GENERATION AND EXPLORATION OF RECONFIGURABLE ARCHITECTURES USING MATHEMATICAL PROGRAMMING

Alastair M. Smith, George A. Constantinides, and Peter Y. K. Cheung

Department of Electrical and Electronic Engineering,
Imperial College London
email: {alastair.smith, g.constantinides, p.cheung} @imperial.ac.uk

ABSTRACT

The purpose of this paper is to detail a high-level analytical modelling and optimisation environment for mixed-granularity Field Programmable Gate Arrays (FPGAs). The work carried out for the purposes of this study involves the creation of an analytical framework that can be used to optimise the design of a reconfigurable device for a set of benchmarks. The strengths of this approach are the simultaneous placement, module selection and architecture generation. In this paper, the problem is cast as a formal optimisation, and may be solved using existing optimisation tools. In addition, the approach is adapted into an heuristic for larger benchmark sets. The design space is explored by examining the trade-offs between area, speed and flexibility, and some comparisons to commercial architectures are drawn.

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) are commonly used for high throughput computation. Traditionally, FPGAs have consisted of Look-Up Tables (LUTs) capable of performing any four input logic function. The recent trend for commercially produced FPGAs has been to introduce more heterogeneous features in order to speed up computation or take advantage of greater logic density, such as DSP blocks and RAM. There has been considerable research into exploring the architectures of homogeneous LUT-based FPGA devices [1, 2]. This has concentrated on exploring the nature of the LUTs, for instance how many inputs they use, and how they are locally interconnected. In this paper, the emphasis is on exploring architectures by examining the ratios and physical placements of the different components that are found in heterogeneous devices.

When designing a new device architecture it is common for the architects to have a base-line parameterisable structure from which many different architectures can be generated, for example by varying the mix of routing resources [3]. A variety of possible architectures are then simulated, with reference designs placed and routed in each architecture using heuristics such as simulated annealing. The final architecture will be one from this set that best suits the area, speed and power consumption metrics for all designs. By using linear programming, this work shows how it is possible to simultaneously place benchmarks and generate heterogeneous architectures, *as well as* perform module selection for given computational structures in a benchmark; *e.g.* decide whether a ROM should be implemented in LUTs or in an embedded component. This paper proposes an approach that allows all three to be performed concurrently, leading to highly optimised architectures, eliminating both the need for exhaustive testing on a set of architectures and the dependence on heuristic parameters. Incorporating module selection into our approach also allows the trade-off of the speed and area advantages of dedicated hardware with the flexibility of more general logic.

The main contributions of the paper can be summarised as follows:

- To our knowledge, the first formulation of heterogeneous FPGA architecture exploration as a formal optimisation problem considering simultaneous placement, module selection and architecture generation.
- A novel integer linear programming formulation addressing this optimisation problem.
- A novel rounding-based heuristic approach to the optimisation problem suited to large benchmarks.

2. RELATED WORK

There has been much work in the general field of reconfigurable computing relating to architectural explorations. The work presented here concentrates on heterogeneous FPGAs containing a mixture of fine-grain and coarse-grain functional units.

In [4] architectures containing embedded memories as well as LUTs are explored. A set of benchmarks is se-

This work has been funded by the EPSRC and Royal Society Grant (2004/R1).

lected with the aim of minimising the area of the architectures produced while maintaining a minimum circuit delay. The benchmarks are mapped to 4-input LUTs to calculate the minimum circuit delay. An attempt is then made to find the best size of embedded memory block by applying algorithms that pack logic into the embedded memories. In [5] an abstract model is used to look at a set of predefined architectures containing specific functional units. These are arranged hierarchically so that the routing delay between components in the same level of hierarchy is the same. Applications are then mapped onto the set of architectures, and the architecture that implies the least communication delay is derived, the idea being to assign computations with data dependencies to resources in the same hierarchical level. While an interesting piece of work, only the set of architectures supplied by the user are examined. Synthesis of circuits to fine-grain resources is also neglected.

In [6] heterogeneous coarse-grain reconfigurable devices for the purposes of encryption algorithms are explored. The available resources on a device are examined to determine if they can be re-used in order to minimise the area consumed by the implementation of the algorithms. The aim is to look at the entire design space, but no account is taken of module selection for different computational structures. There has also been work exploring similar architectures for the purpose of comparing ASICs to FPGAs [7]. The result of this work is a tool that generates domain specific architectures. However, fine-grain resources such as LUTs and multiplexers are not included within the architectures.

In contrast, this work concentrates on exploring the design space of FPGAs containing a mixture of different fine-grain and coarse-grain resources. In particular, the mix of different resource types, and the effect this has on the benchmark performance in each architecture is examined. Synthesis tools are used to map the various computation structures found in the benchmarks to different functional unit types found in commercial FPGAs in order to analyse the area and timing characteristics, and a routing model is used to estimate the inter-component delays. The analytical formulation described in the next section is then used to optimise the architecture for a benchmark set under a given area constraint.

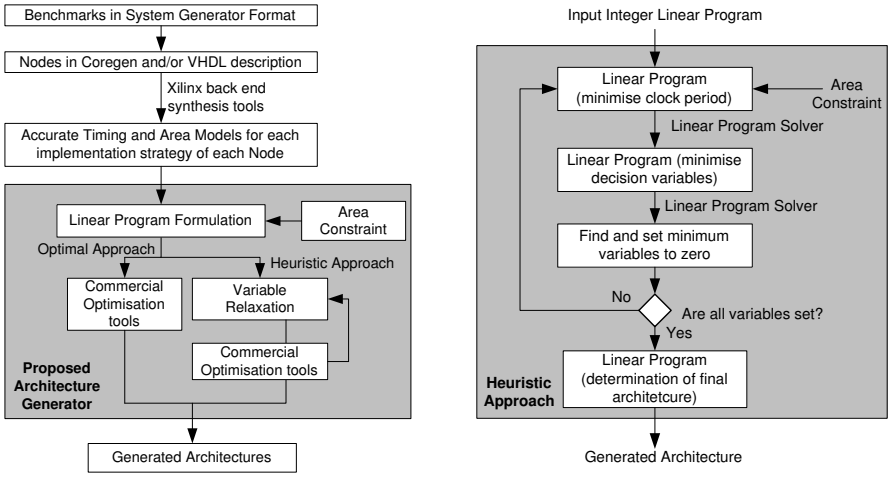
3. DESIGN FLOW AND PROBLEM FORMULATION

The focus of this work is on the DSP domain, and as a consequence, the benchmarks used to test the architectures in this study have been developed in Xilinx's System Generator for MATLAB [8]. Figure 1(a) shows the design flow and how our tool interacts with existing software. The optimisation tool developed transforms the benchmark circuits into a dataflow graph (DFG) format. The benchmarks consist of a set of node types that represent various computations. The tool optimises the architecture for the set of benchmarks supplied.

The architecture generator uses linear programming in order to combine the problem of placement and module selection. Nodes are allowed to be constructed from the various resource types available; the linear programming approach allows this to be done within a unified framework.

The procedure begins by assessing the timing and resource requirements of each computational node under a variety of different implementation strategies, through use of the Xilinx synthesis tools. The term *implementation strategy* refers to whether the node is constructed from LUTs, embedded RAMs or embedded multipliers. Using the vendor tools allows a comprehensive architectural exploration, as full advantage can be taken of special architectural features included in commercial devices. The tools provide near-exact resource usages and intra-node timing models for each implementation strategy of each node. The inter-node delays are ascertained by using a simple linear model in which the cross-chip delay is proportional to the Manhattan distance. The constants of proportionality in this model were obtained by modelling the delay between two circuit elements in a Virtex 2 chip. The optimal architecture is then considered to be one that optimises a measure of the different benchmark clock periods defined below.

The clock period of a circuit can generally be defined as the maximum delay between registers. The clock period will vary with implementation strategy of certain nodes, and consequently with the specified architecture. In order to relate the clock period values between benchmark circuits, the concept of 'relative clock period' is introduced. The relative clock period is defined as (1), where C_b represents the relative clock period of benchmark b, T_{sb} represents the clock period of the benchmark given a particular architecture s. T_{cb} represents the clock period of the benchmark given the optimal set of components for the given area constraint. This means that the speed of each benchmark is normalised by dividing by its optimal speed under the specified area constraint (*i.e.* the best components for each benchmark circuit are selected). C_b can be thought of as *a measure of the speed lost as a result of introducing reconfigurability*; a relative clock period of 1.2 implies that the circuit is 20% slower than it could be in the same area. Note that C_b is a measure of the *worst case* relative clock period over all benchmarks, which is chosen as an average can mask poor performance in one benchmark with good performance in another. A strength of the approach is that it would be possible possible to perform either worst case or average with no change to the optimisation procedure. The overall goal of the optimisation is to minimise the maximum value of C_b over all benchmarks.



(a) The overall design flow (b) The heuristic approach

Fig. 1. Figure (a) shows the overall design flow using the ILP method, and Figure (b) shows the flow of the heuristic, as described in Section 3.2.

$$C_b = \frac{T_{sb}}{T_{cb}} \tag{1}$$

3.1. Linear Programming Formulation

Every benchmark b can be specified as a dataflow graph. A dataflow graph is a pair (V_b, E_b) where V_b specifies the set of vertices (or nodes) and E_b specifies the set of edges between nodes. $(u, v) \in E_b$ denotes an edge produced at node u and consumed at node v. The nodes represent computations and the edges represent the data dependencies or dataflow between nodes. The benchmark set is denoted B, and each $b \in B$ is a dataflow graph representing one benchmark circuit.

As mentioned, the clock period of each benchmark is measured by taking the maximum delay between registers, inputs and outputs. Register outputs and circuit inputs can be considered source nodes while register inputs and circuit outputs can be considered sink nodes. The maximum delay of the circuit is thus the longest path between source and sink nodes. For ease of computation the benchmarks are transformed so that they only have one source and sink node. The delay of each node is obtained by individually synthesising each implementation strategy for each node and automatically running a timing analysis. After having transformed the graph, and ascertained the delays of each node, an annotated directed acyclic graph remains.

In the optimisation process, the objective is to minimise the maximum relative clock period C_{max}. The constraints on C_{max} are given by (2).

$$\forall\, b \in B \qquad C_{max} \geq C_b \tag{2}$$

The maximum clock period of each benchmark is governed by Bellman's equations (3), where T_u represents the time during a clock cycle at which the inputs to node u become valid and the edge weight $w(u, v)$ is a function that accounts for the combinatorial delay of node u and the routing delay between nodes u and v. The source node starts at time 0, *i.e.* $T_{source} = 0$. However, Bellman's equations are inadequate for this problem, as there is no way of representing implementation strategies of different nodes. Thus a modified version is introduced, given by (4-5).

$$\forall\, v \in V_b \quad T_v = \max_{(u,v)\in E_b} (T_u + w(u, v)) \tag{3}$$

$$\forall\, (u, v) \in E \quad T_v \geq T_u + r_{uv} + \sum_{i \in I(v)} d_{ui} c_{ui} \tag{4}$$

$$\forall\, u \in V_b \quad \sum_{i \in I(v)} d_{ui} = 1 \tag{5}$$

Here r_{uv} represents the portion of delay due to routing, and $I(v)$ represents the set of implementation strategies for node v, for example {embedded multiplier, LUT-based, embedded RAM}. The combinatorial delay of node u when implemented in strategy i is represented by c_{ui}. The binary decision variable d_{ui} has the value 1 iff implementation strategy i of node u is used. The linear program also has the constraint that each node should only be implemented one way (5). The routing delay is related to the physical placement on the device, and is given by (6-12).

$$\forall\,(u,v) \in E_b$$

$$r_{uv} = k_1 + k_2(|x_v - x_u - w_u| + |y_u - y_v|) \quad (6)$$

$$|x_v - x_u - w_u| \geq x_v - x_u - w_u \quad (7)$$

$$|x_v - x_u - w_u| \geq x_u + w_u - x_v \quad (8)$$

$$|y_u - y_v| \geq y_u - y_v \quad (9)$$

$$|y_u - y_v| \geq y_v - y_u \quad (10)$$

$$\forall\, u \in V_b \qquad w_u = \sum_{i \in I(v)} w_i d_{ui} \quad (11)$$

$$\forall\, u \in V_b \qquad h_u = \sum_{i \in I(v)} h_i d_{ui} \quad (12)$$

Here k_1 and k_2 are the constants of proportionality used in the linear routing model. The coordinates of the bottom left corner of node u are (x_u, y_u), and the node has width w_u and height h_u. The w_u term in (6) models the routing delay from the bottom right of node u. Note that (6) is nonlinear but may be linearised as (7-10). Equations 11 and 12 are used to give the node the appropriate height for the appropriate implementation strategy, for example a large multiplier implemented in LUTs is typically larger than its equivalent embedded version.

3.1.1. Constraining Node Placement

The architectures under exploration are those in which the resources are grouped into columns; nodes can only be implemented in a particular strategy when placed in a given region. This is similar to some of the most recent devices from Xilinx, and formulating the problem this way allows the placement to be related between benchmarks.

Finally, it is necessary to introduce constraints to prevent region overlap. This overlap can be visualised by observing Figure 2(a). To prevent these regions overlapping, clearly $a_i > b_j$ or $a_j > b_i$. These 'or' constraints are linearised by introducing two binary variables δ_{qr1} and δ_{qr2} as in (13-16). The equations described here have been extended to account for the different region types. In this case I is the set of implementation strategies, R_i is the set of regions of strategy i and A is the area constraint. The aspect ratio of the device is assumed to be 1, giving rise to the $\sqrt{A}$ term. The left and right boundaries of region r of resource type j are specified by a_{jr} and b_{jr} respectively. In these equations, if δ_{qr1} is equal to one then equation 13 is trivially satisfied and thus a_{jr} must be greater than b_{iq}, *i.e.* region jr must be to the right of region iq. Similarly, if δ_{qr2} is equal to one then equation 14 is trivially satisfied, and thus a_{iq} must be greater than b_{jr} *i.e.* region iq must be to the right of region jr.

$$\forall i \in I,\ \forall q \in R_i,\ \forall j \in I,\ \forall r \in R_i,\ (i \neq j)$$

$$\sqrt{A}\delta_{qr1} + a_{iq} - b_{jr} \geq 0 \quad (13)$$

$$\sqrt{A}\delta_{qr2} + a_{jr} - b_{iq} \geq 0 \quad (14)$$

$$\delta_{qr1} + \delta_{qr2} \leq 1 \quad (15)$$

$$\delta_{qrk} = 0, 1, \quad k = 1, 2 \quad (16)$$

To set the widths of the regions, and placement of the nodes within these regions constraints (17-19) must also be added.

$$\forall\, i \in I,\ \forall\, u \in V_b \qquad \sum_{r \in R_i} d_{uir} = d_{ui} \quad (17)$$

$$\forall\, i \in I,\ \forall\, r \in R_i,\ \forall\, u \in V_b$$

$$\sqrt{A}(1 - d_{uir}) + x_v - a_{ir} \geq 0 \quad (18)$$

$$\sqrt{A}(1 - d_{uir}) - x_v - w_{ui} + b_{ir} \geq 0 \quad (19)$$

In other words, the right hand boundary of a given region must be greater than the x coordinate of any node plus its width in that particular implementation strategy. Here d_{uir} is the binary decision variable that takes the value one iff node u is implemented in region r of implementation strategy i, w_{ui} is the width of node u when implemented in strategy i.

Nodes must also be prevented from overlapping within these regions. In this case, constraints similar to those above can be introduced. The above constraints can be thought of as a modified version of the 1-dimensional and 2-dimensional knapsack problems, where the sizes of the nodes are not known *a priori*, and the objective function relates to node interconnect and combinatorial delay. This can be visualised as in Figure 2(b). In this case at least one of (20) must be true.

$$x_u + w_u \leq x_v,\ y_u + h_u \leq y_v,$$
$$x_v + w_v \leq x_u,\ y_v + h_v \leq y_u \quad (20)$$

Linearising this disjunction leads to (21-26), where (25) ensures that at least one of (20) is true.

$$\forall\,(u,v) \in V_b$$

$$\sqrt{A}\delta_{uv1} + x_v - x_u - w_u \geq 0 \quad (21)$$

$$\sqrt{A}\delta_{uv2} + y_v - y_u - h_u \geq 0 \quad (22)$$

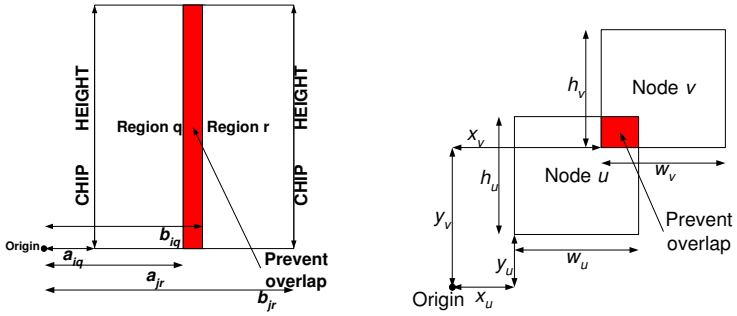
$$\sqrt{A}\delta_{uv3} + x_u - x_v - w_v \geq 0 \quad (23)$$

$$\sqrt{A}\delta_{uv4} + y_u - y_v - h_v \geq 0 \quad (24)$$

$$\delta_{uv1} + \delta_{uv2} + \delta_{uv3} + \delta_{uv4} \leq 3 \quad (25)$$

$$\delta_{uvj} = 0, 1, \quad j = 1, 2, 3, 4 \quad (26)$$

The combination of the above constraints allows the determination of the optimal architecture. The constraints are



(a) Preventing overlap of regions. (b) Preventing overlap of nodes.

Fig. 2. An illustration of the placement constraints in the linear programming formulation

added in for each benchmark, with the region constraints allowing the architecture to be shared between benchmarks. The optimisation problem, once cast in linear form, can be solved through a linear program solver such as ILOG CPLEX [9].

Even if only one benchmark is considered, the number of binary variables modelling node-node placement (21-26) is $2n(n-1)$, n is the number of nodes in a given benchmark circuit. A consequence of this is that the run time of the optimisation procedure explodes for large benchmarks and the proposed linear programming approach becomes less attractive. The next section details how this is accounted for by introducing a heuristic technique that uses variable relaxation in order to find a solution within reasonable time.

3.2. Determination of Architectures Using Variable Relaxation

As mentioned previously, the run time of the ILP solver makes a direct solution of the ILP unattractive for large benchmark sets ($\gg$ 24 hours). As a consequence, it has been necessary to develop a methodology to counter this problem. The ILP framework detailed above allows the development of a heuristic approach in a structured manner. The heuristic technique developed involves a controlled relaxation of the the binary decision variables to reals in the range $[0, 1]$. Removing the integrality allows fast solution through, for example, the Simplex method [10]. The resulting optimum decision variable values may then be interpreted to steer an iterative process in which after each iteration some binary variables are fixed to zero or one. The entire heuristic procedure is shown in Figure 1(b).

In this heuristic approach, only the placement variables are relaxed. This means that the only binary decision variables in the ILP are those that determine the implementation strategy and region of each node.

At this stage, the placement variables δ_{qrx} (13-15) and δ_{uvx} (21-25) do not appear in the objective function and so take arbitrary values amongst the values that satisfy the constraints. For example, if (13-15) are satisfied with $\delta_{qr0} = 0.2$ and $\delta_{qr1} = 0.5$, they will also be satisfied with $\delta_{qr0} = 0.5$ and $\delta_{qr1} = 0.5$. Thus, in order to make placement decisions on the basis of these values, it is necessary to perform a second round of optimisation to reduce them to the minimum feasible values. The linear program is thus re-run with the relative clock period fixed to the minimum from the first stage, while minimising the sum of all decision variables as the objective.

After minimising the sum of the decision variables, each of the sets of placement variables in (16) and (25) are examined to determine which is the closest to zero. The maximum value from the minimum of all of these sets is then chosen to be rounded to zero. In this way, the critical decisions are made first leading to better quality solutions than simply choosing to round the minimum binary variable. The process is repeated until all of the relaxed decision variables have been set and a resulting architecture has been produced.

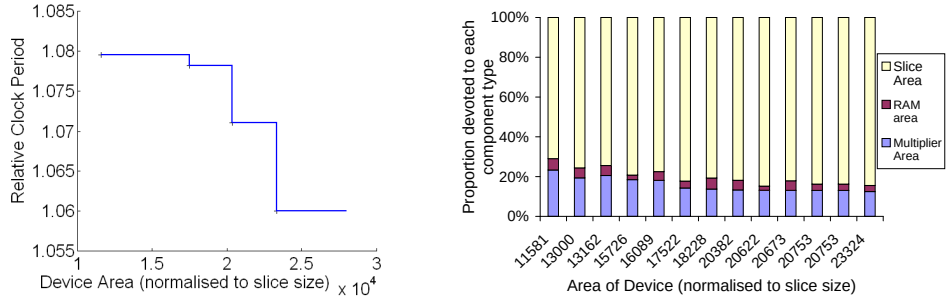
4. RESULTS

The focus of this work is DSP applications, thus a suitable set of DSP benchmarks was chosen for evaluating our approach. The benchmarks include: a 7th order LMS adaptive filter, a two dimensional 5x5 image convolution on a raster-scanned image, a multi-channel IIR filter, an ADPCM encoder, an 8th order polynomial evaluator and a Costas Loop. All benchmarks are of a similar hardware complexity when implemented in their time-optimal strategy.

In order to explore the design area/relative clock period design space, the area that the device is allowed to consume is gradually varied, and the change in relative clock period and in the resulting architecture was observed.

Figure 3(a) shows the Pareto-optimal[1] design curve for the area-speed design space for the case $|R_i| = 1$. Due to

[1]A point on the Pareto-optimal design curve represents a design which cannot have improved characteristics in one optimisation criterion, without sacrificing those in another.



(a) The optimal area-speed design curve (b) The relative proportions of the device

Fig. 3. The results for the architectural exploration

the large nature of the benchmarks, the heuristic approach has been employed. The relative clock period decreases as the area is increased, with reconfigurability penalty varying from 6% to 8%. Figure 3(b) shows that the proportion of the device devoted LUTs decreases as the area is reduced, and the corresponding proportion of multiplier area increases, although the physical area of the multiplier regions does not vary greatly.

A point worthy of note is the comparative proportions of the device. In Figure 3(b), the percentage of the generated architectures devoted to the various components is in the range 70-85% for slices, 12-23% for multipliers and 2-6% for Block RAM. This is in contrast to the Virtex 2 family of devices, that devote around 70% of their (non-routing) area to slices, 10% to Embedded multipliers and approximately 20% to Block RAM for a similar size of device. The reduced amount of area for block RAMs in the devices generated is partially due to the pipelined nature of the benchmark circuits. Memory implemented in lookup tables can often be made to run at high throughput by pipelining the designs. Moreover, the outputs of the RAM blocks often need to be routed to the portions of the device that use general logic, hence to minimise relative clock period, it is often better to implement some of the RAMs in LUTs. This decision is automatically made by the optimisation process, reducing the number of Block RAMs required for an optimal result.

5. CONCLUSION

A novel methodology for heterogeneous FPGA architecture development has been outlined, and an heuristic has been developed in order to extend the approach when the ILP approach becomes a less feasible option. The proposed approach is capable of eliminating many architectural possibilities automatically, in an efficient manner without resorting to exhaustive testing.

Future work will address the simplifying assumptions; while routing time-overhead is modelled, the model may break down in areas of extreme routing congestion. In addition, routing area is not modelled, which typically is a large proportion of the overall area in reconfigurable hardware. We are currently investigating ways of modelling routing area and congestion within the environment presented.

6. REFERENCES

[1] Li, F., Chen, D., He, L., Cong, J.: Architecture evaluation for power-efficient FPGAs. In: ACM International Symposium on Field Programmable Gate Arrays. (2003)

[2] Yan, A., Cheng, R., Wilton, S.: On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques. In: ACM International Symposium on Field Programmable Gate Array. (2002)

[3] Lewis, D., et. al.: The Stratix™ routing and logic architecture. In: ACM International Symposium on Field Programmable Gate Arrays. (2003)

[4] Cong, J., Xu, S.: Architecture evaluation for FPGAs with embedded memory blocks. Technical report, University of California, Los Angeles (1998)

[5] Bossuet, L., Gogniat, G., Philippe, J.L.: Communication costs driven design space exploration for reconfigurable architectures. In: Field-Programmable Logic and Applications. (2003)

[6] Eguro, K., Hauck, S.: Issues and approaches to coarse-grain reconfigurable architecture development. In: 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. (2003)

[7] Compton, K., Hauck, S.: Totem: Custom reconfigurable array generation. In: IEEE Symposium on Field-Programmable Custom Computing Machines. (2003)

[8] Hwang, J., Milne, B., Shirazi, N., Stroomer, J.D.: System level tools for dsp in FPGAs. In: Field-Programmable Logic and Applications. (2001)

[9] http://www.ilog.com: (ILOG CPLEX Optimisation Suite)

[10] Garfinkel, R.S., Nemhauser, G.L.: Integer Programming. John Wiley and Sons, Inc., New York (1972)