Experimental evaluation of place-and-route of regular arrays on Xilinx chips

Erwan Fabiani and Dominique Lavenier
IRISA
Campus de Beaulieu
35042 Rennes Cedex
France

Abstract *This paper presents a synthesis of several experiments to map regular arrays onto Xilinx chips. We show that constraining only the placement results in a limited place-and-route runtime gain under timing constraints. The analysis covers two Xilinx families (XC4000 and Virtex), highlighting the growing influence of the routing step and the necessary need of advanced tools for controlling the routing process.*

Keywords: structural placement, regular arrays

1 Introduction

When implementing a hierarchical structured design on a FPGA component, one must decide between letting the vendor tools do it alone or constraining them with layout directives deduced from the design structure. This method usually consists of attaching relative or absolute locations to the design. This is done at 2 levels: (1) at a logic block level to place complex components (like RAM blocks or multipliers); (2) at a component level to place nearby components that are tightly connected. The expected result is an increase of the clock frequency due to the reduction of routing delays, as well as a fast place-and-route step since no placement has to be found.

We call such a placement a "structural placement" since it takes into account the design structure, as opposed to vendor placement which flattens the design into logic blocks. In the last decade, several authors using structural placement on FPGA have observed benefits over vendor tools, both for frequency and place-and-route runtime criteria ([1] [2] [3]).

Regular arrays mostly benefit from structural placement. They consist of replicated identical cells and the placement method is straightforward: it consists, for a cell, in finding a structural placement, then replicating it according to the layout topology of the array [4]. This method has been extensively investigated in the COSI team at IRISA, especially for linear arrays. The purpose is to provide a fast place-and-route tool for implementing regular arrays generated from automatic parallelization of nested loops [5].

However, the last few years have seen important evolutions in the FPGA architectures, as well as in the synthesis tools. In order to moderate the impact of the increasing size of FPGA components over the design cycle time, vendors have made a huge effort to improve the place-and-route tool. For instance, Xilinx claims that their latest tool (3.1 release 2000) has a runtime speedup of 16 compared to the release of 1996 [6]. Moreover, tools now take structural criteria into account unlike the situation in the past. Xilinx tools, again, now detects registers by their names and is able to map them up to have contiguous bits in a same logic block [7]. The architectural evolution has been principally concentrated on routing resources [8], and Xilinx even claims that the design cycle time criterion was taken into account when the Virtex architecture was created.

Last year, S. Singh was foreseeing the "Death

of RLOC" [9]. He was showing that benefits of structural placement were not as interesting as in the past for both runtime and frequency.

In this context, we have to wonder if implementing regular arrays using structural placement is still profitable considering the evolution of FPGA architecture and synthesis tools. Actually, the usual method to implement a regular array with structural placement only focuses on the placement step: the array placement is fully constrained, but there is no control on the routing step. In addition, the assumption that a smart placement could provide a significant reduction of the routing time is not clearly demonstrated, in spite of the numerous experiments we have done.

This article aims both to evaluate the limited benefits of a structural placement of regular arrays, and to quantify the evolution of the place-and-route runtime between past and current FPGA technologies. The rest of the paper is organized as follows: the next section presents the experimental context. Section 3 describes the set of experiments we have performed on a Xilinx Virtex XCV800 component to measure the place-and-route time, and to estimate the clock frequency with and without placement constraints. Section 4 compares the place-and-route time between the current Xilinx family (Virtex) and the previous one (XC4000). Section 5 concludes and points out the needs for better controlling the routing process.

2 Experimental context

Our experiments consist basically of comparing the results of two implementations: a placed one and an unplaced one. As we want to compare the maximal potential runtime improvement that one can obtain with a structural placement, we focus on implementing arrays with as many cells as possible.

In the placed implementation, one cell is structurally placed to fit in a minimal area rectangular shape. It is also placed to minimize the critical path delay by bringing closer components that are on the critical path. This placement is then replicated (and mirrored as and when needed) for all the cells. In the unplaced implementation, no directive is added to the net-list: the placement is fully done by the vendor tool.

For the two implementations, the net-list (with or without placement) is placed-and-routed by the vendor tool. The runtime of these two steps and the estimated critical path delay are measured. All the comparisons have been made with and without *timing constraint* as this parameter can have a very huge impact on the time for placing-and-routing a circuit. In order to have an standard way of comparing its influences, we use a normalized scale: for each design, the base is equal to the minimal timing constraint (i.e. the maximal frequency constraint) the vendor tool is able to satisfy for placing-and-routing the unplaced implementation. We made experiments for various timing constraints up to 3 times this constraint.

Three designs have been used for our experiments:

- A unidirectional convolution with a constant coefficient multiplier. Each cell has 2 registers, an adder and a constant coefficient multiplier. This design is tested for 4, 8 and 16-bit datapath (named conv4, conv8 and conv16 in the following).

- A systolic filter for DNA similarity search [10]: A cell consists of 4 comparators, 4 multiplexors, 4 adders and 3 registers, with a 12-bit data width.

- A systolic array for computing the k-means clustering algorithm for hyperspectral images [11]: A cell includes 3 comparators, 3 adder/subtracter, 4 multiplexor, 6 registers, a 224 by 8 bit Ram memory, on a 16 bits and 8 bits datapath.

These three designs are all linear systolic architectures. However, their implementation on a FPGA component requires a 2D organization and the inter-cell locality is preserved by means of a snake-like mapping as explained in [4].

3 Placing regular arrays on a Virtex FPGA

To evaluate structural placement, we measured the ratio between placed and unplaced designs according to three criteria:

- *Place-and-route time:* if the ratio is greater than 1, then the place-and-route with placement is faster (fig 1.a). It appears that for some simple designs (conv8, conv16), there is not much improvement induced by placement. On the other hand, for more complex designs (DNA and kmean) the place-and-route time can be significantly speeded-up, especially if no timing constraints are specified.
- *Routing time:* it is important to know if a structural placement also provides routing time improvement. Figure 1.b confirms the previous experiment: the better the routing time, the better the overall place-and-route runtime. This shows that a structured placement of a regular array does not provide a faster routing step. Worse yet, in some cases, the routing time may even increased!
- *Delay:* the ratio shows whether structural placement improves the clock frequency. Figure 1.c shows that designs with placement constraints are always faster. It is particularly true when there are no timing constraints. In fact the delay of the designs with placement constraints are not very effectively reduced using timing constraints.

From these experiments, one can easily detect when regular arrays benefit from structural placement: when no timing constraints are set. In that case, the clock frequency is increased and the overall place-and-route stage takes the shorter time.

Hence, using timing constraints with a structured placement seems to be of poor interest when implementing regular arrays on current Xilinx FPGA component. This contradicts previous results we had from the XC4000 family. Since the overall speedup comes mainly from the placement step, the balance between the placement and the routing step has a great influence. The longer the placement step, the better the overall runtime improvement. To see if this place-and-route runtime ratio is correlated with the FPGA architectures, we compared this ratio between past (XC4000) and current (Virtex) FPGA technology.

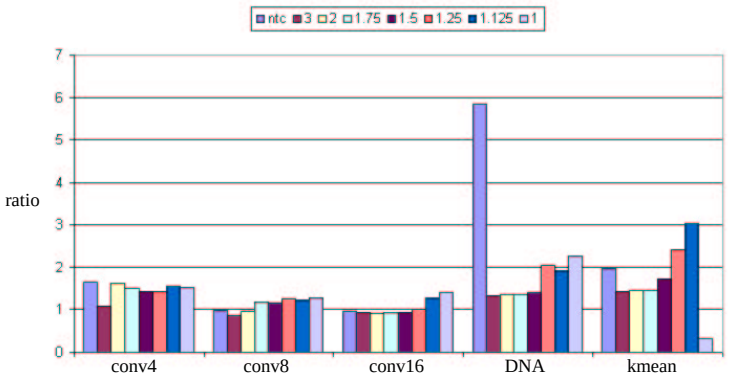
4 XCV800 and XC40250 Place-and-Route Comparison

In this section we compare the place-and-route time for various unplaced designs both on a XCV800 and a XC40250XV component. Those two components have approximately the same number of LUTs (the Virtex XV800 is 10% bigger) and can be placed-and-routed with the same tool (Xilinx PAR 3.1). Those two architectures represent the evolution that occurs in FPGA architecture. The XC40250XV is the biggest component of the Xilinx XC4000 family while the XCV800 is a middle size component of the Virtex family. The principal architectural difference between these two architectures is the amount of available routing resources: a Virtex LUT can access approximately 1.5 more routing resources than a XC4000 LUT.

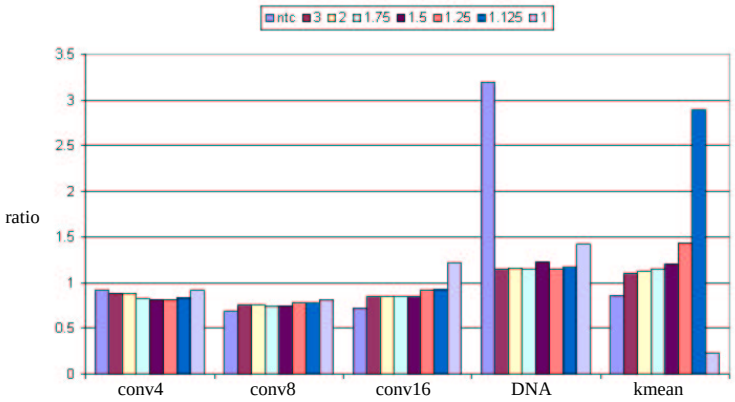
For comparison purpose, the three designs presented in the previous section have been placed-and-routed on the XC40250XV with no placement constraints. Figure 2 gives the ratio between the placement time and the routing time for each placed-and-routed design both on the XC40250XV and the XCV800.

It appears that this ratio is often significantly higher for designs implemented on the XC40250XV, whatever the timing constraint. This is particularly true for array with fine grain cells, as conv4 or conv8. The single exception is the kmean without timing constraints.

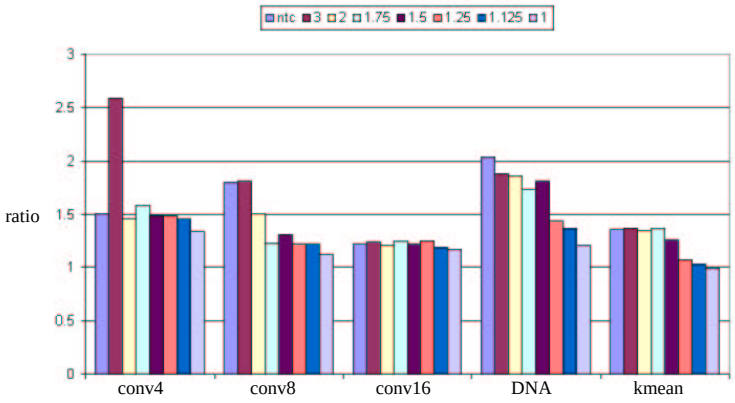
We may now wonder whether this effect is due to a faster placement or a longer routing time. These measures are not detailed here,



(a) Improvement in total place–and–route runtime between a placed and an unplaced design.



(b) Improvement in routing runtime alone between a placed and an unplaced design.



(c) Improvement in critical path delay between a placed and an unplaced design.

Figure 1: Comparison of place and route runtime, routing runtime and critical path delay between linear array implementation done with or without placement constraints. Implementations are done on a Xilinx Virtex XCV800 with timing constraints varying from 1 to 3 time the minimal timing constraint, or without timing constraints (*ntc*).

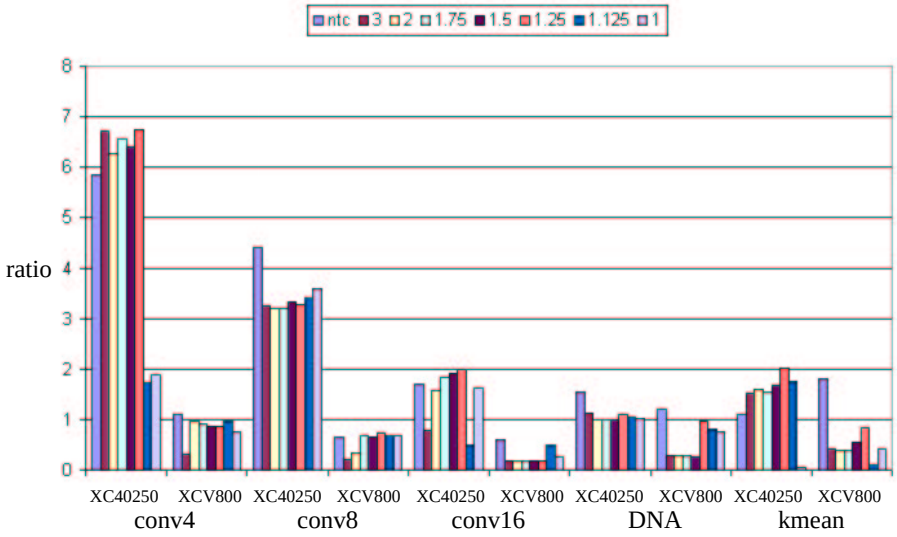


Figure 2: Comparison of place/route runtime ratio for designs on XCV800 or XC40250XV, for various timing constraints.

but globally the XCV800 placement is divided by an average factor of 15, while the routing time is only divided by an average factor of 6. Clearly, the decreasing time of the place-and-route step is mainly due to a faster placement operated by the Xilinx tools.

However those measures should be moderated by the size of the two architectures: although the XC40250XV and the XCV800 have comparable number of logic blocks, the XC40250XV should be regarded as one of the bigger sized FPGA of the XC4000 family, that is to say the maximal reasonable amount of logic that can be implemented regarding the XC4000 family routing architecture. On the other hand the XCV800 is a middle size component of the Virtex family (30% of the bigger component), and does not reach the amount of critical logic a single component of this family can fit.

5 Conclusion

In this article the current benefits of structural placement on regular arrays have been evaluated using a middle size component of the Xilinx Virtex family. It appears that, although structural placement provides interesting gains in clock frequency, it cannot provide a significant time saving for the place-and-route step. This is explained by the fact that a smart placement does not have much influence on the routing step. Moreover, when comparing the implementation of similar designs on an older Xilinx architecture, it appears that the fraction of time spent in the placement step decreases. The conjugation of these two factors tends to weaken the efforts to place a regular array for reducing the place-and-route time.

However, the structural placement of regular arrays, as we investigated it until now, only partially exploits regularity: the regularity of the routing itself is not exploited. Obviously, if all the cells of an array can be placed with the same pattern, then their internal routing could follow identical features. There are no theoretical reasons forbidding to replicate the cell routing inside a regular structure.

Hence, the fundamental issue in reducing the place-and-route time would be the ability to find the placement and the routing of one cell and then to replicate it over the reconfigurable structure. Such a method requires the solution

to three technical problems : (1) location independent routing (i.e. a routing that could be reproduced in all locations of the FPGA), (2) confined routing (i.e. a routing that do not exceed a perimeter devoted to the cell routing), (3) replication of such a routing.

The first problem constrains FPGA routing architecture : it implies that the routing resources are uniform over all the FPGA component, so that any implementation of a net could be reproduced in any location of the FPGA component. For current Xilinx Virtex architecture, this assertion is true. The second need deals with the availability of "routing constraints" provided by vendor tools. Actually, directives to constraint a placement in a defined logic block area exists, but there is no way to constraint the routing resources in such a perimeter. The third need could be currently implemented through the creation of hard macro. The Xilinx Fpga Editor tool permits to save a design as a hard macro, with a fixed placement and routing. Such a hard macro could then be instantiated as any other component in a HDL description.

Thus we assert that applying also the principle of identical implementation of cells for the routing step would reduce significantly its runtime. But such a method is not viable without the possibility of constraining routing resources, as it is done for placement. In addition, such routing constraints would also be useful in other FPGA application domains: to be able to implement components with a fixed routing perimeter will allow the use of prerouted IP component in a design without routing conflicts. Such routing constraints will also facilitate the FPGA partial runtime reconfiguration, since it guarantees that the routing resources used by a reconfigured area of the FPGA would not result in a routing conflict with other areas.

References

[1] A. Koch. Regular datapath on Field-Programmable Gate-Arrays. *PhD thesis, Technical University Braunschweig,* 1997

[2] T. J. Callahan, P. Chong, A. DeHon and J. Wawrzynck. Fast module mapping and placement for datapath in fpgas. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays* ACM, 1998

[3] J. M. Emmert, A. Randhar and D. Bhatia. Fast Floorplanning for fpgas. In *8th International Workshop on Field Programmable Logic and Applications,* Talin, Estonia, 1998

[4] E. Fabiani, D. Lavenier. Placement of Linear Arrays. In *10th International Workshop on Field Programmable Logic and Applications,* Villach, Austria, 2000

[5] E. Fabiani, D. Lavenier and L. Perraudeau. Loop Parallelization on a Reconfigurable Coprocessor. In *WDTA'98 : Workshop on Design, Test and Applications,* Dubrovnik, Croatia, 1998

[6] Xilinx. *Alliance Series Development System Overview*, www.xilinx.com, 2000

[7] Xilinx. *Development System Reference Guide*, 2000

[8] F. de Dinechin. The price of routing in FPGAs. *Journal of Universal Computer Science*,vol. 6, pp. 227-239. Feb. 2000.

[9] S. Singh. Death of the RLOC ?. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 2000

[10] P. Guerdoux-jamet, D. Lavenier. Systolic Filter for fast DNA Similarity Search. In *ASAP'95: International Conference on Application Specific Array Processors,* Strasbourg, France, July 1995

[11] D. Lavenier. FPGA Implementation of the k-means Clustering Algorithm for Hyper-Spectral Images. Los Alamos Unclassified Report 00-3079, July 2000