Koen Bertels
João M.P. Cardoso
Stamatis Vassiliadis (Eds.)

LNCS 3985

Reconfigurable Computing: Architectures and Applications

Second International Workshop, ARC 2006
Delft, The Netherlands, March 2006
Revised Selected Papers

A New Approach to Assess Defragmentation Strategies in Dynamically Reconfigurable FPGAs*

Manuel G. Gericota[1], Gustavo R. Alves[1], Luís F. Lemos[1], and José M. Ferreira[2]

[1] Department of Electrical Engineering - ISEP,
Rua Dr. António Bernardino de Almeida, 4200-072 Porto, Portugal,
{mgg,gca,lfl}@isep.ipp.pt,
[2] Department of Electrical and Computer Engineering - FEUP,
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal,
jmf@fe.up.pt

Abstract. Fragmentation on dynamically reconfigurable FPGAs is a major obstacle to the efficient management of the logic space in reconfigurable systems. When resource allocation decisions have to be made at run-time a rearrangement may be necessary to release enough contiguous resources to implement incoming functions. The feasibility of run-time relocation depends on the processing time required to set up rearrangements. Moreover, the performance of the relocated functions should not be affected by this process or otherwise the whole system performance, and even its operation, may be at risk.
Relocation should take into account not only specific functional issues, but also the FPGA architecture, since these two aspects are normally intertwined. A simple and fast method to assess performance degradation of a function during relocation and to speed up the defragmentation process, based on previous function labelling and on the application of the Euclidian distance concept, is proposed in this paper.

1 Introduction

Field Programmable Gate Arrays (FPGAs) experienced a considerable evolution in the last two decades. Shorter reconfiguration times and the new features introduced recently, such as run-time partial reconfiguration and self-reconfiguration, made possible the implementation of the concept of virtual hardware defined in the early 1990s: the hardware resources are supposed to be unlimited and implementations that oversize the reconfigurable logic space available are resolved by temporal partitioning [1].

Generally, an application comprises a set of functions that are predominantly executed in sequence, or with a low degree of parallelism, in which case their

* This work is supported by an FCT program under contract POSC/EEA-ESE/55680/2004

simultaneous availability is not required. Functions may be swapped in real time, becoming operational only when needed and being substituted if their availability is no longer required. However, when the logic space of an FPGA is shared among several functions belonging to a number of different applications, each with its own requirements in spatial and temporal terms, fragmentation of the logic space may occur [2]. The solution to this problem is to consolidate unused areas within the FPGA without halting the operation of currently running functions. If a new function cannot be allocated immediately due to lack of contiguous free resources, a suitable rearrangement of a subset of the executing functions must be implemented to overcome the problem.

In general, there is a tendency to model the FPGA as a regular array structure and to regard defragmentation as a strictly packing problem [3–5]. While in the first generations of FPGAs this assertion was true regarding the CLBs position inside the array, it was inaccurate when other resources were considered. The presence of dedicated routing resources available to enhance specific applications (like counters and adders), which have a tremendous impact on function performance, were mainly responsible for this inaccuracy. The problem was aggravated in more recent generations by the introduction of memory blocks and of dedicated Digital Signal Processing (DSP) blocks distributed among the FPGA array.

In an FPGA, the access to the reconfiguration mechanism is independent from the operation of the running functions. Therefore, defragmentation may be implemented as a background process, running concurrently with the operation of currently implemented functions, without disturbing or impairing them, instead of just when a new incoming function is claiming area to be implemented. As a result, waiting times will be reduced and the overall system performance improved. A metric to determine when to perform defragmentation is proposed in [6].

2 Labelling functions

To enhance the performance of specific types of functions, FPGA architectures present some special features, like dedicated carry lines to increase speed of arithmetic functions (e.g. counters or adders). In the architecture of Virtex FPGAs from Xilinx, which were used during the experimental phase of this research work, these lines span the FPGA vertically, enabling only the interconnection of vertically adjacent CLBs. The use of dedicated carry lines, with very low propagation delays (in the order of a few picoseconds), enabled us to achieve an operating frequency of circa 145 MHz for a 24-bit binary counter implemented on an XCV200, and 450 MHz in the case of an XC4VFX12.

However, the maximum frequency of operation decreased dramatically if one or more of these dedicated carry lines were substituted by generic interconnection resources. Figure 1 shows how the maximum frequency of operation of the 24-bit counter decreases, in percentage terms, for both FPGAs, as a function of the number of dedicated carry lines that are broken. Notice that despite belong-

ing to different FPGA generations, the counter exhibited a similar behavior in both cases. From this simple example it becomes obvious that it is mandatory for any defragmentation procedure to take into account both architectural and functional aspects, before making any function relocation decisions.

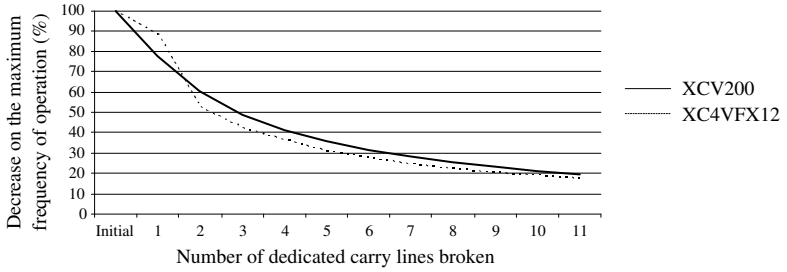

Fig. 1. Performance degradation

If this function is active, i. e. if the function is currently being used by an application, dynamic relocation techniques, as those described in [2], must be applied during the defragmentation procedure, otherwise the function operation will be temporarily halted, which may consequently disrupt the operation of the whole system. Moreover, relocation must be performed keeping as much as possible the vertical orientation of the function placement. Besides, no more than one of the dedicated carry lines linking vertically adjacent CLBs should be broken. This means that only one adjacent CLB may be relocated at a time and that vertical adjacency must not be lost.

These two pieces of information, verticality and adjacency, are essential to enable an efficient defragmentation and should be attached as a label to the function configuration file.

To evaluate the influence of changes in shape and in relative position of CLBs in different functions, the same type of experiments were performed over a subset of the ITC'99 benchmark circuits [7]. The objective was to determine which parameters are involved in the performance degradation of particular functions so as to be able to formulate a simple set of rules to support logic space management. The experiments consisted of displacing vertically and horizontally each one of the functions and changing its relative shape, from a square-like shape to a rectangular one and rotating it 90°. These stressing conditions helped to bring into evidence which parameters are mostly responsible for performance degradation, when functions are moved around. The results of the experiments are summarised in table 1. It is evident that circuits B04, B05, B07, B11, B13 and B14 experienced considerable performance degradation when the relocation of the whole function was carried out horizontally, since all these functions use dedicated carry lines on their implementation. This conclusion confirmed the extremely high importance of keeping intact dedicated carry lines.

Table 1. Evaluation of function performance degradation due to reshaping

Circuit reference	Number of occupied CLBs	Variation in the maximum frequency of operation (%)	
		Vertical relocation	Horizontal relocation
B01	6	−5.5	0.0
B02	1	0.0	0.0
B03	11	−1.9	−4.9
B04	54	−6.1	−29.3
B05	103	−17.3	−36.9
B06	5	−2.7	0.0
B07	31	−23.6	−37.8
B08	17	−5.8	−5.8
B09	12	−1.8	−4.9
B10	20	−7.5	−7.6
B11	39	−10.5	−36.0
B12	119	0.0	−1.2
B13	37	−4.3	−42.8
B14	333	−13.5	−47.8

Some functions, like B11, comprise hundreds of gates but have a reduced number of carry lines. In this case, it is necessary to have a simple method to quickly identify the columns that contain these lines. Otherwise, the ability to reshape the function during defragmentation will be heavily constrained. The label attached to the function configuration file must indicate the relative position inside the function of the column that must be left as it is.

3 Proximity vectors

The first circuit on the list, B01, exhibits a different behaviour when compared to those previously observed. Horizontal relocations do not degrade its performance, most probably because it uses no carry lines. However, vertical relocations decrease its maximum frequency of operation.

The most noticeable aspect of its implementation was the great number of high fanout signals that leave the CLBs located in the central column. To reduce propagation delays these CLBs were strategically positioned by the design tools in the centre of the function floorplan. If the circuit is shifted horizontally, the relative position of the central CLBs is not affected. However, if the central location of these two CLBs is changed, propagation delays will increase and the maximum frequency of operation of this function will decrease. This hypothesis was confirmed by rotating the function 90° and relocating it in only one CLB column.

The CLBs with output signals that drive a large number of inputs, despite keeping their central location, are now, on average, far from their destination

inputs than they were before. This means an increase on the propagation delay not only due to an increase in the length of interconnection lines, whose impact is minor, but mainly because each line has to cross a greater number of Programmable Interconnect Points (PIPs). The small distance that signals have to cross means that small segments linking adjacent routing arrays are used to route them. Therefore, if a new segment has to be added, a new PIP has also to be used, which leads to a noticeable increase in the propagation delay. The other benchmark circuits exhibited a similar behaviour.

A systematic analysis of this problem led to the development of a new method to assess the impact of relocating CLBs whose output signals drive a large number of inputs: the application of the concept of *proximity vectors*, a vector associated to each interconnection and linking the CLB source to the CLB destination.

The length of each vector, called *proximity factor*, is expressed in CLB units and calculated as the modulus of the distance between the CLB source and the CLB destination:

$$|f_{px}| = \sqrt{r^2 + c^2} \tag{1}$$

where:

r=CLB destination row - CLB source row

r=CLB destination column - CLB source column

If the sum of all proximity vectors of one CLB output is minimised (eq. 2), the proximity factor associated to that output will also be minimised. This corresponds, in terms of the propagation delay of a given output, to the best position of that CLB inside the function.

$$|f_{pox}| = min \sum_{d} (f_{p1}, f_{p2}, ..., f_{pd}) \tag{2}$$

When relocating the CLB, if the proximity factor F_{pox} increases, then performance degradation of the function will occur. Generically, we can say that minimising each output proximity factor of a function results in the minimisation of its global proximity factor, which corresponds to the best performance (maximum frequency of operation). The application of this concept to the remaining circuits showed a consistent reproduction of results, confirming the initial hypothesis.

The concept of proximity vectors is based on the application of the Euclidian distance measurement to each net. Since routing is constrained to horizontal and vertical wires, it seems, at first, that the use of the Manhattan distance measurement would be more reasonable. However, a series of experiments performed to compare the use of the two distance measurement methodologies showed a greater correlation between maximum frequency of operation and the Euclidian distance measurement.

The main advantages of this approach are as follows:

1. It can be easily automated and integrated in existing design tools;

2. The computation time of the proximity vectors is extremely low when compared to previous proposed approaches, since only the nets that will be affected by relocation need to have their proximity factor (before and after the relocation) calculated;
3. There is no need to perform a complete analysis of the function performance after each CLB relocation, since, if the minimisation of the global proximity factor of the CLB was assured, the minimisation of the global proximity factor of the overall function is assured, and therefore, no performance degradation occurs.

All these factors enable this method to be used at run time to quickly and reliably assess the strategy used to manage the defragmentation procedure.

4 Conclusions

This paper presents a new approach to assess the performance degradation introduced by the relocation of functions during defragmentation procedures applied to dynamically reconfigurable FPGAs. The proposed approach is able to guide the defragmentation procedure in a reliable and fast way enabling the run-time relocation of running functions, timely releasing enough contiguous space for new incoming ones while avoiding performance degradation.

Current work is aimed at evaluating the influence of other array heterogeneities (that are present in more recent generations of FPGAs), namely memory blocks and dedicated DSP blocks. The influence of hardware embedded processors and the rule they may play in the implementation of defragmentation strategies on run time reconfigurable systems will also be addressed in the future.

References

[1] Ling, X.-P., Amano, H.: WASMII: a Data Driven Computer on a Virtual Hardware, Proc. 1st IEEE Workshop on FPGAs for Custom Computing Machines (1993), 33–42

[2] Gericota, M. G., Alves, G. R., Silva, M. L., Ferreira, J. M.: Run-Time Defragmentation for Dynamically Reconfigurable Hardware, in: New Algorithms, Architectures and Applications for Reconfigurable Computing. Springer (2005) 117–129

[3] Teich, J., Fekete, S., Schepers, J.: Compile-time optimization of dynamic hardware reconfigurations, Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (1999) 1097–1103

[4] Handa, M., Vemuri, R.: An efficient algorithm for finding empty space for online FPGA placement, Proc. Design, Automation Conf. (2004) 960–965

[5] Vinh, P. C., Bowen, J. P.: Continuity Aspects of Embedded Reconfigurable Computing, Innovations in Systems and Software Engineering: A NASA Journal, Springer-Verlag, Vol. 1, No. 1, (2005) 41–53

[6] Ejnioui, A., DeMara, R. F.: Area Reclamation Strategies and Metrics for SRAM-Based Reconfigurable Devices, Proc. Intl. Conf. on Engineering of Reconfigurable Systems and Algorithms (2005)

[7] Politcnico di Torino ITC'99 benchmarks. Available at: http://www.cad.polito.it/tools/itc99.html