Floorplan Assisted Data Rate Enhancement through Wire Pipelining: A Real Assessment

Mario R. Casu
Politecnico di Torino
Dipartimento di Elettronica
C.so Duca degli Abruzzi, 24
I-10129, Torino, ITALY
mario.casu@polito.it

Luca Macchiarulo
University of Hawaii at Manoa
Department of Electrical Engineering
Holmes Hall 437
Honolulu, HI 96822, USA
lucam@hawaii.edu

ABSTRACT

The recent shift towards wire pipelining (WP) mandated by technological factors has attracted attention towards latency-controlled floorplanning. However, no systematic study has been published so far that takes into account block and logic delay limitations. The present work aims at filling the gap by showing that block delay can limit and possibly prevent any real gain WP might promise. Recurring to adaptive WP schemes, on the other hand, allows relevant gains. We built a floorplanner that optimizes for maximum data rate, taking into account various models of block delay, and compares them to the optimal results obtained when no wire pipelining is employed. Experiments with suitable floorplanning benchmarks and case studies are performed to substantiate theoretical intuitions.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*

General Terms

Algorithms

Keywords

Systems-on-chip, Floorplanning, Wire pipelining, Throughput

1. INTRODUCTION

Wire pipelining (WP) is currently used in high-speed design to break the discrepancy between increasing speed of logic blocks and global interconnect delays [1]. This new technique has attracted attention towards latency-controlled floorplanning (see [2], [3] and [4] for example). However, the latency added in wires and the delay of logic and interconnects within the blocks may jeopardize the frequency increase guaranteed by this technique by reducing the actual data rate. To the authors' knowledge, no systematic study has been published so far that takes into account block delay and logic delay limitations, as well as more critical electrical issues related to frequency increase. With this paper we aim at filling the gap by showing that block delay can limit and possibly prevent any real high performance gain the WP technique, if implemented in a too conservative way, might promise (as measured by data rate increase, rather than misleading clock period reduction). 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 which optimizes the data rate of systems 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 wire pipelining is employed.

In section 2 we briefly recall the characteristics of a standard WP technique. Its limits are highlighted through a mathematical derivation in section 3. Section 4 reports some results of performance achievable from our floorplanning strategy employing this WP methodology with and without limitations imposed by the blocks delays. The foundations of the alternative WP methodology are described in sections 5 and 6, while the results of its application using a modified floorplan strategy are outlined in section 7. Finally we summarize the work achievements in the conclusions.

2. THE PROMISE OF WIRE PIPELINING

Interconnect and gate delays scale differently such that we observe an increasing gap between always faster logic and slower wires [1]. The very high clock frequencies allowed by always smaller gate delays cannot be supported by long global wires spanning the entire die. Wire buffering alone cannot solve the problem. Clocked buffers can be then used to segment wires. This technique is called *wire pipelining* (WP) and allows connections to run at a very high frequency at the cost of increased *wire latency*. From a theoretical point of view, if we abstract for a moment from technology and layout limitations, it could be possible to run wires at any desired frequency by simply increasing the number of clocked buffers. This is what we call *the promise* *of wire pipelining.* Now we are allowed to ask if it is always worth to increase the wire frequency at the cost of increased latency. Since in general the answer is very dependent on the kind of system to which we apply WP, in order to give a complete answer, we restrict our analysis to the case (as that of Systems-on-Chip) in which designers connect intellectual properties (IP) taken from libraries and are not allowed to modify their internal structure (we believe most of the conclusions can be easily extended to other design cases). Moreover, who designed the IP did not know the environment in which the SOC designer will insert it and could not predict *a priori* the amount of latency added to its I/O connections. In this type of systems, the designer would like to resort to WP, if needed, in a *transparent way*, i.e. without having to change the blocks he has to connect. In order to avoid an unpredictable behavior due to the unpredictable (before the floorplan) wire latency, he has to implement a *communication protocol* which aims at making the latent interconnect system transparent to the computational blocks. An example of this type of protocols are the so-called Latency Insensitive Protocols (LIP) that, by means of special wrappers around the blocks and pipeline elements along the wires, make a system with added latency functionally equivalent to a system with zero-latency and allow a higher clock frequency [5]. The equivalence is obtained by making the blocks "patient" – i.e. suspending their operation by suitably gating their clock – when the input data are not all ready to be processed because of the latency.

While this technique allows to fulfill the promise of wire pipelining, it is not always true that a frequency increase corresponds to a higher "throughput" (TH). Since the blocks are made patient, from time to time they are suspended from operating and, as a result, *the number of actual computations per clock cycle* is in general < 1.0. This is what we call *throughput reduction.* However, *througphut* and *frequency* alone are not meaningful. What really counts is the *data rate* (DR) that is the product of throughput and frequency. A system without wire latency has TH=1 so that frequency and data-rate have the same meaning. In the LIP case, this assumption does not hold anymore.

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, TH should not be degraded more than $1/N\times$. It can be shown that the throughput limitation of systems with latency insensitiveness comes from the netlist loops [6]. A loop of m blocks with a total of n sequential delays (flipflops) along the loop has the following throughput

$$TH = \frac{m}{m+n} = \frac{m}{m+\sum_1^m n_i} \tag{1}$$

where n_i is the number of delays added to the $i-$th branch of the worst loop. It can also be shown that the system throughput is bounded by the *worst loop*, that is the netlist loop having the minimum ratio $m/(m+n)$ (see, for example, [6]). 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, and convenient, 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 [2]. However, the question about whether an optimal floorplan is beneficial or not for the true data rate, not only for the throughput, is still open. In the following section, we derive a mathematical framework that helps understanding the requirements that a physical design step like floorplanning should fulfill in order to improve the data rate using wire pipelines.

3. MATHEMATICAL FRAMEWORK

The number of delays n_i in a single branch of the worst loop that forces the system throughput is given by

$$n_i = \lfloor l_i / l_{max} \rfloor \tag{2}$$

where l_i is the branch length, computed as Manhattan distance between two pins of two blocks connected by the branch, and l_{max} is the maximum distance allowed between two flipflops so as to respect a frequency constraint F_{max}. If a line of length l is optimally buffered, its delay becomes proportional to l. We normalize to 1 (here and in the rest of the paper) this proportionality coefficient in the following derivation so that "delay" and "length" become synonymous. Therefore frequency F_{max} is simply given by $1/l_{max}$. The data-rate is given by

$$DR = \frac{m}{m+\sum_1^m \lfloor \frac{l_i}{l_{max}} \rfloor} \cdot \frac{1}{l_{max}}. \tag{3}$$

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

$$DR_{UB} = \frac{m}{m+\sum_1^m (\frac{l_i}{l_{max}}-1)} \cdot \frac{1}{l_{max}} = \frac{m}{\sum_1^m l_i} = \frac{1}{l_m}, \tag{4}$$

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

$$DR_{LB} = \frac{m}{m+\sum_1^m \frac{l_i}{l_{max}}} \cdot \frac{1}{l_{max}} = \frac{1}{l_{max}+l_m}, \tag{5}$$

The data rate of a system without wire pipelining DR_0 is simply given by

$$DR_0 = TH_0 \cdot F_0 = 1.0 \cdot \frac{1}{L} \tag{6}$$

where $\mathrm{TH}_0 = 1$ is the unitary throughput for a system without WP and F_0 is the maximum allowed frequency which is limited by L, the longest interconnect delay in the system. Using WP for this kind of systems is convenient only if the speed-up, i.e. the ratio DR/DR_0, is significantly higher than 1.0, in order to compensate for the increasing problems of high-frequency clocks and related power dissipation and timing issues. The speed-up SU lays between two bounds

$$\frac{L}{l_m + l_{max}} \le SU \le \frac{L}{l_m} \tag{7}$$

Interestingly, increasing the frequency to ∞ in the wire pipelining system, and so augmenting indefinitely the wire pipe depth, reduces l_{max} to 0 but cannot further increase the speed-up over $\frac{L}{l_m}$ 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 maximized. If $l_m \ge L$, whatever the number of WP stages and so whatever the frequency used, the solution without WP, i.e. a general slowdown to the maximum allowed frequency $1/L$, would be a better choice.

3.1 Effect of block delay

In our previous derivation we supposed the blocks ideal and attributed all frequency limits to the interconnects. We suppose now that the i-th block is characterized by an input delay d_i, due to both logic gates and interconnects *within* the block, and derive the corresponding system data rates with and without wire pipelining.

For a zero-latency system, the maximum data rate becomes

$$DR_0 = 1.0 \cdot F_{max} = \frac{1}{\max_i(L_i + d_i)} \tag{8}$$

where, in general, $\max_i(L_i + d_i)$ is different from $L + d_{max}$ where L was introduced before and d_{max} is the maximum delay

$$d_{max} = \max_i d_i. \tag{9}$$

With WP, the maximum frequency is now given by

$$F_{max} = \frac{1}{\max(l_{max}, d_{max})} \tag{10}$$

while the throughput is

$$TH = \frac{m}{m + \sum_1^m \lfloor \frac{l_i + d_i}{\max(l_{max}, d_{max})} \rfloor}. \tag{11}$$

Finally, after a number of easy algebraic steps, we can recompute upper and lower bounds for the data rate as follows

$$\frac{1}{l_m + d_m + \max(l_{max}, d_{max})} \le DR \le \frac{1}{l_m + d_m} \tag{12}$$

where d_m is the average delay in the worst loop of m blocks

$$d_m = \frac{1}{m} \sum_{i=1}^{m} d_i \tag{13}$$

Consequently, upper and lower bounds to speed-up SU are

$$\frac{\max_i(L_i + d_i)}{l_m + d_m + \max(l_{max}, d_{max})} \le SU \le \frac{\max_i(L_i + d_i)}{l_m + d_m} \tag{14}$$

The deeper the wire pipelining, the faster is the interconnect, but, due to the slowest block, the frequency limit is $1/d_{max}$. Therefore, even when we reduce the WP flipflop distance to $l_{max} = 0$, the maximum speed-up is bounded by

$$\frac{\max_i(L_i + d_i)}{l_m + d_m + d_{max}} \le SU \le \frac{\max_i(L_i + d_i)}{l_m + d_m}. \tag{15}$$

In the general case, nothing can be said about whether the WP system with delays is worse than the previous ideal case, since it all depends on the delay values d_i. However, it is highly "probable" that the real case is worse. Suppose for instance that all delays are equal to d_{max} and that the longest interconnect is also the average length $L = l_m$. In the ideal case the speed-up is 1.0 while in the real case can be as low as $(L + d_{max})/(L + 2d_{max}) < 1.0$.

In the next section, we present some DR results obtained using a throughput-driven floorplanner [2] with and without considering the blocks' delays.

4. TH-DRIVEN FLOORPLAN

We now report some results obtained running the throughput-driven floorplanner [2] over some of the GSRC and MCNC benchmarks. We collected the results obtained for the non-pipelined systems by trying to identify (through successive floorplans) the maximum frequency for which a non-pipelined floorplan (t.i. a floorplan whose pin to pin length were bounded by l_{max}. For the WP system, we repeated the experiment allowing pipelining and identifying the maximum data rate, for higher frequency values. The results in table 1 were obtained in the ideal case of **no block delay**. The lengths are expressed in percentage of the die edge length, which is computed as the square root of the sum of the blocks' area. 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 equation 7. For the benchmarks considered, the values, in percentage of the die edge, are also reported in table 1. They can be suitably compared to the maximum length L of the case without wire pipelining for a better understanding of the speed-up figure.

Table 1: GSRC and MCNC benchmarks: ideal case with no block's delay.

bench.	DR	$1 - TH$	DR_0	$L(\%)$	$SU(\%)$	$l_m(\%)$
n10	0.926	0.944	0.885	113	4.6	108
n30	0.709	0.986	0.645	155	10	141
n50	0.877	0.965	0.637	157	38	114
n100	0.847	0.932	0.653	153	30	118
apte	0.700	0.993	0.694	144	0.8	143
xerox	0.658	0.974	0.510	196	27	154
hp	0.610	0.988	0.562	178	9	164
ami33	1.087	0.989	1.031	97	5	92
ami49	1.042	0.958	0.769	130	36	96

DR=data rate with WP, TH= throughput with WP, DR_0=data rate w/out WP, L=maximum unlatched length (as a percentage of die area), SU=speedup of WP, l_m=average length.

We normalized the die edge to 17 mm, according to the ITRS'03 estimations for the high-performance dies in years '03-'09. As a result, **all benchmarks have the same total area,** regardless the blocks' number and size. It is interesting to notice that the speed-up is negligible for the benchmarks with less blocks, and instead significant when the blocks are numerous and smaller (n50, n100 and ami49).

Should a different area normalization be used, the speed-up factor of the ideal case without delay would not be affected because both L, l_m and l_{max} would be equally scaled and their ratio be unchanged.

4.1 Effect of block delay

Let us now see what happens when the block delay is taken into account. We considered two extreme approaches. 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. The physical meaning behind that choice is that of attributing the delay to internal (buffered) interconnect whose typical length is linearly related to the block edge.

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 13 FO4. This is the value used in the ITRS for computing the estimated clock frequency of high-performance microprocessors. We took the values of logic and interconnect delay from the roadmap in order to define a proportionality factor between the twos. Again we supposed the wire delays linear with wire length, i.e. the case of buffered wires.

Tables 2 and 3 report the results for these two choices. From table 2 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, the MCNC suite experiences a degradation – except for apte –.

Table 2: Block delay proportional to block's edge

bench	DR	$1-TH$	l_{max}	DR_0	L	SU
n10	0.463	0.78	48%	0.500	200%	-7.0%
n30	0.625	0.8	32%	0.585	171%	+7.0%
n50	0.833	0.75	30%	0.578	173%	+44%
n100	0.868	0.72	32%	0.625	160%	+39%
apte	0.496	0.722	56%	0.448	223%	+11.0%
hp	0.521	0.667	64%	0.529	189%	-2.0%
xerox	0.463	0.778	48%	0.467	214%	-1.0%
ami33	0.833	0.75	31%	0.840	119%	-1.0%
ami49	0.595	0.67	56%	0.610	164%	-2.0%

DR=data rate with WP, TH= throughput with WP, l_{max}=max unlatched length for WP, DR_0=data rate w/out WP, L=maximum unlatched length w/out WP, SU=speedup of WP.

Table 3: Block delay equal to 13 FO4

bench	DR	$1-TH$	l_{max}	DR_0	L	SU
n10	0.521	0.500	96%	0.588	172%	-11.4%
n30	0.446	0.570	96%	0.398	251%	+12.0%
n50	0.446	0.500	112%	0.452	221%	-2%
n100	0.473	0.545	96%	0.435	230%	+8%
apte	0.446	0.5	112%	0.465	215%	-4.0%
hp	0.417	0.67	80%	0.476	210%	-12.4%
xerox	0.391	0.0	256%	0.395	253%	-1.0%
ami33	0.521	0.5	96%	0.529	189%	-2.0%
ami49	0.521	0.5	96%	0.535	187%	-3.0%

DR=data rate with WP, TH= throughput with WP, l_{max}=max unlatched length for WP, DR_0=data rate w/out WP, L=maximum unlatched length w/out WP, SU=speedup of WP.

From table 3 we observe that the impact of a constant block delay is disastrous and there is no more evidence of the effectiveness of using wire pipes. In practice, it seems impossible to obtain a significant advantage w.r.t. the traditional solution (only n30 and n100 show moderate speedups of 12% for a frequency which is around 2.5 times the original!)

What we observe is a trend toward a generalized reduction of the speed-up when the delay becomes more and more 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 data rate 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 speed-up is in general > 1 but the amount of actual data-rate increase depends on the circuit geometric and netlist characteristics. For higher delay values, obtaining a significant speed-up may be a hopeless task.

Until now, we have analyzed a class of systems using WP in a totally transparent way. Blocks are encapsulated within wrappers that make them patient when latent signals have not arrived yet. This is the very reason of the throughput reduction expressed by equation (1) and in the end of the speed-up results reported above. We can now ask how we can break this very strict constraint. In the next section we introduce a new class of WP systems where a modification to the protocol breaks this TH limitation.

5. EXPLOITING THE LOCALITY

The underlying assumption of our previous derivation of the mathematical background for the data rate computation and the consequent results obtained after floorplanning is that *every block reads all its inputs in every clock cycle.* Therefore, when one or more of its inputs are not ready yet because of the added latency, the clock is gated and the block's operation suspended. This is surely the most conservative and also the easiest "plug and play" approach because the wrappers around the blocks, which make then compliant with a WP protocol, do not have to know the actual behavior of the blocks themselves. However it is likely that the computation within the blocks has some degree of *locality*, or, in other words, there certainly exist some 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 to a 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 equation (1) as noticed in [7]. If we are able to build a wrapper that is aware of the states where the block executes a local computation or ignores some of its inputs, we are also able to increase the throughput. From a macroscopic perspective, everything works like if we were able to *open* the loops from time to time. However, this cannot be true for the whole operating time – referring to the previous example, the microprocessor accesses to the L2 memory and reads data from time to time – and in general we can express the average throughput of a loop of m blocks as such

$$TH = \alpha \cdot \frac{m}{m+n} + (1-\alpha) \cdot 1 \qquad (16)$$

where α is the fraction of operating time when the loop is *active*. For $\alpha = 1$, Th is the same of (1) while for $\alpha < 1$ is higher. 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 data rate rather than pure clock cycle and/or throughput).

It is possible to show, though not in the scope of this paper, that a modified wire pipelining 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. Even if the details of the scheme are immaterial, we will use the model to validate the cost schemes that we will present in next section, and the final case study of the paper.

In a system, to be general, there will be blocks for which it is possible to exploit this information and build an *ad hoc* wrapper and some for which, for lack of knowledge or because of intrinsic behavior, it is not allowed. If we are able to exploit this information as much as possible within a floorplanning environment, we believe that the WP system will experience a higher speed-up with respect to the zero-latency system than in the previous examples. Of course, this improvement does not come for free because the wrappers are more complex and have to interact with the intellectual properties to a larger extent.

The next section details the modification we made to the floorplan framework in order to take these facts into account.

6. DYNAMICALLY ADAPTIVE COMMUNICATIONS

If you take the case, hinted at in the previous section, of a CPU to cache communication system, it is clear that the overall performance (measured by valid data that are processed or output by the system in a time unit) depends on the details of the communication between the two blocks: the fraction of time in which the CPU is writing (as opposed to the time in which it is reading) from the memory dynamically dissolves (for the relative period) a limiting cycle in the system. In order to take advantage from this situation, we therefore must have a minimal information about the statistical properties of the signals involved. Even if a full profile information would be necessary to completely exploit such an advantage, as it is made clear by the huge profiling costs of processor-based floorplanning system [3] a solution that makes minimal use of detailed data simulations is welcome. We will 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 wire pipelined system beyond the theoretical limits described and analyzed in the previous sections. We will proceed to deduce some basic cost functions in this section.

Formula (16) relates the final throughput of the system to the single loop's timing behavior (the constant α represents the fraction in which the CPU is reading data, in terms of CPU active periods), while we are interested in a measure related to the real timing of the system. 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 consequences of such an assumption are that some information about the context switching need to be available to every related block of the system that uses the signals: Even if practical communication protocols don't allow an instantaneous dispatching of such information in the general case, it can be shown that certain typical situations in real systems approximate it closely: In particular, 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 back it up with experimental data in the following sections.

Under the assumption, the system in each moment will work at a maximum throughput given by **the most critical active loop present**. This means the most critical loop whose branches are all active at the same time. Let's suppose we are analyzing an entire session of the system behavior: We can, at least in principle, identify periods of time in which the system's throughput is dominated by a single loop, that is, its performance is limited by a certain fraction $m/(m+n)$, where all the connections between the blocks in the loop are active at the same time. In the case where all the communication channels are active, this value is equal to the static throughput computation outlined in [2], while in general, due to the presence of idle communication, such throughput will be larger. In order to quantify the performance of the system, let's suppose that W_i represents the *logical* period of time in which the loop l_i, with associated throughput T_i, dominates the system's throughput. Then, the *physical* period of interval i is given by the formula W_i/T_i, because each data calculation in such conditions implies a number of clock periods equal to $1/T_i$. So, the overall *physical* time spent by the system to complete its computations is equal to $\sum_i W_i/T_i$, while the number of logical time steps is just $\sum_i W_i$. This gives a value for the overall throughput of the computation of $\frac{\sum_i W_i}{\sum_i W_i/T_i}$. This can be written as $\frac{1}{\sum_i w_i/T_i}$, where w_i is the time fraction in which the most critical throughput is T_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 for two reasons:

- the number of loops is potentially exponential in the graph dimension
- each loops' activation depends on the activation of each of its channels

In order to render the analysis of such a system manageable, we propose to define a quantity which defines each channel's activity independently of the rest of the system: the **channel activation ratio**. Such quantity represents the 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. 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 7.

Under the independence assumption, together with assumption 1 that states that communication always occurs in bursts, 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 act_{ji}, that is $w_i = \Pi_j act_{ji}$. This property is valid also under the less restrictive set of assumptions that allow for non-bursty communication.

Therefore, we can compute an average throughput as:

$$TH_{tot} = \frac{1}{\sum_i \frac{\Pi_j act_{ji}}{T_i}}$$

Such computation depends on simple static profiling information on the single communication channels:

The formulation is clear, but it allows the computation of the overall throughput only by analyzing an exponential number of loops. If this were the case, no advantage could be drawn from the last assumption. However, it is possible to see that the same scheme used in [2] could be extended to such a computation, with minimal added complexity.

Differently from [2], though, such an algorithm is only useful to obtain the throughput contribution of the worst loop, rather than the complete throughput, and has therefore the rather limited application of giving an upper bound for the systems' throughput.

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 [2], modified in order to include the effect of channel activity. In detail:

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.
3. This last number is divided by the length of the smallest loop to which the connection belong.
4. **The number is multiplied by the activity of the channel**.
5. All such values are summed.

The algorithm differs substantially from the old one only by the presence of the additional weighting step that takes into account the channel communication properties summarized in the activity value. The rationale of such a choice is that a channel will be contributing to the overall throughput degradation of the system at most up to its activation time. Other minor modifications have been introduced in order to take into account the blocks' delays.

7. RESULTS AND COMPARISONS

In this section we will try to assess the pros and cons of wire pipelining and illustrate the advantage derived by employing an adaptive scheme. Such advantage, however, will have to be tested against the greater complexity of design and verification, faster clock distribution, process variations, which have not entered in the discussion. We believe that the importance of the issue justifies our approach, especially in light of the promising results shown.

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 wire pipelining can provide acceptable advantages under the most adverse model of delay (13 FO4) discussed previously; then, we analyzed the performance of two well-defined systems, an MPEG decoder and a pipelined CPU, and used real profiling information to simulate the systems.

7.1 GSRC benchmarks

The experiments described in this subsection use the same benchmarks of section 4, 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 (at the point that even the all important directionality of the pins had to be guessed or arbitrarily assigned), 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). The next subsection will show how the basic conclusions hold for more general cases of temporally related channels.

After generating (in a suitable format) the "new" benchmarks, we proceeded to compute two values: an optimized non-pipelined solution and an adaptive wire pipelined one. The first is generated in a similar way to section 4. As we were interested in the most critical case of fixed block delay of 13 FO4 (which proved to be untractable - at least for the technological parameters we considered - by the non-adaptive pipelining), all the results shown here refer to such a case. The adaptive solution is generated by employing a floorplanner which optimizes the heuristic cost function described in previous section (we modified the publicly available tool PARQUET [8] in such a way that loop computation and channel activity annotation is 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 bursty 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 is compared to that of the non-adaptive design. Various choices for the systems' frequencies are chosen and simulated in order to obtain a good approximation of the optimum. However, due to the impossibility of simulating a large number of different solutions, we used a binary procedure to find a suitable frequency before running the simulations. Such procedure is guided by the most critical loop cost described in the previous section. The results are shown in table 4. Please note that here and in the following tables the data rate are the result of a simulation of the VHDL system and not estimates from the floorplanner, thus representing real improvements w.r.t. the non pipelined case. The previous table could give a precise estimation because of the static limitations to performance.

Table 4: Block delay equal to 13 FO4, adaptive case

bench	DR	TH	l_{max}	DR_0	L	SU
n10	0.831	0.5820	70%	0.588	172%	29%
n30	0.718	0.4668	65%	0.398	251%	44%
n50	0.722	0.4693	65%	0.452	221%	37%
n100	0.713	0.4636	65%	0.435	230%	39%
ami33	0.63	0.4125	65%	0.529	189%	16%
xerox	0.47	0.3055	65%	0.395	253%	16%

DR=data rate with WP, TH= throughput with WP, l_{max}=max unlatched length for WP, DR_0=data rate w/out WP, L=maximum unlatched length w/out WP, SU=speedup of WP.

As the table shows, but for the case of the benchmark xerox, use of adaptive schemes is always beneficial in data rate, while the corresponding table 3 showed the extreme disadvantage of the non-adaptive scheme with respect to a mainstream high performance design. This has to be weighted against the increase in frequency that, as it is apparent from the ratios of lengths (remember that in our units, lengths are proportionally correlated to delays, in this case to the clock period). While this could, in some cases, be useful to adapt the frequency of the designed block to other high-frequency part of the system, it does not come for free: Clock tree synthesis and tightened skew control might be impossible to achieve. For this reason, it is interesting to see the behavior of the suboptimal solutions, that is the solutions for different (and smaller) target frequencies. Figure 1 shows the trend for the case of benchmark n100. It is apparent that there is space for optimization between the optimal case (length=65%) and the maximum non-pipelined case, and the designer can choose the appropriate trade-off between frequency and data-rate. Another interesting result that can be shown in the figure (but is apparent from the figures in table 4) is that the maximum throughput is reached at the limit length of 65% that corresponds to the inverse of 13 FO4 delay. Were it possible to decrease such a delay, the gains could be even greater. In fact, as shown before, the 13 FO4 delay model is the most stringent in terms of performance. Looking at the case shown in figure 2 helps understanding the difference. In this case, n10 has been optimized considering null delay blocks (or, equivalently, supposing the signals are latched both at the input and at the output of a block). The trend towards better data rates for larger frequencies is apparent.

7.2 Case Study: MPEG

In this subsection, we will try to validate the methodology with a real-life example that is the typical (though small) example for which heavy wire pipelining might be appropriate: an MPEG decoder. We follow in this example the implementation of the decoder described in [9] and used as case study also in [6] for the case of latency insensitive protocols, where it is possible to understand the main quality of the case that makes it suitable for our purposes: the various blocks of the system, communicate with each other in bursts followed by channel idle periods. It is also possible to quantify exactly the duration of the activity periods, thus formulating a floorplanning problem which is completely compliant with our description as of section 4. The only difference is that the system enforces a strict data communication dependency that contradicts the randomness hypothesis: However, the experimental results show that the optimization potential of a pipelined solution is substantial.

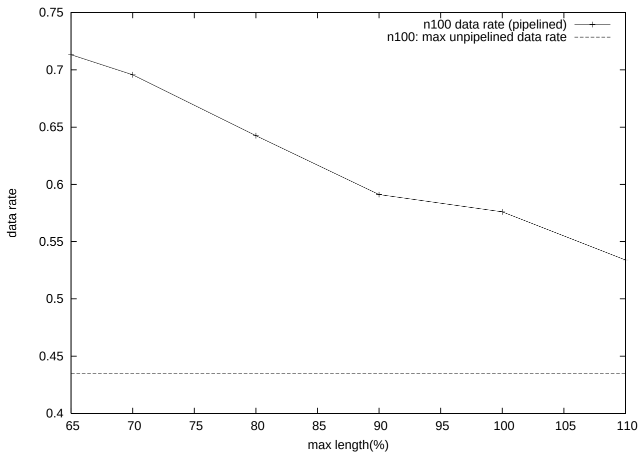

Figure 1: n100 : Variation of data rate with frequency

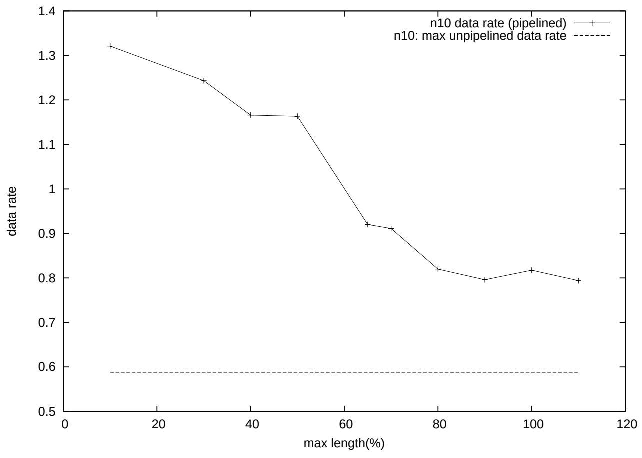

Figure 2: n10 : Optimization of data rate in the ideal case (no block delay)

We first described the system in standard format, together with channel annotations with activities extracted from the real behavior. Then, as a first phase, we optimized the system as detailed for the other benchmarks. This gave us a series of solutions detailed in figure 3: the block 13 FO4 optimization potential is around 42%, while for the unbound optimization (at the same frequency) we reach 76% (this occurs in the graph at length 65%; the graph shows a much larger speedup at very high frequencies, but such speedup would presuppose an higher block frequency than the one fixed by 13FO4).

In a second phase, we used the optimal floorplan thus obtained (case 13 FO4) and simulated the communication channel *with the real system timing* rather than random bursts. This gave us a final throughput of 0.65, at a length of 66%, or in short a data rate of 31% in excess of the maximal non-pipelined frequency, for a doubling of the operating frequency. The degradation in performance can be expleined by the non-random correlation between data bursts. It is likely that more sophisticated cost functions will be able to approximate better the real performance of the system, possibly giving higher final speedups.

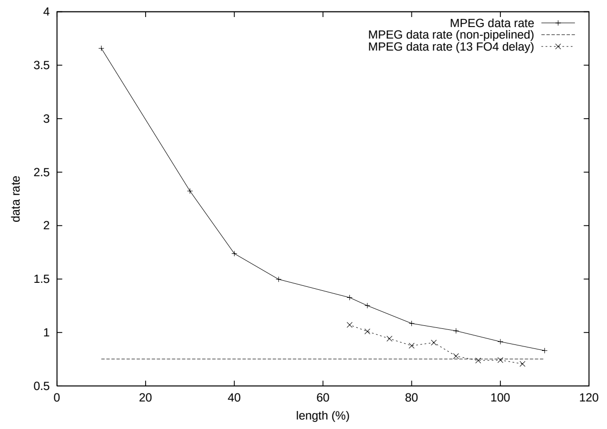

Figure 3: MPEG : Optimization of data rate with 13 FO4 and no block delay

We believe that this example shows the usefulness of the technique in practical cases, and prompts the investigation of more sophisticated schemes to further optimize and investigate the frequency/data rate trade off.

8. CONCLUSIONS

In this paper we first showed how the application of the wire pipelining technique to the design of systems-on-chip, which on the one hand increases the clock frequency otherwise limited by long slow interconnect, may not be able on the other hand to increase the actual data rate, that is the product of frequency and throughput. As a result, a generalized slowdown of the system clock to fit the longest interconnect delay, without wire pipes, may be a better choice. This occurs even using a floorplanning approach which is able to minimize the throughput degradation that a standard latency insensitive wire pipelining technique might imply. The reasons are the added latency in wires and the delay of logic and local wires within the blocks whose global interconnections have to be pipelined, as it is first shown formally and secondly demonstrated by means of experiments.

Then we showed how an enhanced wire pipelining technique in conjunction with a floorplan tool with a suitable cost function is able to increase the data-rate such that the convenience of using interconnect pipelines become evident. Besides standard floorplan benchmarks, we used a MPEG decoder as a case study to make more evident the efficiency of our approach, and at the same time to demonstrate how this methodology can be practically and successfully applied to a real-life example.

9. REFERENCES

[1] The International Technology Roadmap for Semiconductors (ITRS), 2003, SIA.

[2] M.R. Casu and L. Macchiarulo, “Floorplanning for Throughput,” *Proc. ISPD’04.*

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

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

[5] L.P. Carloni *et alii*, A Methodology for “Correct-by-Construction” Latency Insensitive Design”, Proc. ICCAD 99, pp. 309-315.

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

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

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

[9] M. Ikeda *et. al.*, A Hardware/Software Concurrent Design for Real-Time SP@ML MPEG2 Video-Encoder Chip Set, *Proc. European Design and Test Conf.,* pp. 320326, March 1996.