Placement of Linear Arrays

Erwan Fabiani and Dominique Lavenier

IRISA, Campus de Beaulieu,
35042 Rennes Cedex, FRANCE
{efabiani, lavenier}@irisa.fr

Abstract. This paper presents a methodology for mapping linear processor arrays onto FPGA components. By taking advantage of regularity and locality properties of these structures, a placement is pre-defined, allowing vendor tools to skip this phase and produce fast and optimized routing.

1 Introduction

In many compute intensive applications such as image or signal processing, time is mostly spent in executing loops. Speeding-up these applications leads to hardware implementations which directly benefit from the inherent loop parallelism. The resulting architecture is a regular array, often a systolic array, made of simple processing elements dedicated to efficiently performing the body of the inner loops [1]. The structure can either be uni or bi-dimensional, but in the following we will restrict to linear arrays only.

Implementing such nested loops onto FPGA components presents many advantages. First, the regular nature of FPGA component matches perfectly with the architecture we focus on: replication of identical regularly interconnected processing elements. Second, the best uses of FPGA boards (from a performance point of view) have been demonstrated on many compute intensive applications, as illustrated by the numerous applications implemented on the PAM boards [2]. Third, new advanced microprocessor architectures tend to incorporate reconfigurable resources in their data-path. Parallelizing loops on these specific areas is a very attractive way to efficiently exploit reconfigurable computing. Globally, there are three steps as described in [3]:

- **Parallelization:** This step consists in deriving regular array architectures from loop specifications or equivalent formal description such as systems of *affine recurrence equations*. The ALPHA language, developed at IRISA allows the programmer to explore transformations needed for systematic derivation of regular arrays and for automatic parallelization [4].
- **Partitioning:** Since the available reconfigurable resources may not support the entire array, transformation of the architecture is required : splitting the array into sub-arrays or clustering groups of processing elements. The automating of this task is still ongoing research and is not yet fully resolved.

– **Physical Mapping:** This last step maps the architecture on the reconfigurable support. From a RTL description (provided by the previous stages), one must find the best mapping both in term of speed performance and area occupation. This is actually a very time-consuming step which tends to become longer as the FPGA components grow in complexity.

The work presented in this paper deals with the last stage. It focuses on reducing the place-and-route process involved in the physical mapping task by taking advantage of the regular nature of the array we want to map.

2 Regular Place-and-Route Foundation

Place-and-Route steps are very time consuming, especially with the larger FPGA components. This is mainly due to the algorithmic techniques (such as simulated annealing) used for finding reasonable solutions. The advantage of these techniques are their generality: they provide relatively good solutions whatever the structure of the designs. In our case, as we try to shift towards software compilation requirement, the major drawback is definitely the computation time.

One way to limit this time is to provide a pre-defined placement and, of course, the best as possible to optimize the routing phase. The methodology we developed for mapping regular arrays onto FPGA components is mainly based on this idea. Our thesis is that placing an array of processing elements according to its regular and locality properties brings three major improvements over usual place-and-route techniques: (1) the placement time is drastically reduced; (2) the routing time is optimized; (3) the frequency is increased. Our placement strategy for taking advantages of these improvements is based on the following rules:

1. Signals which belong to a same processor have their sources placed in a same restricted area. This implies a reduction of the placement, the routing and the delay time.
2. Identical processors have identical placement: the placement focuses only on one processing element and is replicated over the FPGA component. The time is thus independent of the number of processing elements.
3. Neighboring processing elements are close to each other. Again,the expected benefits are a reduction of the placement, the routing ant he delay time.

A few experiments have been carried out to validate this thesis. Basically, we compare the time to place-and-route a design with and without placement directives. Several linear array designs have been tested using the PPR Xilinx router tool for the XC40O0 family. We observe that the placement phase is more time consuming than the routing phase, and shortening this step results in a significant speed-up (3 on average), even if the routing phase, in some cases increases. We also observe that it does not lead to degraded clock frequency.

3 Regular Place-and-Route Strategy

Figure 1 details the place-and-route environment for mapping regular arrays onto FPGA components. The input and output of FRAP are written in a same structural description, respectively without and with placement directives. The regular placement is performed with the FRAP tool and acts in three steps:

1. All possible shapes for a processing element are generated by combining all shapes of its sub-components.
2. A full *snake* placement of the linear array is determined using the processing element shapes previously computed.
3. The final placement of the processing elements are performed according to their shapes.

From the output of FRAP an EDIF file is generated and input to the vendor place-and-route tools. Since the placement is fully specified, the computation time is reduced to roughly the time for routing the FPGA component.

Steps 1 and 3 deal with processing element placement. We consider those elements rather small, that is a few operators essentially coming from a library, and that finding a good placement is a fast and non critical process.

In step 2, the problem is to place a linear array on a bi-dimensional FPGA structure. The only way to keep two neighbor processing elements close to each other is to implement a snake-like arrangement of the array. The determination of the snake-like arrangement proceeds in two phases [5]: (1) divide the FPGA area in sub-areas that we call *convenient areas*, and (2) for each convenient area, place a maximum number of processing element in a snake-like fashion.

The second phase is solved using the knapsack metaphor [6] allowing the use of different kind of shapes for the processing elements.

Figure 2 is an illustrating example of the result of the FRAP placement. The full FPGA area has been partitioned into three convenient areas. In the convenient area 1, an horizontal snake is made of two different segments, segment of shape A and segment of shape B. The convenient area 2 is also an horizontal snake made only with a processing element of shape A. The convenient area 3 is a vertical snake made of processing elements of shape C. The overall placement requires of determining placements for the 3 different shapes of the processing element.

4 Conclusion and Future Works

We have presented a strategy for placing linear regular arrays onto FPGA components. This strategy uses the knapsack technique and provides fast placement compared to the vendor tools. The speed-up comes mainly from the regular nature of the architecture we focus on, that is, linear arrays of identical processors on which a two-level placement is achieved: (1) a cell-level placement and (2) an array-level placement.

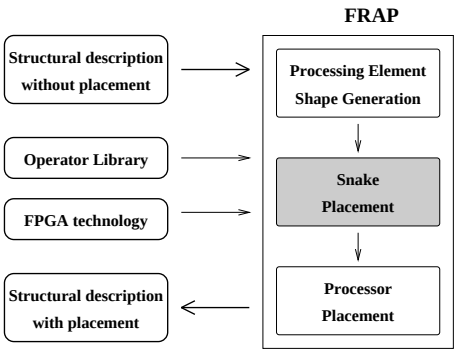

Fig. 1. The FPGA Regular Array Placer (FRAP): the description given as an input is processed and output with placement directive annotations.

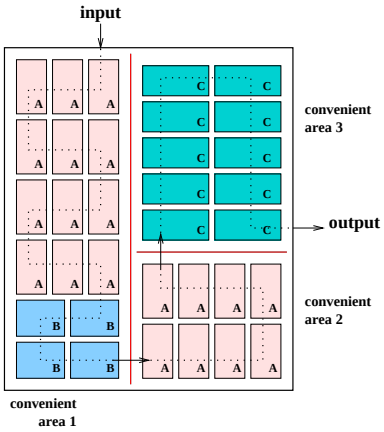

Fig. 2. illustrating placement produced by FRAP

Even if we can drastically shorten the placement step, the overall place-and-route process remains too long to be included into a compiling framework. It may takes a few tens of minutes up to a few hours to achieved a suitable routing, that is definitely too long for programmers who are used to a faster compiling process. Consequently, the next step is to shorten the routing phase.

As for placement, this step can benefit from regular architecture by duplicating routing pattern of the processor cells. Unfortunately, unlike for placement, this strategy cannot be implemented through a few "routing" directives. It requires a detailed knowledge of the routing resources of the target FPGA as well as direct access to the programming of the routing switches.

References

1. P. Quinton and V. V. Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing,* 1(2), 1989.
2. J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati and P. Boucard. Programmable Active Memories: Reconfigurable Systems Come of Age. *IEEE Transactions on VLSI Systems,* 4(1), 1996.
3. 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
4. P. Le Moenner, L. Perraudeau, P. Quinton, S. Rajopadhye, T. Risset Generating Regular Arithmetic Circuits with ALPHARD. *MPPS'96: Massively Parallel Computing Systems,* Ischia, Italie, 1996.
5. E. Fabiani and D. Lavenier. Using knapsack technique to place linear arrays on FPGA. Research report 1335, IRISA, June 2000.
6. S. Martello and P. Toth. Knapsack Problems: Algorithms and Computer Implementation. *John Wiley and Sons,* 1990.