Integrated Floorplanning, Module-Selection and Architecture Generation for Reconfigurable Devices

Alastair M. Smith, *Member, IEEE,* George A. Constantinides, *Member, IEEE,*
and Peter Y. K. Cheung, *Senior Member, IEEE*

***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. A mathematical programming framework is introduced, along with a 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. The approach detailed provides a formal mechanism to explore future technology nodes.**

***Index Terms*— FPGA, Reconfigurable Architectures, Integer Linear Programming, floorplanning, module-selection.**

I. Introduction

For system architectures, reconfigurable devices provide designers with high-throughput, cost-effective platforms. When designing reconfigurable devices, the architects must be aware of the types of system that users intend to map onto the platform. This means that device architectures must be designed with the performance of the different systems they are intended for in mind. Given the wide-spread use of reconfigurable architectures for DSP systems, there have been several advances in reconfigurable chip design for this domain, particularly evident in Field Programmable Gate Arrays (FPGAs).

Traditionally, FPGAs have consisted of Look-Up Tables (LUTs) capable of performing any N-input logic function. 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. More recent introductions into the FPGA fabric include components such as DSP blocks and RAM, for instance the Xilinx Virtex 2 [3] contains lookup-based slices[1], 18kbit embedded SRAM as well as 18-bit multipliers. There are optional registers on the outputs. Similarly, Altera's Stratix II chips [4] contain three different sizes of memory as well as DSP blocks, the latter being capable of performing a variety of fused multiply-add operations. These embedded components have been used to speed up computation or take advantage of greater logic density. In this paper, the emphasis is on exploring heterogeneous architectures by examining the ratios and physical placements of the different components that are found in heterogeneous devices.

When designing a new reconfigurable device, it is common for the architects to have a base-line parameterisable structure from which many different architectures can be generated. A considerable body of work in FPGA research has been based on the Versatile Place and Route tool [5], which provides such a structure. The parameters of this type of framework can then be varied to represent different devices with different characteristics. For example the mix of routing resources can be varied [6], [7], or the number of inputs for each logic function [1], [2]. The most common approach for testing the performance of the architectures is to simulate a variety of possible architectures, 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.

The work presented in this paper takes a different approach to reconfigurable architecture design. By using linear programming (LP) and integer linear programming (ILP), 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 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. 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.

Many of these subproblems have been addressed in previous work. For instance, there is a large body of work on floorplanning from the ASIC community. However, this is the first time that these three problems have been addressed within the same framework. Moreover, this work addresses floorplanning problems for which the underlying architecture is configurable and must be reused for different applications.

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 floorplanning, module

This work has been funded by the EPSRC (UK) under grant numbers EP/C549481/1 and EP/E00024X/1, and under the EPSRC DTA scheme.

[1]A slice is a component capable of performing two 4-input lookup functions as well as additional carry logic.

selection and architecture generation.
- Floorplanning optimisation for configurable devices where the underlying architecture is reused for different applications.
- A heuristic for solution of the combined architecture generation, floorplanning, and module selection problem.
- Bounds on the distance to optimality of the architectures generated, quantifying, for the first time, the *optimal* area/speed advantages for a class of reconfigurable architectures.
- A comparison of the generated architectures to a family of commercial architectures.

The remainder of this paper is organised as follows. In Section II related work is discussed. Section III details the mathematical formulation of the problem as a linear program, based on that presented in [8], and Section IV, presents a heuristic bounding procedure for the optimisation problem, when direct solution of the ILP is not possible due to runtime and computational constraints.

II. Related Work

Many research works have been devoted to automatic floorplanning of digital circuits, we refer the reader to [9] for a comprehensive review of these techniques. The focus of the paper we present here is on ILP floorplanning for reconfigurable hardware. The problem of floorplanning for reconfigurable hardware is somewhat different than that of standard VLSI floorplanning. Reconfigurable hardware can be configured according to the application for which it is to be used. This means that the underlying device floorplan is reused, and that a good architecture floorplan for one application may not be good for an alternative application. In this paper, we build on known ILP floorplanning techniques [10], and extend them into the realm of heterogeneous reconfigurable devices.

Heterogeneous reconfigurable devices provide an extra dimension to the problem of floorplanning, as they provide the designer with choices for implementing different parts of their design. This is essentially a module selection problem, which has been studied in previous works, for example [11] in the VLSI domain and [12] from a High-Level Synthesis perspective. There are also known ILP formulations of the module selection problem, for example [13], which is again from a high-level synthesis perspective. In the context of FPGA technology, [14] combines technology mapping to homogeneous fine-grain FPGAs with placement and routing, although does not consider heterogeneous fabrics. More recent FPGA related works on technology mapping consider embedded memories, for example [15] and there is also work on usage of DSP blocks [16]. In contrast to the paper we present here, existing works have not considered the heterogeneous technology mapping problem in a context where the modules have to be reused when the device is configured for a set of different applications.

There is also a considerable body of work in the field of reconfigurable computing relating to exploration of various aspects of the architecture. However, there are relatively few concerning heterogeneous devices. [17] considers architectures containing embedded memories as well as LUTs. 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. However, more modern architectures containing components such as multipliers are not covered by their work.

In [18] 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, whereas in the work presented here uses linear programming to evaluate a large set of architectures.

Heterogeneous coarse-grain reconfigurable devices have been researched extensively in the RaPiD project [19], [20], [21]. The result of this work has been a tool that generates device architectures, and the most recent work compares RaPiD architectures to standard-cell ASICs and FPGAs. A common theme throughout the RaPiD project has been on using heuristics to solve all parts of the design flow, meaning that their work provides experimentally upper-bounds on the best solutions possible. The methods employed in the paper we present are based on formal mathematical methods, meaning that lower-bounds can be obtained in addition to upper bounds. Moreover, the devices that are generated in our work mix both fine and coarse-grain, allowing closer comparison to modern commercial FPGAs.

More recent advances have attempted to quantify the gap between heterogeneous FPGA architectures and ASIC implementations [22]. In this paper, 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. Our work complements [22] by providing an analytical approach to the problem; while [22] tries to measure the FPGA/ASIC gap for the whole tool chain, our work uses formal techniques to isolate heuristic tool bias from architectural measurements.

The work presented in this paper 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, however, a routing model is used to estimate the inter-component delays, allowing device floorplan to vary as a problem parameter. The work presented in this paper concentrates on generation of heterogeneous FPGAs,

TABLE I

NOTATION USED IN THIS PAPER.

Notation	Meaning
A	The total silicon area of a device ($A = A_x A_y$).
A_x	The width constraint on a device.
A_y	The height constraint on a device.
a_{ir}	Left-hand boundary of region number r of resource type i.
B	The benchmark set.
b_{ir}	Right-hand boundary of region number r of resource type i.
C_b	Relative clock period of benchmark b for a given target architecture.
C_{max}	The maximum relative clock period of all benchmarks.
d_{ui}	Decision variable that is 1 when node u is implemented by resource type i and zero otherwise.
d_{uir}	Decision variable that is 1 when node u is implemented in region r of resource type i and zero otherwise.
E_b	The set of dataflow edges (u, v) connecting the output of node u to one of the inputs of node v in G_b.
f_{ui}	The combinatorial delay of node u when implemented by resource type i.
h_u	Height of node u.
h_{ui}	Height of node u when implemented by resource type i.
I	The set of all resource types.
$I(u)$	The set of all resource types/implementation strategies that can implement node u.
m_{uv}	Routing delay between nodes u and v.
R_i	The set of regions of resource type i.
T_{sb}	Clock period of benchmark b given the target architecture s.
T_{ob}	Clock period of benchmark b given the time-optimal set of components for a given area constraint.
T_u	The start time during a clock period of node u.
V_b	The set of nodes in G_b.
w_u	Width of node u.
w_{ui}	Width of node u when implemented by resource type i.
x_u	x-coordinate (left hand side) of node u.
y_u	y-coordinate (bottom) of node u.
δ_{uv1}, δ_{uv2}, δ_{uv3}, δ_{uv4}	The four decision variables associated with the relative placement of two arbitrary nodes u and v.
δ_{irjs1} δ_{irjs2}	The two decision variables associated with the relative placement of regions r of type i and region s of type j.
X_{max}	Total used width of the device.
Y_{max}	Total used height of the device.
ζ_{uv}	Horizontal wirelength between nodes u and v.
γ_{uv}	Vertical wirelength between nodes u and v.

and synthesis effects are minimised by using mathematical programming. This allows the determination of performance bounds and true optima. The analytical formulation described in the next section is used to optimise the architecture floorplan for a benchmark set under a given area constraint.

III. DESIGN FLOW AND PROBLEM FORMULATION

The following section details the formulation of the combined problem as linear program. The heuristic methods developed in later Section IV of this paper are based on the ILP formulation, and in particular the constraints and variables introduced in Section III-B. For reference Table I gives the notation used throughout the paper.

The architecture generator uses linear programming in order to combine the problem of benchmark floorplanning and module selection, as well as underlying architecture floorplanning. This is as illustrated in Figure 1(a-c), in which the proposed framework has been used to generate an architecture

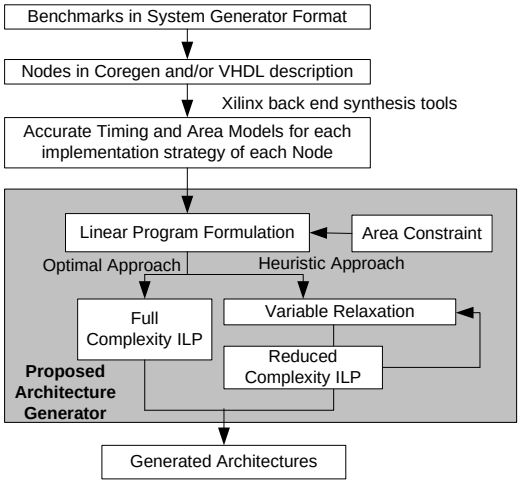

Fig. 2. The overall design flow of the optimisation problem.

to be used specifically for two different benchmarks, and has performed the technology mapping and floorplanning for each configuration.

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 [23]. Figure 2 shows the design flow and how our tool interacts with existing software. The benchmark circuits are specified as a dataflow graph (DFG). The benchmarks consist of a set of node types that represent various computations. The tool optimises the architecture for the set of benchmarks supplied such that the architecture can be reconfigured in order to implement any one of the benchmarks.

The computational structures in the benchmarks are allowed to be constructed from the various resource types available; the linear programming approach allows this to be done within a unified framework. Existing synthesis tools supplied by Xilinx are used to determine various constants, such as 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 different implementation strategies. The term *implementation strategy* refers to whether the node is constructed from LUTs, embedded RAMs or embedded multipliers.

Inter-node delays are estimated 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.

In order to relate the clock period values between benchmark circuits, the concept of 'relative clock period' has been introduced [8]. The relative clock period is defined in (1), where C_b represents the relative clock period of benchmark b, T_{sb} represents the minimum clock period of the benchmark given a particular architecture s, and T_{ob} represents the minimum clock period of the benchmark given the optimal set

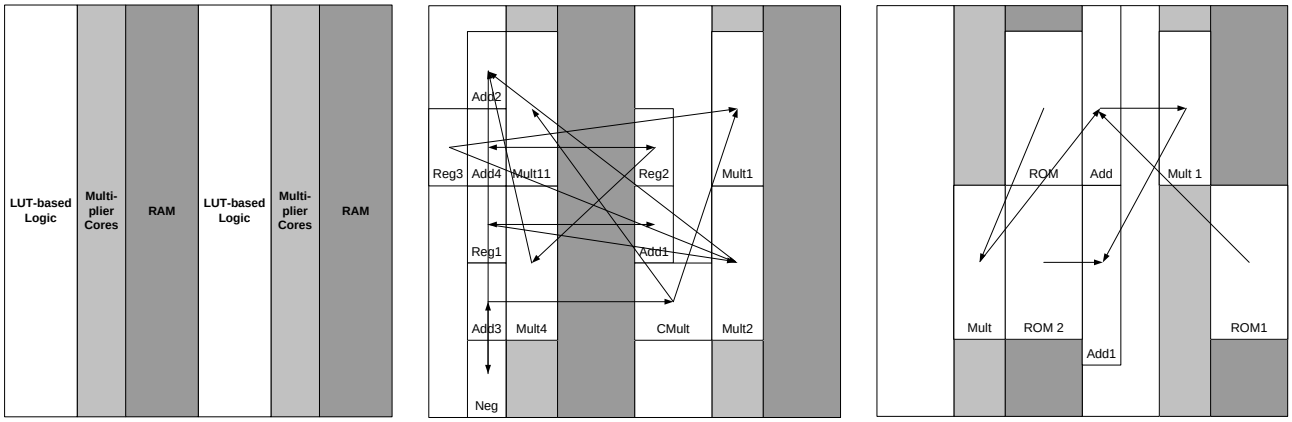

Fig. 1. A simple scaled-down example floorplan of a throughput-optimal architecture/mapping combination generated by our work for a particular area constraint. (a) The architecture, (b) A mapping of a 1st order LMS adaptive filter, (c) A mapping for a 2nd order polynomial evaluation. Arrows indicate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

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, rather than the best components for the entire set of benchmark circuits, 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, if the device were designed solely for that specific benchmark. Note that C_b is a measure of the *worst case* relative clock period of each benchmark. The overall goal of the optimisation is to minimise the maximum value of C_b over all benchmarks.

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

A. Linear Programming Formulation

The benchmark set is denoted B, and each $b \in B$ is a dataflow graph representing one benchmark circuit. 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 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. 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), which includes a linearisation of the max operator.

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

The delay of the circuit is dependent on whether a node is mapped to slices or embedded hard-IP, so the constraints dealing with circuit delay (based on Bellman's equations (3) [24]) have to be formulated to account for this, where T_u represents the time during a clock cycle at which the inputs to node u become valid and the edge weight $f(u, v)$ 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 within a clock cycle, *i.e.* $T_{source} = 0$. Bellman's Equations have been formulated in such a way as to account for the different ways of implementing certain nodes. Thus the version of Bellman's equations used in the ILP formulation are as in (4-5).

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

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

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

Here m_{uv} represents the portion of delay due to routing, and $I(u)$ represents the set of implementation strategies for node u, for example {embedded multiplier, LUT-based, embedded RAM}. The combinatorial delay of node u when implemented in strategy i is represented by f_{ui}. The binary decision variable d_{ui} has the value 1 if and only if 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).

$$\forall\ (u, v) \in E_b \qquad m_{uv} = k_1 + k_2(\zeta_{uv} + \gamma_{uv}) \tag{6}$$

$$\left.\begin{array}{rcl}\zeta_{uv} & \geq & x_v - x_u - w_u \\ \zeta_{uv} & \geq & x_u + w_u - x_v\end{array}\right\} \quad \zeta_{uv} = |x_v - x_u - w_u| \quad (7)$$

$$\left.\begin{array}{l}\gamma_{uv} \geq y_u - y_v \\ \gamma_{uv} \geq y_v - y_u\end{array}\right\} \quad \gamma_{uv} = |y_u - y_v| \quad (8)$$

$$\forall\ u \in V_b \qquad w_u = \sum_{i \in I(v)} w_i d_{ui} \quad (9)$$

$$\forall\ u \in V_b \qquad h_u = \sum_{i \in I(v)} h_i d_{ui} \quad (10)$$

In the routing delay model k_1 and k_2 are the constants of proportionality, and have been ascertained by evaluating inter-component delays on a Virtex 2 device. The Manhattan model has been shown to work well for uncongested designs. In Equation (6), 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. Note that the routing delay is taken from the bottom right hand corner of the output node to the bottom right hand node of the input node, hence the w_u term in (6). This is because the nodes are generated by Xilinx Coregen, and use this same convention for inputs and outputs.

In the LP, variables ζ_{uv} and γ_{uv} are used for the horizontal and vertical distance between nodes. Because the linear program is a minimisation problem, the absolute brackets in (6) can be linearised by introducing the appropriate weights (7-8); minimisation of the objective ensures that (7-8) hold at equality. Equations (9) and (10) 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.

B. Constraining Node Placement

In order to represent the benchmark floorplanning problem, nodes within any one benchmark must be prevented from overlapping with each other. The related constraints can be thought of as a modified version of a 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. This can be visualised as in Figure 3. In this case at least one of the terms in (11) must be true.

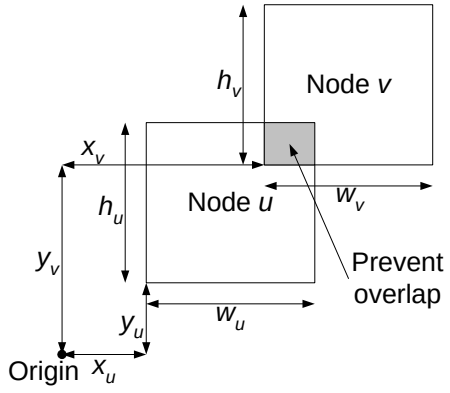

Fig. 3. An illustration of the placement constraints in the linear programming formulation

$$\begin{array}{c}(x_u + w_u) \leq x_v \vee (y_u + h_u) \leq y_v \vee \\ (x_v + w_v) \leq x_u \vee (y_v + h_v) \leq y_u\end{array} \quad (11)$$

(11) can be represented in a mathematical program by introducing four binary variables δ_{uv1}, δ_{uv2}, δ_{uv3} and δ_{uv4} as in (12-17), where (16) ensures that at least one of the terms in (11) is true. A_x is the constraint on the device width, and A_y is the constraint on the device height, the overall area constraint $A = A_x A_y$. Thus the aspect ratio of the device may be varied.

In these equations, if δ_{uv1} is equal to zero then equation (12) is satisfied with $x_v \geq x_u + w_u$, *i.e.* x_v is to the right of node u. Similarly, if δ_{uv1} is equal to one then equation (12) is trivially satisfied, as long as x_v is within the device boundary. By summing these variables (16) ensures that at least one variable is zero, ensuring that any two nodes do not overlap.

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

$$A_x \delta_{uv1} + x_v - x_u - w_u \geq 0 \quad (12)$$

$$A_y \delta_{uv2} + y_v - y_u - h_u \geq 0 \quad (13)$$

$$A_x \delta_{uv3} + x_u - x_v - w_v \geq 0 \quad (14)$$

$$A_y \delta_{uv4} + y_u - y_v - h_v \geq 0 \quad (15)$$

$$\delta_{uv1} + \delta_{uv2} + \delta_{uv3} + \delta_{uv4} \leq 3 \quad (16)$$

$$\delta_{uvj} \in \{0, 1\}, \quad j = 1, 2, 3, 4 \quad (17)$$

Although most computational nodes can be approximated by a single rectangle, some nodes need to be constructed from more than one. As an example, Figure 4 shows the floorplan of a Xilinx Coregen [25] relationally placed 8-bit multiplier. The floorplan of this multiplier is represented in the ILP by two rectangles, which are placed by setting appropriate integer variables in the ILP formulation.

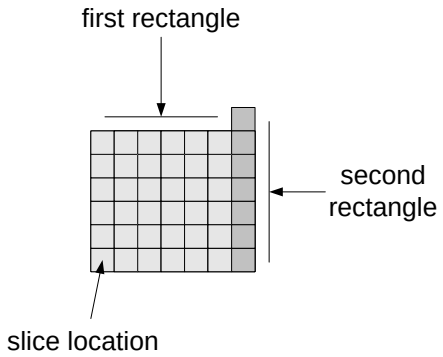

Fig. 4. An illustration of how more complex node floorplans can be constructed from rectangles

C. Representing an Architecture in the ILP

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, and is similar to some of the most recent devices from Xilinx [26] and Altera. This is illustrated in Figure 1, in which small benchmarks have been fed into the proposed system to

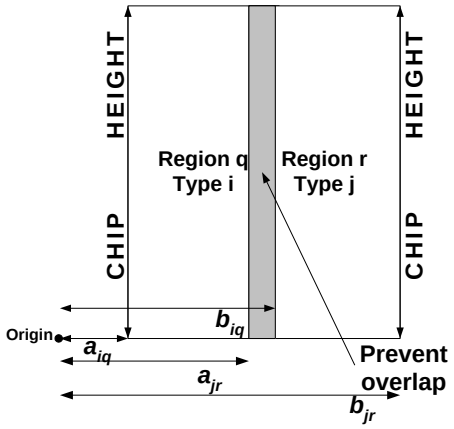

Fig. 5. An illustration of the preventing region overlap for the linear programming formulation.

produce an architecture, and some corresponding benchmark floorplans. To accurately represent this type of architecture within the ILP, there are constraints to represent the underlying architecture, and constraints that map the computational structures of the individual benchmarks onto the architecture floorplan.

In order for the system to automatically generate the architecture floorplan, the architecture template must be specified. The underlying architecture template is column based, hence it is necessary to introduce constraints to prevent overlap between the columns of the different types of resource. This overlap can be visualised by observing Figure 5, and is a one dimensional version of Figure 3. To prevent the regions of different resource types overlapping, clearly $a_{iq} > b_{jr}$ or $a_{jr} > b_{iq}$. These constraints can be represented in a mathematical program by introducing two binary variables δ_{qr1} and δ_{qr2} as in (18-21). 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 resource type i. The left and right boundaries of region r of resource type j are specified by a_{jr} and b_{jr} respectively.

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

$$A_x\delta_{qr1} + a_{iq} - b_{jr} \geq 0 \quad (18)$$

$$A_x\delta_{qr2} + a_{jr} - b_{iq} \geq 0 \quad (19)$$

$$\delta_{qr1} + \delta_{qr2} \leq 1 \quad (20)$$

$$\delta_{qrk} \in \{0,1\}, \quad k = 1,2 \quad (21)$$

To map the computational nodes of the individual benchmarks onto the architecture, and set the widths of the regions of particular types of logic, constraints (24-23) must also be included in the ILP formulation.

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

$$A_x(1 - d_{uir}) + x_v - a_{ir} \geq 0 \quad (22)$$

$$A_x(1 - d_{uir}) - x_v - w_{ui} + b_{ir} \geq 0 \quad (23)$$

$$\forall\ i \in I,\ \forall\ u \in V_b \qquad \sum_{r \in R_i} d_{uir} = d_{ui} \quad (24)$$

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 if and only if node u is implemented in region r of implementation strategy i, w_{ui} is the width of node u when implemented in strategy i.

The combination of the above constraints allows the determination of the optimal architecture. The constraints are added in for each benchmark, with the constraints on the regions of the resource types allowing the architecture to be shared between benchmarks. Valuable lower-bounds on the optimum speed for a given area constraint can be achieved once the problem has been cast in linear form, by using linear-program solvers such as ILOG CPLEX [27].

IV. Heuristic Determination of Reconfigurable Architectures

ILP is a known NP-complete problem [28]. In the context of the ILP outlined in the previous section, the large number of binary variables makes a direct solution of the ILP impossible for even relatively small benchmarks sets. In order to counter this issue, a heuristic has been developed based on the ILP framework.

The ILP framework allows the development of a heuristic approach in a structured manner. To make this approach scalable, it is first important to observe the growth of the various integer variables in the system. The number of binary modelling node-node placement δ_{uvx} (12-17) is $2n_{nodes}(n_{nodes} - 1)$, where n_{nodes} is the number of nodes in a given benchmark circuit. Similarly, the number of binary variables representing the floorplan of the underlying architecture δ_{qrx} (18-21) is $n_{regions}(n_{regions} - 1)$, where $n_{regions}$ is the number of regions. Finally, the number of binary variables representing the floorplanning of nodes onto the architecture δ_{uir} (22-24) is at most $2n_{nodes}n_{regions}$. A consequence of this is that, while valuable lower-bounds can be achieved, the computation required for exact solution explodes even for benchmarks and architecture templates of modest size.

The heuristic used aims to minimise the effect of the binary variables on run-time by removing the exponential dependence of the ILP on these variables. The approach is an iterative procedure containing two crucial elements. The first element of the heuristic deals specifically with benchmark node-floorplanning, the second element deals with architecture floorplanning and floorplanning of nodes onto the architecture. The entire heuristic procedure is shown in Figure 6.

The heuristic technique developed is based around a controlled relaxation of the binary decision variables δ_{uvx} (12-17) to reals in the range $[0, 1]$. Removing the integrality allows fast solution through, for example, the Simplex method [29], and is a commonly employed method in integer programming solvers. The resulting optimum decision variable values may then be interpreted to steer an iterative process in which after each iteration some binary variables relating to benchmark

floorplanning are fixed to zero or one, allowing a gradual crystallisation of the benchmark floorplans. The rounding heuristic is detailed in Section IV-B.

In order to determine the architecture floorplan, and mapping of nodes onto the architecture, each iteration of the procedure is broken into a number of steps. Firstly, an ILP is run to minimise the relative clock period with no constraints on node locations other than the relaxed placement variables. This run of the ILP is fast, as the binary variables introduced in (17) 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, as described in Section IV-A 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 1). The sum of the relaxed decision variables is then minimised to reduce the binary decision variables δ_{uvx} (12-17) to their minimum value, as the smallest of these variables implies the least overlap (see Section IV-B for more detail). Finally, the relaxed variable to round is determined for this iteration, thus gradually avoiding overlap between computational nodes. In each run of the linear program, the binary decision variables δ_{uvx} (12-17) remain relaxed to real values, and are only fixed to integer values in the rounding phase.

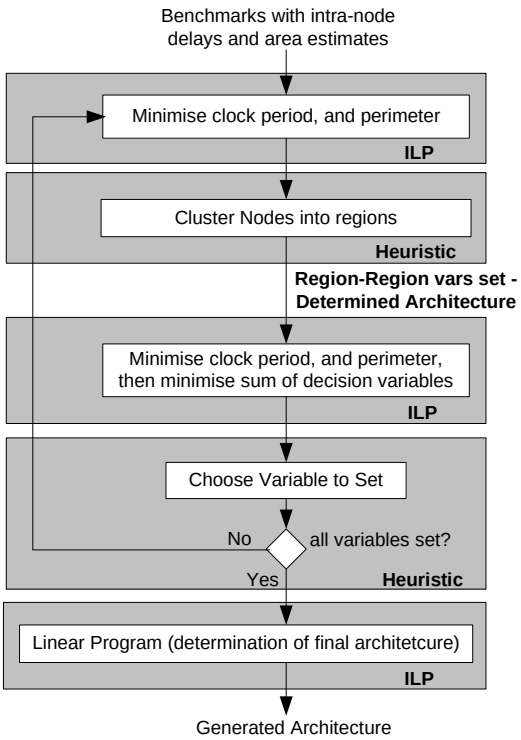

Fig. 6. The heuristic used to solve the combined optimisation problem.

A. Achieving a Scalable Run-Time Using Clustering

One particular aspect of the proposed heuristic, introduced in order to guarantee scalable run-time, 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, δ_{qrx} (18-21), and binary variables related to the placement of nodes within regions, δ_{uir} (24-23). If these variables are left to the ILP solving software to determine, a single run of the partially relaxed ILP can take over one hour, and given that the entire procedure takes over 500 iterations for the benchmark set used in our experiments, such an approach is not appropriate. Hence a phase has been introduced in the optimisation procedure, used 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 [30]. In this instance, it is used to choose where regions of different types should be placed in relation to one another, and in which of the regions nodes should be implemented, in order to give a suitable architecture for the floorplan constraints in the given iteration of the overall procedure.

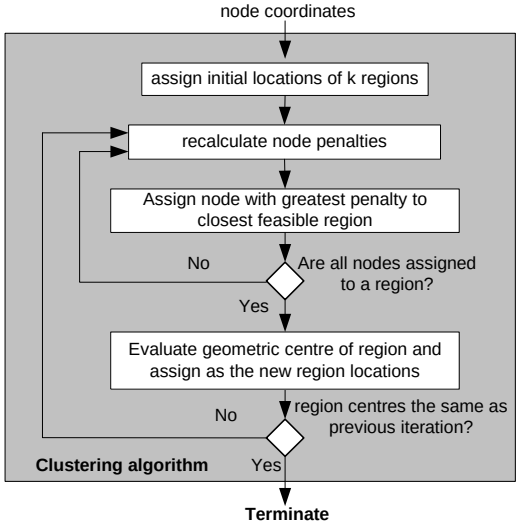

Fig. 7. The clustering algorithm.

The algorithm developed begins similarly to the k-means 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 8. 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 floorplanning a node into a particular region adds placement constraints on other nodes.

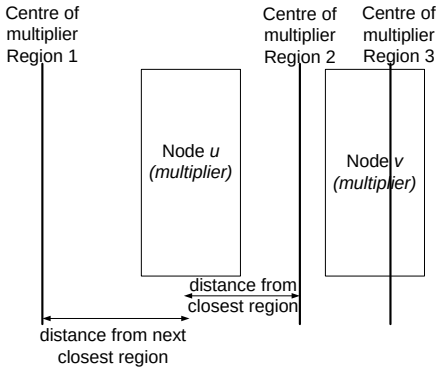

Fig. 8. An example of how the penalty of node u is calculated in the clustering procedure.

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.

The modification of the well known k-means heuristic is important due to the additional constraints introduced in this particular problem. The importance arises due to constraints on the relative placement of the node locations. After each iteration of the overall procedure in Figure 6 a variable representing the relative placement between two nodes is fixed, hence enforcing constraints on the floorplan of those nodes. The introduction of these constraints is illustrated graphically in Figure 9. The figure shows how constraints on node locations can be thought of as edges in a graph, with nodes in the graph representing the computational nodes and edges representing horizontal constraints; an edge produced by node u and consumed by node v means node v must be to the right of node u. 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. This is done by also including the constraint edges implied by a particular clustering. A feasible clustering is one for which there are no cycles in the graph. Hence to determine which region a node can be feasibly placed within, a constraint graph must be constructed for each region placement of each node, and a cyclicity check must be instantiated.

B. Determination of Architectures Using Variable Relaxation

In order to set the placements of benchmark nodes relative to one-another, a heuristic for setting the associated decision variables was developed. In terms of the overall heuristic, this part of the algorithm is denoted by the box entitled 'Choose variable to set' in Figure 6. This heuristic has been developed

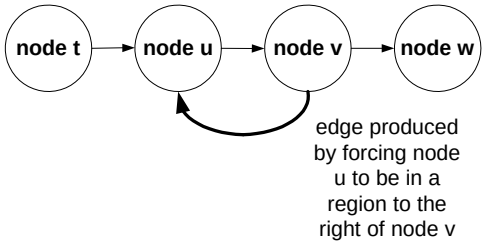

Fig. 9. Graphical representation of an infeasible layout. An edge (u, v) is present if node v is constrained to be to the right of node u through floorplanning of the benchmark *or* through floorplanning of the reconfigurable regions in which the nodes are to be implemented. This can lead to conflicts (cycles in the graph), which are detected by the proposed algorithm

to ensure an architecture floorplan falls out of the overall heuristic procedure.

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 (12-16) do not appear in the objective function and so take arbitrary values amongst the values that satisfy the constraints. For example, if (13-16) are satisfied with $\delta_{uv1} = 0.2$, $\delta_{uv2} = 0.2$, $\delta_{uv3} = 0.2$ and $\delta_{uv4} = 0.5$, they will also be satisfied with $\delta_{uv1} = 0.5$, $\delta_{uv2} = 0.5$, $\delta_{uv3} = 0.5$ and $\delta_{uv4} = 0.5$. Thus, in order to make placement decisions on the basis of these values, it is necessary to perform a 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, while minimising the sum of all decision variables is used as the objective.

After minimising the sum of the decision variables, each of the placement variables is scaled according to how much free space there is given the area constraint. 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. The placement variables related to the x-direction are divided by the free space in the x-direction, and similarly the y variables are divided by the free space in the y-direction. After scaling the variables, a decision is made as to which of the relaxed variables to examine.

In deciding which variables to round, each scaled set of variables (12-16) is then examined. In order to make critical decisions first, the minimum of these four is chosen, and the set with maximum variable of all of these minima is chosen to be examined. Thus the decision of which set of four variables to be examined becomes as in (25). 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.

$$\max_{b\in B}\ \max_{u,v\in V}\ \left(\min\left(\frac{\delta_{uv1}}{(A_x - X_{max})}, \frac{\delta_{uv2}}{(A_y - Y_{max})}, \frac{\delta_{uv3}}{(A_x - X_{max})}, \frac{\delta_{uv4}}{(A_y - Y_{max})}\right)\right) \quad (25)$$

The particular variable chosen to be rounded to zero out of the four is determined based on a method that attempts to minimise area, while maintaining clock period. The method involves examining the amount of space the device consumes in both x and y dimensions and setting variables as below.

The decision of whether to place the nodes side-by-side, or above/below is based on how much space is used in each direction, and is calculated by summing the widths of all nodes that have overlapping vertical coordinates, and summing the heights of all nodes that have overlapping horizontal coordinates (nodes may, at this stage, overlap due to the relaxation of the binary decision variables). This is illustrated in Figure 10. In the example nodes u and v have identical x and y coordinates. The used space in the horizontal direction will be the sum of the widths of nodes u, v, p, q and r, and the used space in the vertical direction will be the sum of the heights of nodes u, v and s. Node t has no overlapping coordinates, and is displayed for illustrative purposes only.

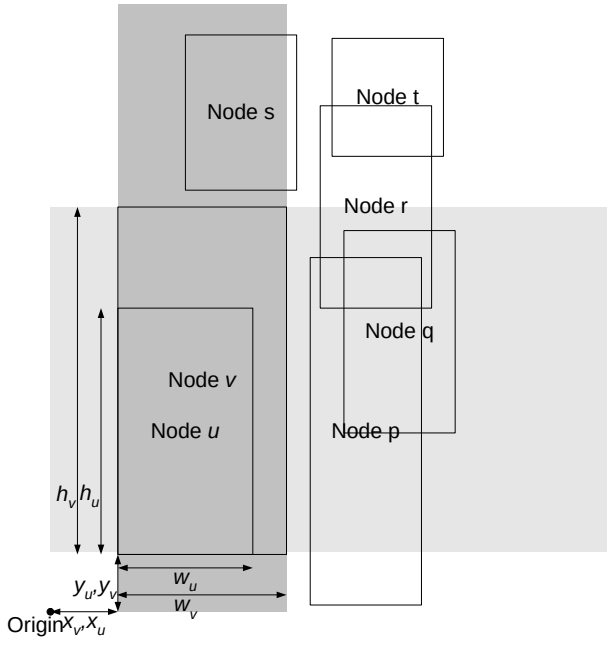

Fig. 10. Calculating the used space in the two dimensions.

The final decision of which variable to round is made according to which direction has less used space. Thus if more space is used in the x direction, then the minimum 'y' variable (δ_{uv2}, δ_{uv4}) is chosen. Similarly, if more space is used in the y direction, then the minimum 'x' variable (δ_{uv1}, δ_{uv3}) is chosen. The variable chosen is rounded down to zero, as this forces the appropriate constraint to be true.

Once the appropriate values are fixed to zero it is also possible to set any related variables to one. This is due to the constraints specified by (21) and (17). If one of the variables in these equations is zero then the others may be one. If δ_{uv1} variable in (13) is zero, this indicates that node v is to the right of u. Thus δ_{uv3} in (13) must be identically one (*i.e.* v must not exceed the right hand perimeter of the silicon area). Similarly as it doesn't matter whether u is above or below v variables (13) δ_{uv2} and δ_{uv4} can both be set to one (nodes u and v must not exceed the top or bottom perimeters of the device).

In each iteration of the overall heuristic only one set of benchmark floorplanning variables is set. This allows the device architecture to be crystallised out gradually, and allows the architecture floorplan to be dependent on the floorplan of all benchmarks. Due to implied constraints, such as those imposed by the transitivity of the relative node placements, it is not necessary to fix all sets of benchmark floorplanning variables. Some experimental observations in respect of this are given in Section V-B.

The proposed heuristic, while complex, sufficiently utilises the information resulting from partial linear program relaxations. As a result, high-quality upper-bounds are achieved, as demonstrated by their closeness to the known lower-bounds on speedup achieved through the ILP approach described in Section III. This will be demonstrated in the following section.

V. Results

The ILP and heuristic frameworks were used to explore various aspects of the problem space. In order to evaluate the performance of the heuristic approach we compare to the optimal floorplanning approach. The heuristic framework is then used to perform an architectural study of reconfigurable hardware.

A. Scalability of Optimal and Heuristic Approaches

In order to draw a comparison between optimal and heuristic approaches within feasible run-time constraints, in this section we use the ILP and the heuristic to perform floorplanning and technology mapping for a single benchmark. In these experiments the target architecture is left unconstrained, hence the comparison is essentially that of single benchmark ILP technology mapping and floorplanning versus a single benchmark version of the first phase of Figure 6.

In order to study the scalability of both approaches benchmarks of increasing complexity were chosen. A set of Horner scheme polynomial evaluators was used for this purpose. The number of nodes and edges of these benchmarks varies linearly with polynomial order, with the smallest having 7 nodes and 12 edges, and the largest 61 nodes and 102 edges and the number of integer variables varies quadratically (approximately 400 and 30,000 integer variables respectively for smallest and largest problems).

The results of the study of the scalability of this problem are shown in Figures 11 and 12. The platform used was an Intel Pentium 4 CPU running Fedora (Linux) at 2.8 GHz with GB of memory. The commercial optimisation suite CPLEX [27] was used. The run-time of the optimal approach has been averaged by using different settings in CPLEX: the runtime of these methods was seen to vary widely dependent on the problem

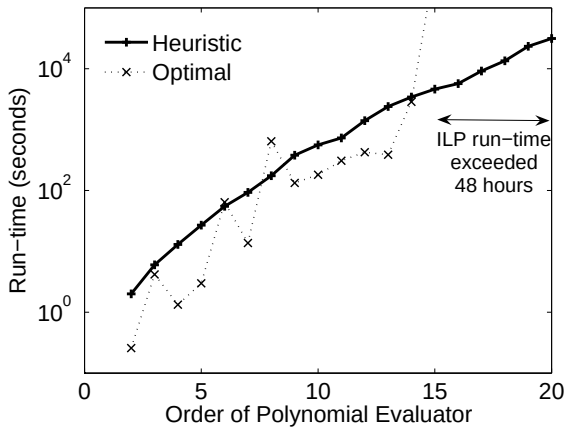

Fig. 11. A comparison of the runtime performance of optimal and heuristic floorplanning methods.

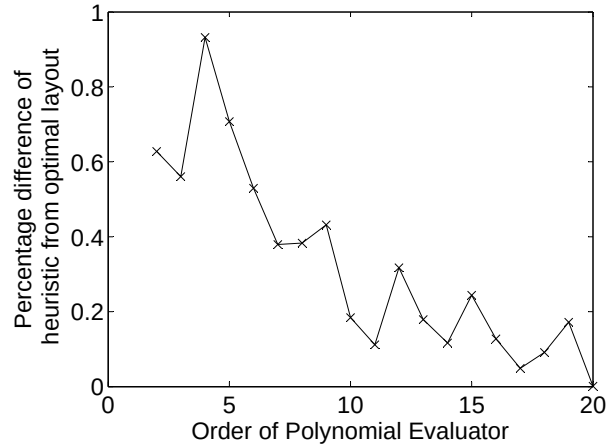

Fig. 12. A comparison of solution quality (clock period) of optimal and heuristic floorplanning methods.

size, and no particular setting could be deemed better than others. For the five largest circuits, ILP floorplanning took days to solve, whereas the heuristic method always took less than one hour.

The quality of the heuristic compared to the optimal approach is shown in Figure 12. This is a comparison of the clock periods, which can be evaluated from observing the value C_b in solution of the ILP. From the figure it is evident that, in purely floorplanning terms, the heuristic performs worse than the optimal approach by a relatively small degree (with the heuristic being no worse than 1% away from the optimal clock period).

B. Architectural Exploration

The focus of this work is DSP applications, thus to perform our exploration of the design space, a suitable set of DSP benchmarks was chosen for evaluating our heuristic approach. The benchmarks include: an LMS adaptive filter, as used in [31]; a multi-channel IIR filter and a Costas Loop, as supplied with Xilinx System Generator [23]; a programmable two dimensional 5x5 image convolution on a raster-scanned image, based on that supplied with System Generator; an ADPCM encoder [32]; and a Horner scheme polynomial evaluator. All benchmarks are of a similar hardware complexity when implemented in their time-optimal strategy, area figures relative to the size of a slice are given in Table II.

The first results presented evaluate the efficiency of the floorplanning and technology mapping heuristic for the different types of benchmark, in a similar fashion to Section V-A. Table II shows the clock periods of each benchmark when there are no constraints on the architecture, *i.e.* T_{ob} in (1). The results presented for the optimal case show the best solution obtained by the ILP solver, which is a known feasible solution, and therefore an upper bound on the optimal solution. The ILP solver also provides a lower-bound on the solution in cases where the optimum cannot be found within the defined time frame, this bound is also given in Table II. The results show that the heuristic presented deviates from the best known solution by 5.8% at worst, which is in contrast to the figure of 1% in Section V-A due to the different characteristics of the benchmarks.

The second set of results taken used the following methodology. The heuristic was used to generate an architecture for all benchmarks. Each benchmark was individually remapped 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.

On average the heuristic procedure required around 48 hours to complete. This took 700 iterations of the procedure in Figure 6, meaning that on average less than 10% of the variables related to floorplanning needed to be set. This is a consequence of implied constraints, such as the transitivity of the relative placement of nodes.

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.

The results of the architecture evaluations are shown in Table III. The table entries showing ‘no solution’ mean that the ILP solver was unable to find a solution given the time constraint of the software.

The most striking feature of the results in Table III, is that the architecture generated by our optimised automatic generator results in a 58% speed improvement over archi-

TABLE II

CLOCK PERIOD OBTAINED BY RUNNING THE ILP AS A FLOORPLANNING UTILITY IN WHICH THERE ARE NO CONSTRAINTS ON THE DEVICE ARCHITECTURE, *i.e.* THE BEST ARCHITECTURE FOR EACH BENCHMARK GIVEN THE SET OF RECONFIGURABLE COMPONENTS.

Benchmark Circuit	Clock Period (nanoseconds)			Benchmark Silicon Area (relative to slice size)		Number of Nodes	Number of Edges
	Optimal Approach		Heuristic Placement	When Implemented in Slices	When Implemented in Optimal Strategy		
	Upper Bound	Lower Bound					
Adaptive Filter	68.732	65.537	71.355	3080	453	49	100
ADPCM encoder	85.26	85.26	90.264	295	200	35	86
Costas Loop	75.423	75.244	76.123	1857	318	26	55
IIR Filter	27.316	27.316	27.346	1349	380	43	90
Image Convolution	22.524	22.494	22.764	5792	432	39	70
P'nomial Evaluator	102.99	102.99	103.565	2974	367	22	39

TABLE III

THIS TABLE SHOWS A COMPARISON A COMMERCIALLY AVAILABLE DEVICE FAMILY TO A DEVICE GENERATED BY THE ABOVE OPTIMISATION PROCEDURE. RELATIVE CLOCK PERIOD IS EVALUATED BY COMPARING TO THE BEST KNOWN MAPPING OF THE BENCHMARK FROM TABLE II.

Benchmark Circuit	Clock Period (nanoseconds)									
	Generated Device		Xilinx XC2V2000		Xilinx - No Mults		Xilinx - No RAM		Xilinx - LUT only	
	Upper Bound	Lower Bound	Upper Bound	Lower Bound	Upper Bound	Lower Bound	Upper Bound	Lower Bound	Upper Bound	Lower Bound
Adaptive Filter	69.335	65.537	69.377	65.567	no solution	79.509	69.9065	65.5372	no solution	79.509
ADPCM encoder	86.26	85.26	85.26	85.26	85.313	85.302	85.26	85.26	85.313	85.302
Costas Loop	76.219	75.244	75.516	75.509	83.652	82.671	75.73	75.244	83.5007	82.668
IIR Filter	27.364	27.316	27.528	27.316	35.409	35.316	28.883	28.883	36.927	36.883
Image Convolution	22.73	22.494	22.524	22.524	22.524	22.494	22.524	22.494	22.524	22.494
Polynomial Evaluator	104.312	103.328	105.507	104.935	165.118	160.631	105.073	102.987	165.553	160.815
Maximum Relative	1.013		1.024		1.60		1.06		1.61	

tectures without embedded multipliers, but only a 1% speed improvement over the Xilinx design. It can be concluded that the Xilinx design is close to optimum for this benchmark set; further improvements can only be made by using different types of embedded components, rather than different orderings or arrangements of the existing ones.

The largest gains seen are in the polynomial evaluator benchmark, which has a large number of multipliers on the critical path. This means that there is significant room for improvement over devices with no multipliers. When comparing the Xilinx device to the one created by the presented heuristics, the polynomial evaluator is also the benchmark with the largest improvements. The lower bound of the clock period for the Xilinx architecture is greater than the best solution of the heuristically determined architecture, meaning that although the true optimum has not been determined, it is certain that the generated architecture marginally outperforms the Xilinx architecture. In this case, the gains are due to the heuristic procedure minimising routing delay by widening certain regions. It is particularly inefficient to map this benchmark onto the Xilinx device, as much of the chip must be routed across due to each region of multipliers and memory being only one component wide; our technique automatically re-designs the device to have wider columns of these components.

In terms of the actual device construction, the generated hardware was slightly different to Xilinx's. For similar sized devices, the Xilinx Virtex 2 devotes approximately 62%, 13% and 25% of its functional area to slices, multipliers and embedded memory respectively, whereas the devices generated by our tool consume 44%, 31% and 29%. When observing the mapping of the nodes in each benchmark, it was noticed that almost all nodes capable of being implemented by a hardwired component were mapped to the embedded version. The area devoted to each component reflects this, and is hence closer to the proportions of the time-optimal component selection exhibited by the benchmarks used.

Interestingly, the introduction of memory components has little effect on circuit clock period, despite embedded memories having significantly lower latency than those built from slices. This observation can be explained by the fact that memory is rarely on the critical path of these benchmark circuits, and our optimisation system notices this, and takes it into account when performing technology mapping. However, the density advantages of embedded components can be seen by observing the area figures given in Table II. The biggest area savings are in the image convolution benchmark, which requires nine embedded memories four of which are RAM, and require over 1000 slices each, consuming over 40 times the area of a single 18kbit embedded memory.

VI. CONCLUSION

This paper has described a novel 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. The proposed framework thus allows new embedded components to be tested in order to impact on future technology generations and in light of this, we intend to address the issue of alternative embedded cores in future work. The focus of our results have been on DSP applications, and it is also the intention of future work to extend our tool to target different application domains.

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, pp. 3–12.

[2] A. Yan, R. Cheng, and S.J.E. Wilton, "On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2002, pp. 147–156.

[3] Xilinx Inc., San Jose, "Virtex 2 platform FPGA handbook," 1999.

[4] Altera Corporation, San Jose, *Stratix II Device Family Data Sheet*, 2005.

[5] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds. 1997, pp. 213–222, Springer, Berlin.

[6] S. Brown, M. Khella, and Z. Vranesic, "Minimizing FPGA interconnect delays," *IEEE Design and Test of Computers*, 1996.

[7] D. Lewis, V. Betz, D Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, "The Stratix^{TM}routing and logic architecture," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2003, pp. 12–20.

[8] A. M. Smith, G. A. Constantinides, and P. Y. K. Cheung, "Generation and exploration of reconfigurable architectures using mathematical programming," in *IEEE Field-Programmable Logic and Applications*, T. Rissa, S. J. E. Wilton, and P. H. W. Leong, Eds., 2005, pp. 341–346.

[9] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys*, vol. 23, no. 2, pp. 143–220, 1991.

[10] S. Sutanthavibul and E. Shragowitz, "An analytical approach to floorplan design and optimisation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 6, pp. 761–769, 1991.

[11] M. Pedram, N. Bhat, and E. Kuh, "Combining technology mapping with layout," *VLSI Design*.

[12] K. Ito and D. Suzuki, "A high-level synthesis method for simultaneous placement and scheduling considering data communication delay," in *Asia-Pacific Conference on Circuits and Systems*, 2002, pp. 149–154.

[13] K. Ito, L. E. Lucke, and K. K. Parhi, "Ilp-based cost-optimal dsp synthesis with module selection and data format conversion," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, pp. 582–594, 1998.

[14] N. Togawa, M. Sato, and T. Ohtsuki, "A simultaneous technology mapping, placement, and global routing algorithm for field-programmable gate arrays," in *IEEE/ACM international conference on Computer-aided design*, 1994, pp. 156 – 163.

[15] W. Gong, G. Wang, and R. Kastner, "Storage assignment during high-level synthesis for configurable architectures," in *ICCAD*. 2005, pp. 3–6, IEEE Computer Society.

[16] G. W. Morris, G. A. Constantinides, and P. Y. K. Cheung, "Using dsp blocks for rom replacement: A novel synthesis flow," in *IEEE Field-Programmable Logic and Applications*, T. Rissa, S. J. E. Wilton, and P. H. W. Leong, Eds., 2005, pp. 77–82.

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

[18] L. Bossuet, G. Gogniat, and J.-L. Philippe, "Communication costs driven design space exploration for reconfigurable architectures," in *Field-Programmable Logic and Applications*, P. Y. K. Cheung, G. A. Constantinides, and J. de Sousa, Eds. 2003, pp. 921–933, Springer, Berlin.

[19] K. Eguro and S. Hauck, "Issues and approaches to coarse-grain reconfigurable architecture development," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003, pp. 111–120.

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

[21] K. Compton and S. Hauck, "Automatic design of area-efficient configurable asic cores," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 662–672, 2007.

[22] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[23] J. Hwang, B. Milne, N. Shirazi, and J. D. Stroomer, "System level tools for dsp in FPGAs," in *Field-Programmable Logic and Applications*, G. J. Brebner and R. Woods, Eds. 2001, pp. 534–543, Springer, Berlin.

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

[25] Xilinx Inc., San Jose, *Xilinx CORE Generator*.

[26] Xilinx Inc., San Jose, "Virtex-2 Pro and Virtex-2 Pro X Platform FPGAs: Complete Data Sheet," 2004.

[27] ILOG, Gentilly France, *CPLEX Optimisation Suite*.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, New York, 1979.

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

[30] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. 1967, pp. 281–298, University of California Press.

[31] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003, pp. 81–90.

[32] http://resource.intel.com/telecom/support/appnotes/adpcm.pdf, "Dialogic adpcm algorithm," .