A New Approach to Latency Insensitive Design

Mario R. Casu
Politecnico di Torino/CERCOM
C.so Duca degli Abruzzi 24
I-10129 Torino, Italy
mario.casu@polito.it

Luca Macchiarulo
Politecnico di Torino/CERCOM
C.so Duca degli Abruzzi 24
I-10129 Torino, Italy
luca.macchiarulo@polito.it

ABSTRACT

Latency Insensitive Protocols have been proposed as a viable mean to speed up large Systems-on-Chip where the limit in clock frequency is given by long global wires connecting together functional blocks. In this paper we keep the philosophy of Latency Insensitive Design and show that a drastic simplification can be done that results in even no need to implement any kind of protocol. By using a scheduling algorithm for the functional blocks activation we greatly reduce the routing resources demand of the old protocol, the area occupied by the sequential elements used to pipeline long interconnects and the complexity of the gating structure used to activate the modules.

Categories and Subject Descriptors

B.7 [**Hardware**]: Integrated Circuits

General Terms

Algorithms, Design

Keywords

Interconnections, Pipelining, System-on-Chip

1. INTRODUCTION

The increasing complexity of integrated systems has naturally led to the idea of System-on-Chip where the designers have the opportunity of connecting together complex components taken from libraries of Intellectual Properties (IP) as in board-level design practice. The time-to-market driving force claims for an efficient reuse of such components therefore leading to the birth of the IP market. The integration and communication of IP's led to a shift in the designers' way of thinking because it became clear soon that the performance of such systems would have been communication-constrained rather than computation-constrained as in almost all previous integrated circuits design.

DAC 2004, June 7–11, 2004, San Diego, California,USA.

The platform-based design philosophy aims at correct-by-construction integration of IP components [1]. The orthogonalization of all the concerns related to the so called "silicon and design complexity" [2] is one of the objectives, 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 [3]. 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.

To deal with the problem with a uniform methodology that minimally affects traditional design flows, a few years ago, Luca Carloni and Alberto Sangiovanni-Vincentelli proposed a methodology they called "Latency Insensitive Protocols," (LIP) whose basic idea is to modify a design that works under the assumption of zero-delay connections between blocks (the "pearls") by encapsulating them with suitable wrappers (called "shells") and connecting them through internally pipelined blocks ("relay stations") complying with a formally defined protocol that guarantees identity of behavior [4][5].

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 [6], [7]. 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 [8]. 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. We show in this paper that 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 [8]). 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 reacting 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.

In section 2 we recall the issue of the throughput limitation resulting from the introduction of memory elements along interconnects. Then section 3 shows the possible implementation of a latency insensitive system using clock scheduling. How to calculate the schedule is discussed in section 4 and examples are used to clarify the method. Finally the effectiveness of the methodology is proven by a case study in section 5.

2. THROUGHPUT COMPUTATION

The addition of latency in interconnects is not always beneficial in terms of throughput. The presence of loops in the original netlist to which memory elements are added in order to pipeline the interconnects is of the essence for the computation of the final system throughput. We can follow the problem in the example of figure 1.

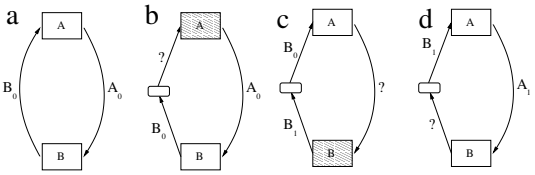

Figure 1: Throughput limitation example.

In part a, two blocks are shown to be connected by a bidirectional link in such a way that, *in every clock cycle* the output of A is needed by B and vice versa. As long as both connections are supposed to take no time, no performance bottleneck arises. If, on the other hand, we suppose that one of the two connections needs a pipeline in order to guarantee the maximum frequency constraint, it is clear that the system made up of the two blocks will no longer be able to work at the maximum throughput. Let's suppose that at moment t_0 A and B issue a valid datum (A_0 and B_0). At the next clock tick t_1, while block B is able to produce the next valid datum, having processed A_0, block A will not operate on any valid datum, given that its next legal input B_0 was delayed by one clock cycle by the presence of the pipelining element. 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 unit in a coordinated way.

With this in mind, let's follow how such a strategy could be employed to "legalize" the situation of figure 1. At time 1 (fig. 1b) block A should be prevented from operating: let's suppose that its clock is gated while B is able to produce a new output B_1 (dependent on its state and on input A_0 which it can read). At time 2 (fig. 1c), A will finally be able to compute its next output A_1 (based on its valid input B_0); at the same time, B has to be stopped as the datum it needs from A (that is, A_1) is just about to be computed, and therefore not ready yet. From this evolution it is clear that, even though the whole system proceeds in a synchronous fashion and evolves on the basis of the clock ticks, any single block is stalled from time to time. In this example, both A's and B's output (though not at the same time) are stalled once every 3 clock cycles. It is possible to prove that not only this result is possible (t.i., there exists a schedule that will allow a throughput of $\frac{2}{3}$) but also that this is the best possible result for that cycle: The presence of a new pipelining element in the cycle has the consequence of adding one clock delay to any computation that starts at a block and ends at the same block. A computation that, without the added element would have taken 2 clock cycles (the number of elements in the loop) to complete will now take 2+1 clock cycles, thus degrading the average performance to $\frac{2}{3}$. A similar line of thought brings to the following conclusion, which we will be constructively prove in section 4:

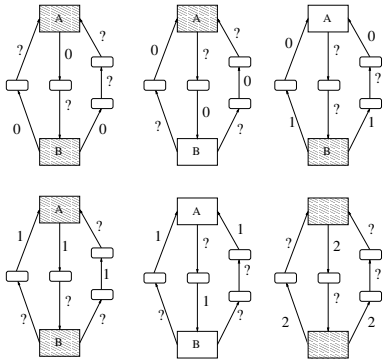

Figure 2: Combined loop.

Theorem: A strongly connected component of a block netlist always admits a schedule which allows an average throughput of $\frac{m}{m+n}$ with m blocks and n delays chosen on the loop that makes the expression minimal, and no schedule exists that allows a better average throughput.

The positive part of the theorem will be justified by the scheduling calculation of section 4. A sketch of proof for the negative part (no better schedule exists) is: Such a loop has $m+n$ synchronous delay, and the single datum takes $m+n$ clock cycles to complete the loop; at most m data can circulate at the same time. Therefore, it is not possible to have a higher throughput than $\frac{m}{m+n}$.

Looking at the case of figure 2, where invalid data are marked as "?", even though the left loop might proceed at a throughput of $\frac{2}{2+2} = \frac{1}{2}$, the presence of a node (and edge) in common with the right loop with a smaller maximal throughput of $\frac{2}{2+3} < \frac{1}{2}$, forces it to slow down accordingly.

The problem can be cast as a maximum time-to-ratio problem [13]. This observation is the theoretical basis for the calculations described in section 4, that show how a schedule that guarantees an *a priori* satisfaction of the maximum throughput can be evaluated on the basis of the circuit topology only, and doesn't need an explicit message passing between the various blocks and/or memory elements.

In order to get a sense of the issues at stake in practical cases, we developed an alternative method using a heuristical cost function within a simulation annealing based floorplan environment, whose full description is outside the scope of the paper (but see [11]). The results shown in section 5 will take into account interconnect delays that are consequence of such a floorplanning step, and therefore actually incorporate some geometric information of the problem.

The remaining part of this paper is then devoted to the methodology we developed for the optimal scheduling of the functional modules when interconnects connecting them are suitably pipelined so as to maximize the clock frequency.

3. SCHEDULING IMPLEMENTATION

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 3, two shells communicate with a unit latency and are activated two times every three clock ticks (two '1's loaded in their shift-registers). Notice that the schedule of the second

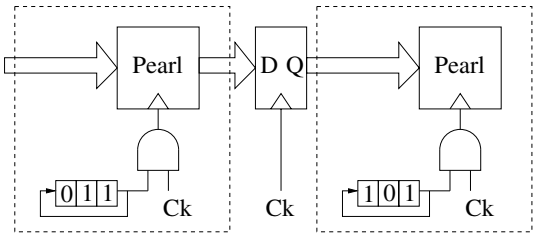

Figure 3: Conceptual scheme for scheduled pearls.

shell is obtained by a double left shift of the first one, because there is a sequential delay of two between them. The scheme in figure is "conceptual", the actual clock gating requiring at least a deglitching latch between the shift register and the *and* gate. The pipeline register schedule is obtained by one left shift of the first pearl schedule. Anyway the designer may decide to clock it always (as shown in figure, then saving the shift-register area) because the right schedule on the second pearl avoids consuming garbage data produced by the flipflop during its non-valid clock ticks.

If we compare the system in figure 3 to an analogous latency insensitive system [5], the saving in terms of resources is relevant. The relay station FSM is substituted with 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).

4. SCHEDULING CALCULATION

The advantage that might be obtained using such new latency insensitive design is clear from the simple discussion of previous section. But we still lack an element that makes it possible to implement a correct by construction latency insensitive system, t.i. a method to compute, for each and every shell, a valid schedule which respects timing constraints and ensures mutual synchonization. In order to be valid the schedule must satisfy two basic requirements: every valid datum has to be processed by the corresponding shell (no data loss), and every datum has to be processed only once (no data duplication). In other words the 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 [9]) describing the block connectivity.

An extension of the method described by Boyer *et alii* in [12], can be adapted for the issue here at stake. We refer to the original paper for a detailed description of the method, of which we simply give a brief summary. The problem solved in [12] consists in finding a schedule for memory elements that guarantees the maximal throughput (even above classical retiming) of synchronous circuits. Their circuit model is somewhat different from ours in that they imagine every combinational delay concentrated in the functional blocks, while they consider the connections delay-free. The method is so articulated:

1 The netlist is annotated by adding node weights representing their combinational delays and edge delays representing the number of **functional** flip-flops separating them.

2 A new graph is generated from the previous one by parametrizing the edge weights as $d(v) - \lambda \cdot w(v,u)$, where $d(v)$ is the starting node's weight, while $w(u,v)$ is the original edge weight. λ is a rational number representing the average time per datum of the system, the inverse of its throughput.

3 An iterative application of Bellman-Ford's algorithm is applied on the resulting graph in order to find the minimum value of λ that guarantes non positive cyclic paths in the graph weighted as in step 2 (and thus makes it possible for Bellman-Ford to terminate with no failure giving shortest paths from a given node).

4 The results of the last successful application of the algorithm, in terms of absolute path lengths from a reference node are used to compute the schedule on every single node, t.i. the possible phases of a clock of period λ that guarantees the correct computation of all data.

5 Finally, a minimum number of flip flops is inserted that enforces the schedule.

Step 3 of the strategy amounts to solve the maximum cost to ratio problem using the Lawvere method ([13], pp. 94-97). Expressed with our terminology, the cycle limiting cost-to-ratio is the cycle for which the ratio between the number of functional clocked elements divided by the overall combinational delay is maximum. In other words, optimum λ is the smallest period, or the inverse of the largest throughput.

The extension of the method to our case is straightforward: we need to add information that forces the insertion of pipelining in interconnects (which, we recall, are non-functional memory elements), by indicating how larger the delay from a node to another is due to the non-null wire delay. An easy technique is that of adding to the graph nodes

that model the interconnect delay. So, for example, if the connection between two nodes needs to be twice pipelined, we introduce two nodes with no sequential elements (because the starting graph models the functional behavior of the circuit, without any additional memory element) and with unitary delay. If we consider the result of the application of step 2 of the original algorithm to such a graph, we observe that in fact we need to annotate each edge originated in a functional block with $1-\lambda$ (each block has unit delay and adds a functional clocked element at its output), and each edge coming out of a "wire block" with 1. In practice, each edge will have a weight of $n+1-\lambda$ where n is the number of pipelining memory elements needed for its connection. It is easy to see then that the Lawvere strategy for the max cost-to-ratio problem, in this particular case, will give a value of λ which is exactly the inverse of the maximum throughput $(m/(m+n))$, because it will enforce the inequality $\sum_{i \in cycleC}((n_i+1)-\lambda) \leq 0$ for all cycles, inequality which amounts to forcing λ to be greater then or equal to $(m+n)/m$.

In our case, the computation can be cast in such a way that cycles of progressively higher and higher lengths are investigated: we first use $\lambda = 2/1$, then we choose between $\lambda = 3/1$ and $\lambda = 3/2$ depending on whether Bellman-Ford terminated with a legal solution or not, and so on, investigating in fact just a single value of λ for each value of m. Therefore, the convergence to the optimal value of throughput is fast, and the algorithm always terminates after a number of Bellman-Ford steps (each of complexity $O(|V| \cdot |E|)$ where $|V|$ and $|E|$ are the number of nodes and edges respectively), that are bounded from above by the longest loop. As in practical cases the smallest loops are typically the throughput-limiting factors, normally a few iterations are needed.

The final Bellman-Ford solution gives an optimal scheduling of the nodes, in the sense that it can be used to compute a sequence of clock enabling node by node that guarantees the safety of the system, while at the same time allowing the maximum possible throughput.

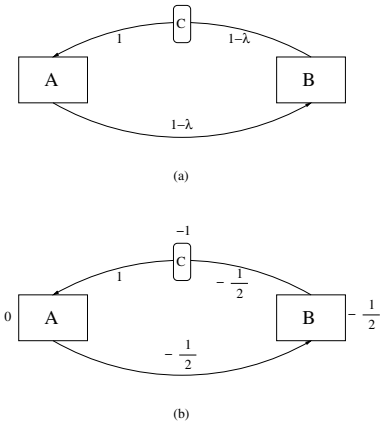

Figure 4: Schedule Calculation Example.

We cannot detail the proof of such claim due to space limits, but we try to illustrate how the algorithm works by analyzing two simple cases of a single loop and of two intertwined loops. Let's start considering figure 4, part a. It represents a simple loop made of 2 IP blocks (A and B) and a pipelining flip flop (C). According to the above description, we take into account the added delay (of up to 1 clock cycle) that requires the flip flop by adding one more node with a delay of 1. The other edges are labelled with $1-\lambda$. The Bellman-Ford problem gives the inequality (constraint that the cycle weight is always smaller than 0) $3-2\lambda \leq 0$, that fixes a minimum period of 3/2 and a corresponding maximum throughput of 2/3. In order to obtain a schedule for such a period we substitute the value $\lambda = 3/2$ and obtain the graph of figure 4b. A minimum length (in practice the last iteration of Bellman Ford needed to find the optimal value of λ) annotates the nodes (IPs and FF) as shown. They represent a fractionary schedule satisfying the timing constraint. Now, as we need to work with integer multiple of clock cycles we have to convert the exact fractionary schedule given (a@0, b@−1/2 , c@−1) into a pseudo periodic sequence. [12] shows that this can be done by considering the values of the schedule in successive periods and taking their floor ($\lfloor s_0 + k\lambda \rfloor$). In this case we find schedules for a (0,1) for b (1,2) and for c (0,2) - where (0,1) means that the block is enabled every zeroth and first time slot of the three slots in the schedule. It is possible to see that the schedules are equivalent to those shown in figure 1. It is particularly important to note that, even if the algorithm computes schedules for all elements in a loop, only IP blocks need to be clock gated to ensure functionality. This is due to the fact that flip flops are "sandwiched" between gated blocks and don't perform any computation on latched data. So, even if a value is overwritten in the same flip flop, it will be used only once by the following block, thus enforcing data coherence. The fact that overwriting doesn't ever result in data loss is enforced by the strategy described in section 4.2.

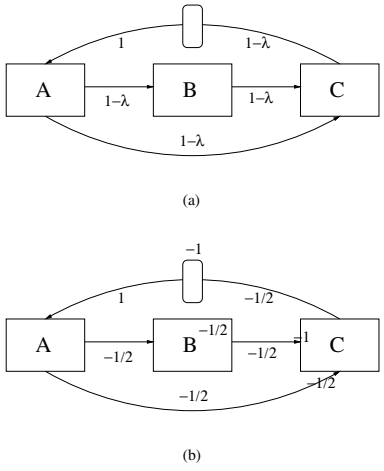

Figure 5: Schedule Calculation Example.

The more interesting case of figure 5 is useful to illustrate the technique and to draw an important conclusion. Proceeding as before, we find the weights of figure 5, where the optimal value of λ is still 3/2 as the critical loop has the same structure as before. The only difference w.r.t. the other case is that the IP c has two different schedules for its input, a fact potentially (even if not necessarily) dangerous for data coherence. In certain cases this asynchronism has

to be corrected by adding the correct delays to the "faster" branch of the reconvergent fanout. How this can be achieved in general will be discussed in the next subsections.

4.1 Reconvergent Fanouts - Feedforward case

As described in section 2, the presence of reconvergent fanouts in the netlist might give rise to throughput degradation of latency insensitive design. This is still true for our case, as a different sequential delay (number of flip flips) in two branches of a reconvergent fanout will de-synchronize data, thus obliging us to introduce void data on the fastest branch from time to time. This loss, however, can be easily avoided by adding the appropriate amount of queues on the fastest branch, in order to re-equalize the arrival times. The necessity of equalizing feedforward reconvergent paths has to be carefully analyzed on a case by case basis in order not to waste resources. Equalizing flipflops should not be added to an unbalanced path if the tighter limit is imposed by a loop or by another reconvergent path.

4.2 Reconvergent Fanouts - Feedback case

The same problem of reconvergent fanouts is possible inside loops (for example figure 5 has a reconvergent fanout that gives rise to two different loops). In this case, however, 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 possible implementation is shown in figure 6. 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.

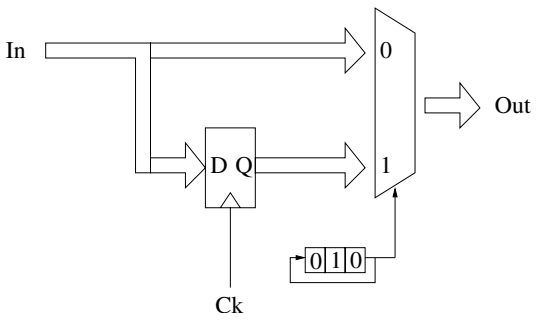

Figure 6: Fractional synchronizer.

4.3 Initialization and transient

One last problem needs to be solved before the system can be used in a transparent way with any possible IP netlist. The problem is already apparent by looking at figure 4. Even if the schedules computed allow the computation to be performed indefinitely in such a way that the same valid pattern will repeat itself periodically, we cannot start the computation from reset and pretend it will give rise to such a pattern. The problem is that at reset, the pipelining flip-flop is not yet holding a valid datum that node a needs in order to start its computation. For this reason, it is necessary to allow for a transient period that is arranged to distribute correctly valid "initial data" according to the chosen schedule.

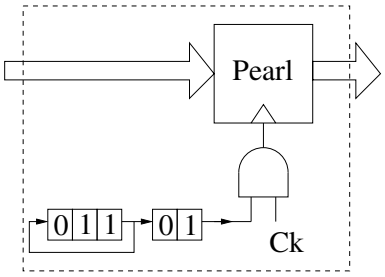

Figure 7: Transient and steady scheduling.

In the simple example, it is sufficient to wait for a clock cycle without activating any block, in order for the first valid output of block b to be latched by the flip flop. In general, a more elaborate initialization might be needed. From a practical point of view, this means that the scheduling computation will have in general a part which is used during initialization only, as shown in figure 7: The first part of the shift register will be used only once before the regular pattern takes charge. We can show that the maximum length of such a transient is equal to the longest sequence of consecutive pipelining flip flops. In practical cases, it is very likely to be smaller than the steady state part.

5. 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 [9][5]. The schematic in figure 8 represents the functional blocks of a Mpeg2 encoder. The small blocks along interconnects rep-

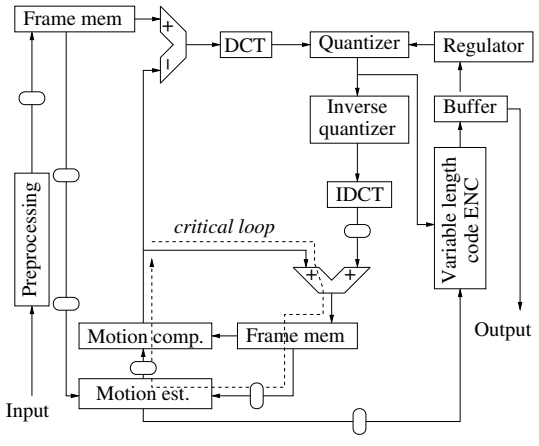

Figure 8: MPEG block diagram with pipelining memory elements.

resent pipeline flipflops. Their position and number is obtained after a physical design step which produces the floorplan reported in figure 9.

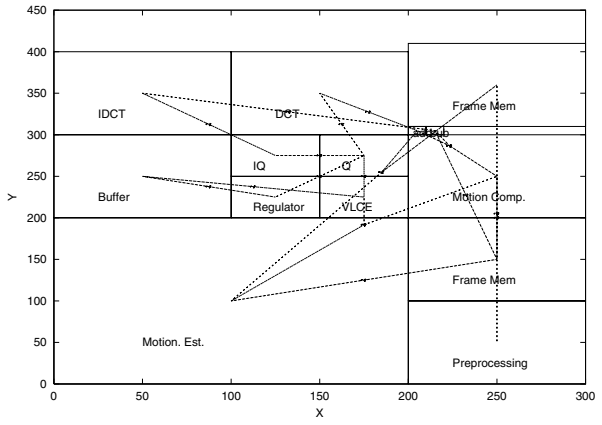

Figure 9: MPEG Floorplanning Example.

As we wanted to evaluate the effectiveness of the technique on a somewhat more grounded basis, taking into account the fact that the real concern that brought to the whole reflection on LIPs was one of physical design, we decided to implement and test the schedule calculation algorithm after a throughput-oriented floorplan of the MPEG system has been attempted in order to evaluate the effect of geometric constraints. We used a throughput-oriented floorplanner whose details can be found in [11].

In figure 8 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 8 we applied an implementation of the scheduler described in section 4 in order to identify valid schedules and path unbalance.

In figure 10 we report the results of a VHDL simulation of the entire system, after schedule computation. Only the output waveforms of the critical loop blocks are reported. The numbers in figure represent the data progressively produced at each block enabling as defined by its schedule. The simulation clearly 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.

6. CONCLUSIONS

The main aim of this work was that of showing that the employment of a pre-estimation of latency insensitive systems help in reducing their hardware and congestion overhead. The basic claim is that insensitive designs which have to be applied with small or no detailed knowledge of the building blocks, is bound to be defined by a statically defined schedule. On the other hand, we believe that the extension of the techniques here described to more flexible systems might lead to higher improvement of their overall performance. This will be the object of further investigations on the field.

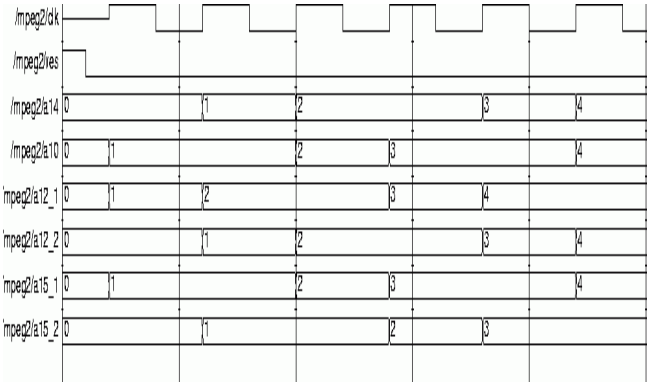

Figure 10: Shell simulation for the MPEG critical Cycle.

7. REFERENCES

[1] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," IEEE Computer, Jan. 2002, pp. 70-78.

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

[3] K. Keutzer *et al.*, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design IEEE Trans. CAD, Vol. 19, No. 12, Dec. 2000, pp. 1523-1543.

[4] L.P. Carloni, K.L. McMillan and A.L. Sangiovanni-Vincentelli, "Theory of Latency-Insensitive Design," IEEE TCAD, vol. 20, No. 9, Sept. 2001, pp. 1059-1076.

[5] L. Carloni and A. Sangiovanni-Vincentelli, "Coping with Latency in SOC Design," *IEEE Micro*, Vol. 22, No. 5, pp. 24–35, Sept.-Oct. 2002.

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

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

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

[9] L.P. Carloni and A.Sangiovanni-Vincentelli, "Performance Analysis and Optimization of Latency Insenstive Protocols," Proc. DAC'00, pp. 361-367, June 2000.

[10] H. D. Lin, D. G. Messerschmitt "Improving the iteration bound of finite state machines," ISCAS'89 pp. 1923-1928, vol.3.

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

[12] Francois R. Boyer, *et alii*, "Optimal design of synchronous circuits using software pipelining techniques," ACM Transactions on Design Automation of Electronic Systems, vol. 6, n. 4, pp. 516-532, 2001.

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