A NOVEL HEURISTIC AND PROVABLE BOUNDS FOR RECONFIGURABLE ARCHITECTURE DESIGN

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

This paper is concerned with the application of formal optimisation methods to the design of mixed-granularity FPGAs. In particular, we investigate the appropriate mix and floorplan of heterogeneous elements: multipliers, RAMs, and LUT-based logic, in order to maximise the performance of a set of DSP benchmark applications, given a fixed silicon budget. We extend our previous mathematical programming framework by proposing a novel set of heuristics, capable of providing upper-bounds on the achievable reconfigurable-to-fixed-logic performance ratio. Moreover, we use linear-programming bounding procedures from the operations research community to provide lower-bounds on the same quantity. Our results provide, for the first time, quantifications of the optimal performance/area-enhancing capability of multipliers and RAM blocks within a system context, and indicate that only a minimal performance benefit can be achieved over Virtex II by re-organising the device floorplan, when using optimal technology mapping.

1. INTRODUCTION AND BACKGROUND

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. Recent introductions into the FPGA fabric, such as DSP blocks and RAM, have been used to speed up computation or take advantage of greater logic density. Much of FPGA research has concentrated on exploring the nature of 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 number of LUT inputs [1]. 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 is selected to best suit the area, speed and power consumption metrics for all designs. By using integer linear programming (ILP), this work shows how it is possible to achieve simultaneous placement of benchmarks and generation of 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 problems 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.

There are several existing works that are related to the research presented in this paper. In [2] architectures containing embedded memories as well as LUTs are explored. A set of benchmarks is selected 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 [3] 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. Their aim is to explore 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 [4]. 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.

More recent advances have attempted to quantify the gap

This work has been funded by the EPSRC (UK) under grant number EP/C549481/1 and under the EPSRC DTA scheme.

between heterogeneous FPGA architectures and ASIC implementations [5]. Synthesis tools are used to create both ASIC and heterogeneous FPGA implementations of algorithms in order to evaluate the performance gains of ASIC over FPGA. In contrast, the work presented in this paper concentrates on generation of heterogeneous FPGAs, using methods that minimise synthesis effects by using mathematical programming. This allows the determination of performance bounds and true optima.

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

- A new set of heuristics for the solution of the *combined* architecture generation, floorplanning, and module selection problem, for which an ILP was first presented in [6].
- Provable bounds on the distance to optimality of both the generated architectures and a family of commercial architectures, quantifying the optimal area/speed advantages for a class of reconfigurable architectures.

2. DESIGN FLOW AND PROBLEM FORMULATION

The following section describes certain key features of the ILP formulation of the combined problem of module selection, floorplanning and architecture generation, first introduced in [6]. This includes a description of how certain parameters used in the formulation are obtained.

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 [7]. Figure 1 shows the design flow and how our tool interacts with existing software. The benchmarks consist of a set of computational 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 floorplanning and module selection across configurations of the device. This is shown in Figure 2, in which the proposed framework has been used to generate an architecture to be used specifically for two different benchmarks, and has performed the technology mapping and floorplanning for each configuration.

The computational nodes are allowed to be constructed from the various resource types available, *i.e.* using specialised embedded components or slice-based logic; the linear programming approach allows this to be done within a unified framework. Existing synthesis tools are used to automatically determine routing delay models and component-level timing and area estimates, used in the ILP formulation. The tools allow assessment of the timing and resource requirements of each computational node under a variety of

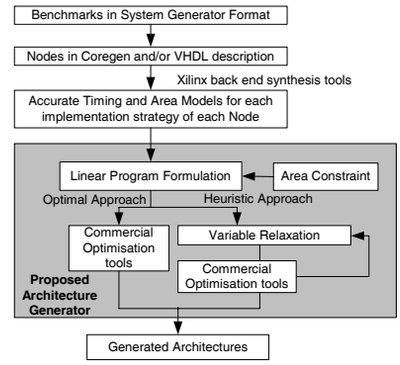


Fig. 1. The overall design flow of the optimisation problem.

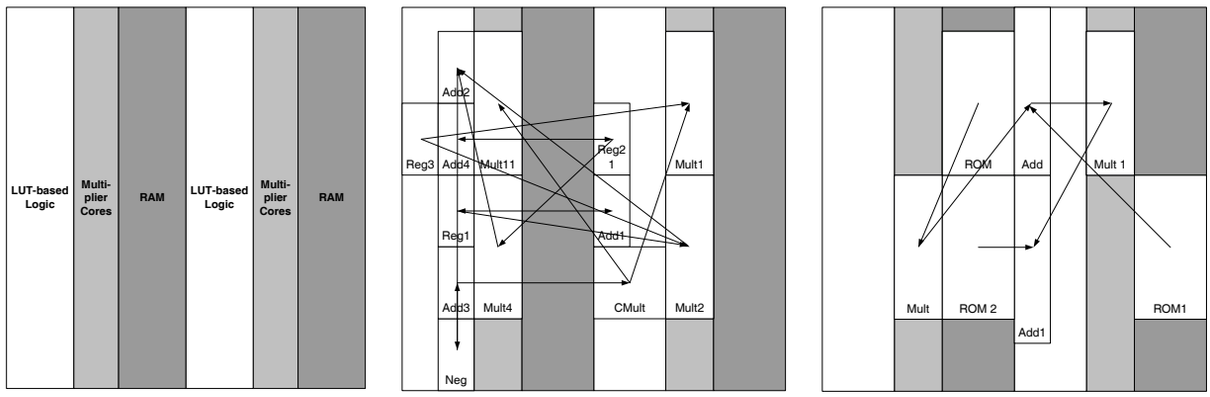
different implementation strategies. The term *implementation strategy* refers to whether the node is constructed from LUTs, embedded RAMs or embedded multipliers, a decision that is made automatically by the proposed design flow, while simultaneously considering clock period, floorplanning and area considerations.

The optimal architecture is considered to be one that optimises a measure of the different benchmark clock periods. In order to relate the clock period values between benchmark circuits, the concept of 'relative clock period' was introduced [6]. The relative clock period of a benchmark b is defined as R_b (1), where T_{sb} represents the minimum clock period of the benchmark given a particular architecture s, and T_{cb} represents the minimum clock period of the benchmark given the optimal set of available 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 available components for each benchmark circuit, rather than the best components for the entire set of benchmark circuits, are selected). R_b can be thought of as *a measure of the speed lost as a result of introducing reconfigurability*. The overall goal of the optimisation is to minimise the maximum value of R_b across all benchmarks.

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

2.1. Linear Programming Formulation

Benchmarks in the system are specified as Dataflow Graphs (DFGs). In a dataflow graph, nodes specify different types of computation that can be performed, with edges representing communication, or dataflow, between nodes. Using Bellman's equations [8], it is possible to formulate the problem of determining the critical path of the circuit, and hence minimum clock period of a circuit. Bellman's equations have been formulated in a way to incorporate module selection and cross-chip delay using the Manhattan model [6], where the constants of proportionality in the Manhattan model



(a) (b) (c)

Fig. 2. A simple scaled-down example floorplan of a throughput-optimal architecture/mapping combination generated by our work for a particular area constraint. (a) The architecture, (b) A mapping of a 1st order LMS adaptive filter, (c) A mapping for a 2nd order polynomial evaluation. Arrows indicate the edges in the data-flow graph; their Manhattan length is related to the contribution of inter-node routing delay to circuit critical path.

were obtained by modelling the delay between two circuit elements in a Virtex 2 chip.

In the device floorplan, nodes must be prevented from overlapping with each other. The related constraints can be thought of as a modified version of the 2-dimensional packing problem, where the sizes of the nodes are not known *a priori*, and the objective function relates to node interconnect and combinatorial delay. The fundamental run-time scalability problem with the ILP model from [6] is to ensure no overlap of computational nodes, illustrated graphically in Figure 3. No overlap is equivalent to at least one of the inequalities in (2) being satisfied. To cast this disjunction in linear form, it may be replaced by (3-8), where (7) ensures that at least one of the inequalities is true. In these inequalities, A_x and A_y are constraints on the device width and height, thus the overall area constraint $A = A_x A_y$.

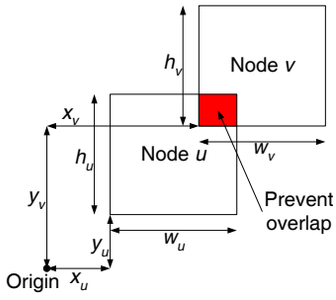


Fig. 3. An illustration of the placement constraints in the linear programming formulation

$$(x_u + w_u \leq x_v) \vee (y_u + h_u \leq y_v) \vee (x_v + w_v \leq x_u) \vee (y_v + h_v \leq y_u) \quad (2)$$

$$A_x \delta_{uv1} + x_v - x_u - w_u \geq 0 \quad (3)$$

$$A_y \delta_{uv2} + y_v - y_u - h_u \geq 0 \quad (4)$$

$$A_x \delta_{uv3} + x_u - x_v - w_v \geq 0 \quad (5)$$

$$A_y \delta_{uv4} + y_u - y_v - h_v \geq 0 \quad (6)$$

$$\delta_{uv1} + \delta_{uv2} + \delta_{uv3} + \delta_{uv4} \leq 3 \quad (7)$$

$$\delta_{uvj} \in \{0, 1\}, \quad j = 1, 2, 3, 4 \quad (8)$$

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, similar to the most recent devices from Xilinx. This is illustrated in Figure 2, in which small benchmarks have been fed into the proposed system to produce an architecture and some corresponding benchmark floorplans. To prevent the regions of different resource types overlapping, similar constraints to the node-overlap constraints must, of course, be introduced, and can be linearised in the same manner. Constraints are also included to ensure nodes can only be placed in the correct type of region. These have been formulated in a way that allows the optimal region locations and widths, as well as mapping of nodes to regions, to be determined through solution of the ILP.

3. HEURISTIC DETERMINATION OF RECONFIGURABLE ARCHITECTURES

The run time of the ILP solver makes a direct solution of the ILP published in [6] unattractive for large benchmark sets ($\gg$ 24 hours). This is because the number of binary variables modelling node-node placement grows quadratically

with the number of nodes. As a consequence, it has been necessary to develop a methodology to counter this problem. The ILP framework summarised above allows the development of a heuristic approach in a structured manner, resulting in upper bounds on the achievable, and steering the search for the optimal solution. The heuristic procedure is summarised in Figure 4.

The technique developed is based around a controlled relaxation of the binary decision variables δ_{uvx} (3-8) to reals in the range $[0, 1]$. Removing the integrality allows fast solution through, for example, the Simplex method [9]. 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 rounding heuristic is detailed in Section 3.2.

The heuristic procedure can be summarised as follows. Firstly, an ILP is run to minimise the relative clock period with no constraints on node locations. This run of the ILP is fast, as the binary variables introduced in (8) are not present; the only binary variables present are those defining the module selection, *i.e.* whether a component should be constructed from LUT-based logic or embedded components. A clustering heuristic is then applied in order to partition the device into column-based regions of each resource type, and group nodes from all benchmarks into the appropriate regions. Once the regions are assigned, the clock period is minimised with constraints on the regions (see Figure 2). Finally, the sum of the relaxed decision variables is minimised before determining which of the relaxed variables to round, thus gradually avoiding overlap between computational nodes. In each run of the linear program, the binary decision variables δ_{uvx} (3-8) remain relaxed to real values, and are only fixed to integer values in the rounding phase.

3.1. Achieving a Scalable Run-Time Using Clustering

A novel aspect with the proposed heuristic, introduced in order to guarantee scalable runtime, is the introduction of the clustering phase. As the number of regions allowed on the device is increased, so is the number of binary variables related to floorplanning of regions and placement of nodes within regions. This has a significant effect on run-time, hence a phase has been introduced to determine the locations of the regions, as well as the assignments of nodes to regions.

The clustering algorithm is based on the well known k-means algorithm [10]. In this instance, it is used to choose where regions should be placed in relation to one another, and in which region nodes should be implemented, in order to give a suitable architecture. A modification of the k-means heuristic has been made due to the additional constraints introduced in this particular problem. In a standard k-means problem, there are no constraints on the node locations; the introduction of these constraints means that the clustering of each node has to be verified.

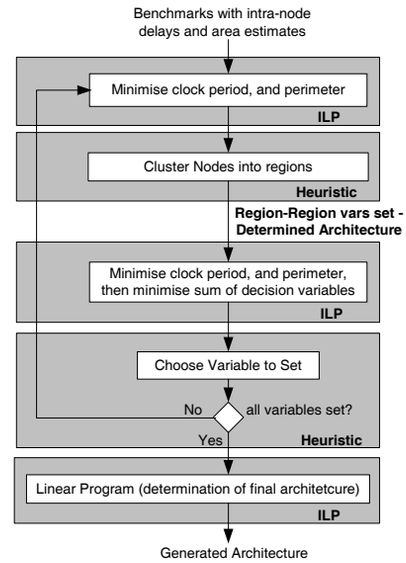


Fig. 4. The heuristic used to solve the combined optimisation problem.

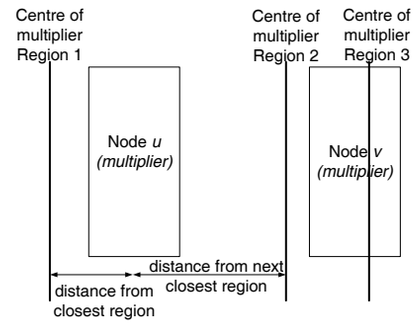


Fig. 5. An example of how the penalty of node u is calculated in the clustering procedure.

For each resource type, an arbitrary starting location for the centre of each region is defined. Nodes are assigned one-by-one to their closest feasible region, with the order in which nodes are placed determined by the penalty of the node. The penalty is a value that defines how much the floorplan is affected if a node is not assigned to its closest region, and is calculated as follows. The difference between the horizontal coordinate of the node and the location of the closest region that it can be feasibly placed within is calculated, as is the difference between the horizontal coordinate of the node and its next closest feasible region. The penalty is then calculated as the difference between these values. This is illustrated in Figure 5. Nodes that are close to one region but

far from their next closest region are placed first. If a node can only be placed in one region it is given a large penalty, so that it is assigned to a region immediately. The penalties are recalculated after each node has been assigned to a region, as placing a node in a region adds placement constraints on other nodes.

After all nodes have been clustered, each region has a new centre assigned that corresponds to the mean of the centre of all nodes that have been assigned to that cluster. The nodes are then re-clustered, with the process repeating until there is no change in the location of each of the regions' centres between iterations in a similar way to the classical k-means algorithm.

3.2. Determination of Architectures Using Variable Relaxation

The critical feature of this heuristic approach is that the decision variables related to relative placement of nodes are relaxed. This means that the only binary decision variables in each ILP run are those that determine the implementation strategy of each node. After the post-clustering stage, and after the perimeter of the device has been minimised, the variables (3-7) do not appear in the objective function and so take arbitrary values amongst the values that satisfy the constraints. The linear program is thus re-run with the relative clock period fixed to the minimum, while minimising the sum of all decision variables is used as the objective.

In deciding which variables to round, each set of variables (8) is then examined. The minimum of these four is chosen as a candidate to round down, and in order to make critical decisions first, the pair (u, v) with maximum variable of all of these minima is chosen. In order to consider global, rather than just pair-wise, efforts, each of the placement variables is also scaled according to how much free space there is. The term *free space* refers to the difference between the used space and the allowable space in each dimension, as defined by the width and height constraints. Thus the decision of which set variable to round becomes that given in (9), where X_{max} and Y_{max} refer to the used space in the x and y directions, and are determined by examining the solution of the perimeter minimisation phase. In scaling the variables this way, the dimensions of the device can be accounted for, and the area efficiency of the heuristic can be improved.

$$\max_{b \in B} \left(\min_{u,v \in V} \left(\frac{\delta_{uv1}}{(A_x - X_{max})}, \frac{\delta_{uv2}}{(A_y - Y_{max})}, \frac{\delta_{uv3}}{(A_x - X_{max})}, \frac{\delta_{uv4}}{(A_y - Y_{max})} \right) \right) \quad (9)$$

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: an LMS adaptive filter; a multi-channel IIR filter and a Costas Loop, as supplied with Xilinx System Generator; a programmable two dimensional 5x5 image convolution on a raster-scanned image, based on that supplied with System Generator; an ADPCM encoder; and a Horner scheme polynomial evaluator. Area figures (in multiples of a slice[1]) are given in Table 1.

The results taken used the following methodology. The heuristic was used to generate an architecture for all benchmarks. Each benchmark was individually re-mapped onto the generated architecture using the full ILP, where the parameters of the commercial device are specified as constraints in the ILP. This approach provides a higher quality solution than the mapping used during the heuristic architecture creation procedure. Results were then taken for a mapping of each circuit onto a Xilinx XC2V2000 device. Similarly, parameters of the commercial device are specified as constraints in the ILP. The device generated by our procedure was given the same overall area as the Xilinx device, with the areas devoted to each component type determined by the heuristic procedure, however the device is constrained to have the same number of regions of each embedded resource type as the Xilinx device.

In order to quantify the benefit that embedded multipliers provide, benchmarks were mapped to a device created by removing the multiplier columns, leaving only 18kbit embedded memory blocks and lookup logic. Similarly, a device with no embedded memory (but with multipliers) was examined. Finally, a comparison to homogeneous fine-grain fabrics was performed by removing the columns of both multipliers and embedded memory.

On a Pentium 4 - 3.4GHz running Fedora (Linux), the run-time of the heuristic for the combined problem with the benchmark set supplied is approximately 48 hours, whereas the combined ILP does not provide a solution before the computer's memory resources (2GB RAM) are exhausted. The execution time of the combined floorplanning and technology mapping heuristic is at most 30 minutes for a single benchmark, and the ILP lower-bounding procedure is run for an equivalent length of time for each mapping of a benchmark onto an architecture.

Results of the architectural evaluations are shown in Table 1. The results show an upper and lower bound on the clock periods of each benchmark circuit for each architecture described above. The upper bound is a known feasible solution, determined by the ILP solution software. Similarly, the lower bound is a known, provable lower bound on the clock period of each benchmark. The table entries showing 'no solution' mean that the ILP solver was unable to find a solution given the time constraint of the software.

One of the most striking features of the results in Table 1,

[1] A slice is a component capable of performing two 4-input lookup functions with optional output registers. Fast carry-logic is also included.

Table 1. This table shows a comparison a commercially available device family to a device generated by the heuristic procedure. Relative clock period is evaluated by floorplanning the benchmark using an ILP to determine T_{cb} in (1). Figures showing the area required for each benchmark are also given, with embedded component areas normalised to the size of a slice. 'no solution' indicates that the solution software was unable to find a feasible solution given the time constraint.

Benchmark Circuit	Area relative to slice size		Bounds on Clock Period (nanoseconds)									
	Optimal Components	Implemented in slices	Generated Device		Xilinx XC2V2000		Xilinx - No Mults		Xilinx - No RAM		Xilinx - LUT only	
			Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower
Adaptive Filter	453	3080	69.335	65.537	69.377	65.567	no solution	79.509	69.907	65.537	no solution	79.509
Polynomial Evaluator	367	2974	104.31	103.34	105.51	104.94	165.12	160.63	105.07	102.99	165.55	160.82
IIR Filter	380	1349	27.364	27.316	27.528	27.316	35.409	35.316	28.883	28.883	36.927	36.883
Image Convolution	432	5792	22.73	22.494	22.524	22.524	22.524	22.494	22.524	22.494	22.524	22.494
Costas Loop	318	1857	76.219	75.244	75.516	75.509	83.652	82.671	75.73	75.244	83.501	82.668
ADPCM encoder	200	295	86.26	85.26	85.26	85.26	85.313	85.302	85.26	85.26	85.313	85.302
Maximum Relative			1.013		1.024		1.60		1.06		1.61	

is that the maximum relative clock period of the Xilinx design is close to optimal, shown by the relative clock period of 1.024. Small gains can be achieved by re-evaluating the floorplan, shown by the results for the heuristically generated architecture. However, these can only reach a maximum of 2.4%. It can therefore be concluded that significant further improvements can only be made by using different types of embedded components, rather than different orderings or arrangements of the existing ones.

Another feature of the results is that the architecture generated by our optimised automatic generator results in a 58% speed improvement over architectures without embedded multipliers. This means that there is significant room for improvement over devices with no multipliers. Interestingly, the introduction of memory components has little effect on circuit clock period. This observation can be explained by the fact that memory is rarely on the critical path of these benchmark circuits, and our optimisation system takes this into account when performing technology mapping. However, the density advantages are clear from the area figures given in Table 1: without the embedded components, circuits can require over 13 times the area than slice-based implementations.

5. CONCLUSION

This paper has described a heuristic approach to the combined reconfigurable architecture design, floorplanning, and technology mapping problem. As a result, we have been able to present formal upper and lower bounds on the speed attainable by reconfigurable architectures based on slices, 18x18 multipliers, and 18kb embedded RAM blocks. The proposed methodology has been able to automatically design an architecture capable of supporting several input benchmark circuits, and has quantified the *optimal* speed and logic density achievable on the basis of these embedded components for the first time. These results indicate that, while a significant system-level speedup is attained by incorporating embedded multipliers for DSP benchmarks, further improvements in speed are likely to arise only as a result of the design of new embedded components.

6. REFERENCES

[1] E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2000.

[2] J. Cong and S. Xu, "Architecture evaluation for FPGAs with embedded memory blocks," Tech. Rep., University of California, Los Angeles, 1998.

[3] K. Eguro and S. Hauck, "Issues and approaches to coarse-grain reconfigurable architecture development," in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003.

[4] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.

[5] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," in *ACM International Symposium on Field Programmable Gate Arrays*, 2006.

[6] A. M. Smith, G. A. Constantinides, and P. Y. K. Cheung, "Generation and exploration of reconfigurable architectures using mathematical programming," in *Field-Programmable Logic and Applications*, 2005.

[7] J. Hwang, B. Milne, N. Shirazi, and J. D. Stroomer, "System level tools for dsp in FPGAs," in *Field-Programmable Logic and Applications*, 2001.

[8] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[9] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, John Wiley and Sons, Inc., New York, 1972.

[10] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability"*. 1967, University of California Press.