Designing optimized pipelined global interconnects: Algorithms and methodology impact

Vidyasagar Nookala and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455
{vidya,sachin}@ece.umn.edu

Abstract— **As across-chip wire delays exceed a clock cycle, interconnect pipelining becomes essential. However, the arbitrary insertion of flip-flops can change the differentials of latencies along paths in the circuit, and this can cause the implemented circuit to have a different functionality than was intended by the designer. Although it is possible to use design techniques that maintain the functionality of the circuit, an additional concern is a reduction in the throughput. This may be overcome by careful choices at various stages of design that impact the across-chip wire latencies. This paper surveys the published work on wire-pipelining, describes its impact on circuit as well as system level throughput, and outlines some of the problems to be resolved in formulating a wire-pipelining centric strategy for physical design.**

I. Introduction

The fraction of the interconnect delay in the shrinking system clock period has been increasing across process generations, and became dominant in the deep submicron (DSM) regime, where interconnects, in particular, those whose used for across-chip communication, have become a major bottleneck in determining the system performance. Even the theoretically best optimizers cannot ensure that the delay of a long global wire does not exceed a clock period [1]. This scenario, which is further aggravated by increasing die sizes [2], has caused researchers to look into alternative design methodologies that will enable multicycle across-chip communication, so that across-chip interconnect is removed from all the timing constraints, and the chip speed is determined by the most critical intra-block/local combinational path, in order to continue employing higher frequencies.

One solution is to provide a slower clock for the flip-flops latching signals from global wires, whose delays exceed the system clock cycle. However, the practicality of this approach is challenged by increased clock routing complexity, and accomplishing synchronization between the clock domains. Another approach is to adopt Latency Insensitive (LIS) [3] methodology, where each of the modules (called "pearls") comply a prespecified latency insensitive protocol, and are surrounded by wrappers (called "shells") which communicate with other modules through internally pipelined elements called relay stations comprising memory elements and some control logic. A related approach, Globally Asynchronous Locally Synchronous (GALS) [4], uses handshake protocols for inter-module communication. However, both GALS and LIS methodologies have yet to find widespread use, partly due to the lack of adequate CAD support, and partly because they entail a massive change in design practice, which designers are reluctant to embrace.

Another alternative is to employ *wire-pipelining*, where the delay of an interconnect is distributed over several clock cycles by inserting flip-flops. For instance, a corner-to-corner wire of length 4cm in a 2cm$\times$2cm chip has a projected delay of 1.4ns [1] in 70nm technology, which puts an upper bound of about 700MHz on the operating frequency. To operate the chip at a frequency of, say, 3GHz (corresponding to a clock period of 0.33ns), the delay of the wire can be spread over four clock cycles by inserting four flip-flops on the wire. Though it complicates the clock network routing, wire-pipelining, besides allowing higher operating frequencies, enables designers to remain in the purview of the traditional VLSI design methodology. Although such aggressive insertion of flip-flops in a circuit permits higher operating frequencies, there are several issues associated with wire-pipelining:

- The arbitrary introduction of extra flip-flops into a circuit can alter its cycle level behavior, requiring correction.
- Even if the above is corrected, an increase in the number of clock cycles required for each computation can result in reduced throughput.

In particular, the latter brings into question the utility of operating at a higher frequency at all, since it is possible that the use of a lower frequency, with a lower degree of wire-pipelining, may actually increase the overall throughput. Therefore, while a traditional methodology, augmented with wire-pipelining, can still be employed in nanometer technologies, the focus, particularly of physical design, must be throughput-centric. The next sections of this paper describe various aspects of wire-pipelining, including the abovementioned problems, and also outline the challenges faced in achieving a wire-pipelining aware physical design.

II. Techniques for wire-pipelining

We will begin by surveying the literature on optimally determining the locations of memory elements on a single multicycle interconnect net. It is reasonable to assume that the delay of an optimally buffered interconnect varies linearly with its length [5]. If we can determine the maximum length of a wire whose delay is within a clock period, called *critical sequential length* in [6], then the number of clock cycles required by a signal traveling the length of a particular wire can be estimated as the ratio of its length to the critical sequential length. The authors of [7] use this idea to pipeline an interconnect for a given clock cycle time. The approach uses the Elmore delay model to analytically compute the minimal number of buffers required to optimize the delay of a wire of a certain length, and from this, estimate the critical sequential length.

Two other recent works [8], [9] approach wire-pipelining at the global routing level. The technique of [8] finds a wire-pipelining solution to optimize a given interconnect topology such as a Steiner tree [5]. This approach extends the dynamic programming based buffer insertion algorithm of [10] by augmenting the buffer library with flip-flops. Given a Steiner tree, target clock period, required times at each of the destinations, candidate buffer/flip-flops insertion locations, a buffer and flip-flop library, the algorithm finds an optimal assignment of flip-flops and buffers on the buffer locations, which minimizes the number of flip-flops between the source and the latest

This work was supported in part by the NSF under award CCR-0205227 and by a gift from Intel Corporation

destination of the net. In [9], wire-pipelining is handled in conjunction with global routing. The approach, based on the fast path algorithm [11], is to simultaneously route and insert buffers and flip-flops to optimize a two pin wire. The algorithm transforms the chip area into a grid graph, where the edges and vertices corresponding to the given blockages are deleted, and finds a minimal latency route from the source to the sink of the net. The solution involves the propagation of wave-front from a vertex to its neighbors, similar to maze routing [5]. Both of the approaches use distributed Elmore wire delay models, and keep track of multiple partial solutions at every step and use techniques to prune inferior solutions to decrease the search space.

III. Wire-pipelining aware design

The methods described in Section II can serve as aids in the evolution of a wire-pipelining aware design strategy for circuits in the nanometer era. Specifically, the approaches estimate the number of clock cycles and optimization elements (buffers and flip-flops) required by a particular wire/net, which will facilitate the formulation of new design objectives at various levels of the existing methodology, such as logic synthesis and physical design.

A. Wire-retiming

Initial research attempts focused on retiming to integrate wire-pipelining. Retiming [12] is a transformation which involves repositioning of registers in a circuit primarily to reduce the clock period. A number of efficient algorithms for retiming a circuit have been proposed in the last decade. However, most of them assume that wire delays are negligible as compared to those of the devices or gates of the circuit. With the dominant nature of wire delays in the DSM designs, this is an unrealistic assumption. This gained attention in the research community in the last couple of years and resulted in approaches such as [13]–[15], which extend retiming by including the wire delays, given a placement, besides the gate delays, in the formulation. The objective of [13], [14] is to minimize the clock period of a circuit by retiming flip-flops into the critical wires of the circuit. While [13] focuses on gate level net-lists, [14] handles block level layouts such as System-On-Chip (SOC) circuits. In [15], retiming is combined, at floorplanning level, with module selection. The work assumes multiple implementations for each module, each with a different latency and area, for a given frequency specification, and this is captured by a area-delay curve, where the area reduces as latency increases. The objective is to choose an implementation for each module that minimizes the area of the floorplan subject to a lower bound on each wire latency.

B. Frequency constrained wire-pipelined circuits

The advantage of the above retiming-based implementations is that the functionality of the circuit is not altered. On the other hand, since retiming merely moves around the existing latencies in a circuit and preserves cycle latencies and input-output latencies in the circuit, there is a lower bound on the achievable clock cycle time that depends on the latencies that are available in the circuit. In other words, it is quite possible that the use of retiming-based methods may prevent the specified clock period from being met.

An alternative approach, which avoids this limitation, can have the following design flow. After the blocks of the circuit are designed subject to a clock frequency, a block-level placement of the circuit is performed. Wire-pipelining is then carried out on the global wires of the circuit, and this may insert flip-flops on a wire if the delay of the wire exceeds a clock cycle. However, such a method is tantamount to arbitrarily inserting extra flip-flops in a circuit, which can change the functionality of the circuit. After the wires of a circuit are pipelined, the following two problems must be resolved:

- The latencies of the cycles of the circuit may increase.
- There may be nonuniform increases in the latencies of different paths to a block from the inputs of the circuit.

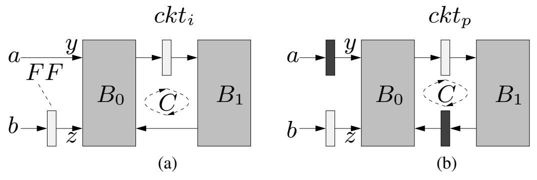

Fig. 1. A circuit with inputs a and b; y and z are the input ports of B_0. (a) The circuit before pipelining its wires. (b) The circuit after pipelining its wires.

Figure 1 depicts a circuit comprising two combinational logic blocks B_0 and B_1, which also form the cycle C, before and after pipelining the wires of the circuit. The two scenarios are labeled ckt_i and ckt_p, as shown in Figures 1(a) and 1(b), respectively. The insertion of an extra flip-flop on C increases its latency to 2 in ckt_p from 1 in ckt_i. Hence, the output of each block of C propagates back to itself after 1 clock cycle in ckt_i, whereas it takes an extra clock cycle in ckt_p, thus altering the original functionality of the cycle. Moreover, with the insertion of an extra flip-flop between a and y, the inputs a and b reach y and z, respectively, after an equal number of clock cycles in ckt_p, which is not the case in ckt_i. Hence, ckt_i and ckt_p are not functionally equivalent.

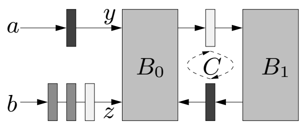

Fig. 2. A solution to the problem shown in Figure 1.

Wire-pipelining can therefore result in a totally different microarchitecture. This is not the desired result and therefore, must be corrected. A few methods for correction [16], [17] have been proposed recently. The technique of [16] lies in ensuring that every block receives its inputs at the correct clock cycle. For increased cycle latencies, an approach similar to the *c-slow* concept of [12] is used. The idea is to *slowdown* the input issue rate of the circuit by some factor ρ, i.e., inputs are allowed to change only every ρ^{th} clock cycle. For instance, the cycle C of ckt_p will be functionally equivalent to C of ckt_i, if the inputs a and b are permitted to change only every other clock cycle in ckt_p. As a result, ckt_p computes its outputs only every 2 clock cycles, causing a reduction in the throughput. For a general cycle, slowdown ρ required by the cycle in wire-pipelined circuit is the ratio of the initial and wire-pipelined latencies of the cycle. Moreover, if the ratio is not an integer, it is rounded to the nearest highest integer, since ρ represents the number of clock cycles between successive input changes.

In general, a circuit may have more than one cycle and each of these may require a different slowdown. The critical cycle, which requires the maximum slowdown, sets the lower bound for the slowdown required for the entire circuit, since the latencies of other cycles can be increased to match this slowdown. Obtaining minimal ρ is an instance of Maximum Cycle Ratio (MCR) Problem [18], as shown in [16]. As the circuit computes its outputs only every ρ clock cycles, the frequency speedup obtained by pipelining the wires of a circuit does not entirely translate into the performance speedup, and is reduced by a factor of ρ.

In addition, the latency difference between any two paths to a block from the inputs of a circuit must also be maintained in its wire-pipelined version. However, the slowdown has implications on the path latencies of a wire-pipelined circuit. For example, the latency difference of the paths $a \rightarrow y$ and $b \rightarrow z$ in ckt_i must be amplified by a factor of $\rho = 2$ in ckt_p, since it receives its inputs only every 2 clock cycles . Therefore, since the latency difference between the paths $b \rightarrow z$ and $a \rightarrow y$ is 1 in ckt_i, 2 extra flip-flops must be inserted on the path $b \rightarrow z$ in ckt_p, as shown in Figure 2.

In [17], an alternative approach, in the lines of latency insensitive design style, for functional correction is proposed. The technique is based on the software pipelining model of [19], which involves obtaining a schedule for each block. Similar to [16], the method handles increased cycle latencies by slowing down the circuit. However, unlike [16], fractional values are allowed for ρ, thus yielding a better throughput. To accomplish this, each block is provided with a shift register based clock gating mechanism. For example, a slowdown of $\rho = \frac{3}{2}$ indicates that each block of the circuit must receive the clock signal only twice every 3 clock cycles, and this can be achieved by gating the clock signal for a clock cycle. Similar to [16], slowdown affects the input-path latency differences. In addition, a fractional slowdown can result in fractional slacks for some paths, and this is solved by inserting *fractional synchronizers* in the circuit.

C. Impact on system level performance metrics

Wire-pipelining has a different kind of performance impact on microprocessors, such as superscalar processors. The execution time, T_{exec}, of a program on a processor is the product of three terms:

$$T_{exec} \quad = \quad num\_instr \cdot \text{CPI} \cdot T_{clk} \qquad (1)$$

where CPI is the average number of clock cycles per instruction and T_{clk} is the clock cycle time. In general, CPI may vary across different programs executed on the processor, based on the instruction mix executed, the accuracy of the branch predictor, if used, etc. Furthermore, even for the same program, CPI can change with the input set executed. Several superscalar processor simulators such as SimpleScalar [20] have been widely used in academia and industry to estimate the CPI for a given architecture and a given set of programs, most commonly benchmark programs such as SPEC [21]. Such simulators permit optimizations at software (compiler) as well as hardware (microarchitecture) levels, and execute a program cycle by cycle.

The way that wire-pipelining is interpreted in the microprocessor context is different from how it is considered for a general logic circuit. For a given program and an input set, the number of instructions, num_instr, is invariable[1]. Pipelining the buses of a microprocessor can be absorbed internally by the processor through, for instance, pipeline stalls. This may increase some operation latencies, branch misprediction and cache miss penalties, etc., thus increasing the CPI, while allowing higher operating frequencies. For instance, an extra latency on a wire between a reservation station [20] and an integer ALU effectively increases, by one, the latency of all of the integer ALU operations in the program. In such a scenario, the correction primarily involves adjusting the operation and stall latencies. In microprocessors, the throughput metric is the CPI, unlike MCR for a general circuit, and this can be determined by incorporating wire-pipelining models in the chosen simulator and cycle-accurate simulations on the selected set of programs and input sets.

[1]Compiler level optimizations may affect num_instr, however.

D. Impact on physical design

It must be clear from the discussion of Sections III-B and III-C that wire-pipelining can degrade the throughput of a circuit, indicating the necessity for a throughput optimal design strategy [22]. Since the latency of a wire is determined by the positions of the corresponding blocks and the routing of the wire, the throughput, CPI or MCR, can be reduced by using better objective functions in physical design, which will attempt to reduce the lengths of throughput critical wires. This marks a clear deviation from the traditional design flow, where the design goals are, in general, reducing a weighted sum of area and total wire-length. The change can start at the floorplanning level, where the objective may be to find a CPI/MCR optimal block level placement, in addition to minimizing the total wire-length. This can be propagated to the next levels, such as pin assignment and global routing, as shown in Figure 3, which also lists some of the operations that are affected at the corresponding stage. For instance, the goal of the global routing step may be to minimize the congestion along the routes of the throughput-critical wires. In some cases, where pin assignment or routing is realized net-by-net, the nets may be processed in the decreasing order of throughput criticality.

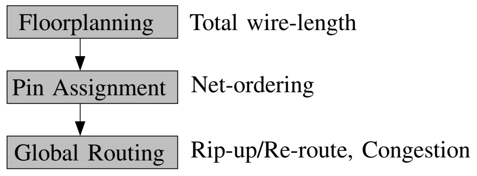

Fig. 3. Higher stages of physical design, and some of the optimization steps impacted by wire-pipelining.

Most of the published work so far has focused on floorplanning. In [23], the objective of the Simulated Annealing (SA) based floorplanner used is to minimize MCR. Since SA involves a huge number of moves, given the complexity of computing MCR, it is impractical to evaluate this parameter at every SA step. This issue is approached by assuming that each of the blocks contributes a latency of 1, in which case, cycles formed by a fewer number of blocks are critical than those formed by a relatively higher number of blocks. To model this, for each wire, a weight inversely proportional to the size of the shortest cycle it belongs to, is assigned. The cost function of SA is the weighted sum of wire latencies.

Other recent works [24], [25], also at floorplanning level, focus on microprocessors to reduce CPI. Similar to [23], [24] and [25] heuristically evaluate throughput of a given layout, CPI in this case. While the SA based floorplanner of [24] uses a Look Up Table (LUT) to estimate the CPI at each step, [25] captures the throughput with an assignment of weights to the buses, determined from cycle-accurate simulations, and the weight of each bus is proportional to the number of data transfers along the bus.

IV. Challenges ahead

There has been a good progress towards wire-pipelining awareness at the floorplanning level [23]–[25], and each of these can be extended to other physical design stages such as pin assignment and routing. However, the problem lies in the accuracy of the throughput estimations of these methods. For instance, in [24], an LUT of size 50-100 as mentioned in the paper, may not be accurate enough to estimate the CPI of thousands of candidate layouts traversed during SA. On the other hand, the use of access ratios to optimize a layout, as in [25], does not necessarily model the quantitative impact of bus latencies on CPI. For example, the impact of a branch misprediction may change across different penalty values for the same program.

Regarding [23], the assumption that short cycles are dominant may not be true in general, and therefore may not work for all circuits. A prototype of a design flow, similar to that of [23]–[25], is shown in Figure 4. The physical design stages use the throughput, MCR or CPI, determined by the estimator to optimize the layout. If the results are not satisfactory, a few microarchitectural changes may be made and the whole process can be repeated. In such a design flow, the following lists some of the problems to be solved:

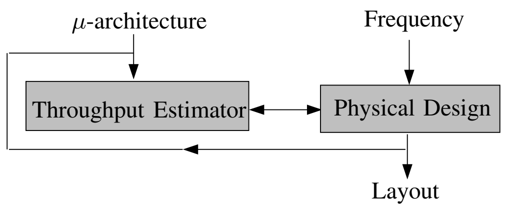

Fig. 4. Wire-pipelining aware physical design.

- **Throughput estimation:** This, clearly, is the bottleneck in the design flow of Figure 4. For microprocessors, each cycle-accurate simulation typically consumes a huge amount of time, with some programs executing for days. Several time reduction techniques have been proposed in the literature, such as fast forwarding, reduced input sets, etc., to simulate only representative parts of a program. However, the cost of doing this is reduced accuracy of the results. The runtime-accuracy tradeoffs must be carefully evaluated, based on the number of simulations required to build cost functions. In addition, there is need to focus on methods to reduce the number of simulations, without compromising the accuracy.
- **Frequency-CPI tradeoff:** The material presented so far caters to the fixed frequency designs, where the only optimization variable is CPI/MCR. It is also possible to explore the frequency-throughput tradeoff as shown in [26], again at floorplanning level. This method does not assume wire-pipelining, however. For wire-pipelining case, in [25], where the bus access ratios are invariable across clock frequencies, iterative of floorplanning is suggested, with a different frequency at each iteration. In general, wire-pipelining complicates the scenario when exploring frequency-CPI tradeoff. This may not be of prime concern, however, since in most cases, the frequency is a strict constraint given to the designer, and the job of the designer is to achieve the maximum possible performance subject to that constraint.
- **Power consumption:** Power, both switching and leakage, can be affected by wire-pipelining in a number of ways. For instance, power increases as the number of flip-flops inserted increases. Furthermore, throughput and power consumption of a circuit can vary in contrasting ways, suggesting a tradeoff. For example, while pipeline stalls affect CPI, techniques such as clock gating can reduce the effect on power consumption. On the other hand, there could be some power intensive operations, which execute on blocks with huge logic. In such a case, even when a few number of such operations are executed, the power consumption may be high, and this may not be the case with CPI.
- **Buffer explosion:** The number of buffers and flip-flops required to optimize the wires of a circuit increases exponentially as the technology advances, as shown in [6].
- **Local interconnect:** As the circuit complexity and clock frequencies increase, at some point, the individual blocks become big enough, and the local wire delays can exceed a clock cycle. In such a scenario, it may be wise to reduce the granularity by splitting each block into sub-blocks.

V. Conclusion

With the dominating nature of global interconnect, wire-pipelining has become essential in high-frequency design. However, unless it is carefully applied, it can alter the functionality of a circuit. Corrective techniques may be used to avoid this, but the increased cycle/operation latencies can reduce the throughput of the circuit. Therefore, considerable research must be carried out in developing throughput-sensitive physical design techniques that ensure, from early stages of the design cycle, that the performance impact of wire-pipelining is low.

References

[1] J. Cong, "An interconnect-centric design flow for nanometer technologies," in *Proc. IEEE*, vol. 89, pp. 505–528, Apr. 2001.
[2] S. Borkar, "Obeying Moore's law beyond 0.18 micron," in *Proc. IEEE ASIC/SOC Conf.*, pp. 26–31, Sep. 2000.
[3] L. P. Carloni *et al.*, "Theory of latency-insensitive design," *IEEE TCAD*, vol. 20, pp. 1059–1076, Sep. 2001.
[4] D. M. Chapiro, *Globally-Asynchronous Locally-Synchronous systems*. PhD thesis, Stanford University, Stanford, California, Oct. 1984.
[5] N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, Massachussets, 2nd ed., 1995.
[6] P. Saxena *et al.*, "The scaling challenge: Can correct-by-construction design help?," in *Proc. ACM ISPD*, pp. 51–58, Apr. 2003.
[7] R. Lu *et al.*, "Flip-flop insertion for early interconnect planning," in *Proc. IEEE DATE conf.*, pp. 690–695, Mar. 2002.
[8] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proc. IEEE ICCAD*, pp. 268–273, Nov. 2002.
[9] S. Hassoun *et al.*, "Optimal buffered routing path constructions for single and multiple clock domain systems," in *Proc. IEEE ICCAD*, pp. 247–253, Nov. 2002.
[10] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE ISCAS*, vol. 2, pp. 865–868, May 1990.
[11] H. Zhou *et al.*, "Simultaneous routing and buffer insertion with restrictions on buffer locations," *IEEE TCAD*, vol. 19, pp. 819–824, Jul. 2000.
[12] C. E. Leiserson *et al.*, "Optimizing synchronous circuitry by retiming," in *Proc. Third Caltech Conf. on VLSI*, pp. 87–116, Mar. 1983.
[13] C. Chu *et al.*, "Retiming with interconnect and gate delay," in *Proc. IEEE ICCAD*, pp. 221–226, Nov. 2003.
[14] C. Lin and H. Zhou, "Retiming for wire pipelining in system-on-chip," in *Proc. IEEE ICCAD*, pp. 215–220, Nov. 2003.
[15] A. Tabbara *et al.*, "Retiming for DSM with area-delay tradeoffs and delay constraints," in *ACM DAC*, pp. 725–730, Jun. 1999.
[16] V. Nookala and S. S. Sapatnekar, "Correcting the functionality of a wire-pipelined circuit," in *Proc. ACM DAC*, pp. 570–575, Jun. 2004.
[17] M. R. Casu and L. Macchiarulo, "A new approach to latency insensitive design," in *Proc. ACM DAC*, pp. 576–581, Jun. 2004.
[18] A. Dasdan *et al.*, "Efficient algorithms for optimum cycle mean and optimum cycle cost to time ratio problems," in *Proc. ACM DAC*, pp. 37–42, Jun. 1999.
[19] F. R. Boyer *et al.*, "Otimal design of synchronous circuits using software pipelining techniques," *ACM TODAES*, vol. 6, pp. 516–532, Oct. 2001.
[20] D. C. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," Technical Report CS-TR-97-1342, The University of Wisconsin, Madison, Jun. 1997.
[21] J. L. Henning, "SPEC CPU 2000: Measuring CPU performance in the new millennium," *IEEE Computers*, vol. 33, pp. 28–55, Jul. 2000.
[22] L. Scheffer, "Methodologies and tools for pipelined on-chip interconnect," in *Proc. IEEE ICCD*, pp. 152–157, Oct. 2002.
[23] M. R. Casu and L. Macchiarulo, "Floorplanning for throughput," in *Proc. ACM ISPD*, pp. 62–67, Apr. 2004.
[24] C. Long *et al.*, "Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects," in *Proc. ACM DAC*, pp. 640–645, Jun. 2004.
[25] M. Ekpanyapong *et al.*, "Profile-guided microarchitectural floorplanning for deep submicron processor design," in *Proc. ACM DAC*, pp. 634–639, Jun. 2004.
[26] J. Cong *et al.*, "Microarchitecture evaluation with physical planning," in *Proc. ACM DAC*, pp. 32–35, Jun. 2003.