AZ, Apr. 2004.

[21] M.R. Casu and L. Macchiarulo, "Floorplanning for Throughput," Proc. ISPD 04, Phoenix, AZ, April 2004.

[22] M. Ekpanyapong *et al.*, "Profile-Guided Microarchitectural Floorplanning for Deep Submicron Processor Design," *Proc. DAC 04,* June 2004, San Diego CA.

[23] C. Long *et al.*, "Floorplanning Optimization with Trajectory Piecewise-Linear Model for Pipelined Interconnects," *Proc. DAC 04,* June 2004, San Diego CA.

[24] http://vlsicad.cs.ucla.edu/GSRC/GTX/

[25] http://www.eecs.umich.edu/~dennis/bacpac/

[26] Y.I. Ismail and E.G. Friedman, "Effects of Inductance on the Propagation Delay and Repeater Insertion in VLSI Circuits," *IEEE TVLSI,* vol. 8, no. 2, April 2000.

[27] C. Alpert, "The ISPD98 Circuit Benchmark Suite," in Proc. ISPD'98.

[28] J. Dambre *et al.*, "Toward the Accurate Prediction of Placement Wire Length Distributions in VLSI Circuits," in IEEE TVLSI, No. 4, 2004.

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

[30] http://vlsicad.eecs.umich.edu/BK/parquet/

[31] M.R. Casu and L. Macchiarulo, "A Detailed Implementation of Latency Insensitive Protocols," Proc. FMGALS 2003, Pisa, Italy, Sep. 2003.

[32] M.R. Casu and L. Macchiarulo, "Issues in Implementing Latency Insensitive Protocols," Proc. DATE 04, Paris, March 2004.

[33] Francois R. Boyer, *et alii*, "Optimal design of synchronous circuits using software pipelining techniques," ACM TODAES, vol. 6, n. 4, 2001.

[34] Eugene Lawvere, *Combinatorial Optimization: Networks and Matroids*, New York, Chicago, Holt Rinehart and Winston Ed., 1976.