FPGA FAMILY COMPOSITION AND EFFECTS OF SPECIALIZED BLOCKS

Pongstorn Maidee† *Nagib Hakim*‡ *Kia Bazargan*†

† Department of Electrical and Computer Engineering
Univsity of Minnesota, Minneapolis, MN, USA
maid0013@umn.edu , kia@umn.edu

‡Intel Corp.
Santa Clara, CA, USA
nagib.hakim@intel.com

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) have gained wide acceptance among low- to medium-volume applications. However, there are gaps between FPGA and custom implementations in terms of area, performance and power consumption. In recent years, specialized blocks – memories and multipliers in particular – have been shown to help reduce this gap. However, their usefulness has not been studied formally on a broad spectrum of designs. As FPGAs are prefabricated, an FPGA family must contain members of various sizes and combinations of specialized blocks to satisfy diverse design resource requirements. We formulate the family selection process as an "FPGA family composition" problem and propose an efficient algorithm to solve it. The technique was applied to an architecture similar to Xilinx Virtex FPGAs. The results show that smart composition technique can reduce the expected silicon area up to 55%. The benefit of providing multiplier blocks in FPGAs is also shown to reduce total area by 20% using the proposed algorithm.

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) have recently been widely used in digital systems, especially for low- to medium-sized applications. However, it is well known and verified that the area efficiency of FPGAs is about 35 times worse than that of the standard-cell implementation [1]. Furthermore, their power consumption is about 14 times higher. Recognizing that most applications implemented on FPGAs use multipliers and memory units, contemporary FPGAs provide these specialized functional blocks. Since density gains of specialized blocks could be up to two orders of magnitude compared to the FPGA implementation, the area efficiency gap could be significantly reduced if such blocks are well utilized. Furthermore, power consumption reduces as a lot of programmable components are removed and circuits can be optimized. There are several works dedicated to specialized blocks. Architectural aspects have been studied in [2, 3, 4], to name a few. Mapping tools supporting various specialized blocks have also been reported [5, 6]. Although these papers showed promising results for specialized blocks, they did not consider the interaction between different specialized blocks. As a result, effects of specialized blocks considering a large set of applications cannot be inferred.

Although widely ignored in the literature, a selected set of FPGA sizes in a family affects the overall cost resulting from extra unused area. Thus, determining an appropriate set of FPGAs that contain the right mix of resources is an important problem. We call this problem "FPGA family composition" in this paper. It becomes even more important and more difficult for contemporary FPGAs containing specialized blocks, especially with the fact that FPGAs are entering more domains, hence requiring FPGA architects to study more dedicated hardware types to be embedded in FPGAs. FPGA family composition under the influence of specialized blocks is also considered in this paper. To the best of our knowledge, this is the first time that the FPGA family composition problem is formally studied.

Previous work on choosing an appropriate FPGA size has mostly focused on first choosing a specific domain, and then selecting the number of resources such as cross-bars, floating point units, etc., on the FPGA fabric. Simulated annealing (SA) and interger linear programming (ILP) techniques were used for architecture exploration [7, 8]. These approaches have two limitations : (1) SA and ILP would take a very long time and depending on the mixture of resources, SA might not even converge to a good solution. (2) the set of "representative" applications to be implemented by the FPGA should be determined in advance. Thus, although the resulting architecture is well tuned for these representatives, it would not necessarily provide good estimates on how the architecture would perform if a new application from the same application domain is mapped to the architecture [7]. Note that even though timing and congestion can be estimated using floorplaning during architecture exploration [8], the estimations are not accurate due to the lack of detailed routing information. Furthermore, power consumption is not considered because several important factors are not known [8]. This paper introduces a high-level architecture exploration which considers all designs in the domain. The main objective is to minimize the total area by appropriately composing an FPGA family. Even though timing and power consumption cannot be directly captured in the process, they are generally reduced with the FPGA area as a result of shorter connections. The technique can be used in conjunction with the existing approaches to ensure that the resulting architecture suitable for the selected representatives is also reasonable for other designs.

The rest of the paper is organized as follows. First, mapping from designs to FPGA functional blocks is discussed in Section 2. Interaction among these blocks is also formalized in this section. The FPGA family composition problem is formally stated in Section 3. The problem is solved in two steps: finding the candidates to be included in the family and choosing a good set from the candidates. The first step is elaborated in Section 4, while the second step is discussed in Section 5. Experimental results are reported in Section 6. Finally, the paper is concluded in Section 7.

2. BASIC APPLICATION MODULES AND FPGA BUILDING BLOCKS

Basic application modules are defined as a set of well-defined modules from which a designer can compose a particular design. For example, a set of modules in a digital filter may include multipliers, adders and FFs. In this work, the basic application modules are assumed to be LUTs, multipliers and RAM blocks, although our technique can be used to study any set of specialized blocks. Considering all applications, the number of modules used for each type can be plotted as a histogram. We analyzed many designs collected from [1, 3, 5] and the histogram of each module was plotted and fitted with a normal distribution.[1] In the rest of the paper, we use LUT, MEM and MUL to represent random variables (RVs) of modules LUT, block RAM and multiplier, respectively. Each of the RVs has a normal distribution.

FPGA building blocks are a set of distinct functional blocks provided in FPGAs. Different FPGA families may have different sets of building blocks. FPGA building blocks may differ from an application's basic modules. However, every module must be implementable by at least one of FPGAs' building blocks. The mapping between application basic modules and the building blocks may not be one-to-one: two modules could be mapped to the same type of block, and one module could be implemented by several blocks. RVs of basic modules have to be mapped to RVs of FPGA resources so that the requirement of each resource type can be computed and FPGA resources can be allocated to satisfy the demand of users.

2.1. Mapping application module distributions to FPGA block distributions

Assuming that the distributions of design modules are given (the distributions may be dependent), they must be translated to distributions of FPGA blocks. Let A, B and C be design modules and W, X, Y and Z be resources available in an FPGA. The RVs corresponding to modules and resources are denoted by their name with subscript rv. Design modules implementable by a resource type X can be described by either 1) $X = S\{n_aA, n_bB, n_cC\}$ which means that one functional block of type X can implement n_a, n_b, n_c of module A, B, C, respectively, at the same time, or 2) $X = E\{n_aA, n_bB, n_cC\}$ which means one block of type X can implement just one type of modules A, B, and C at a time with the quantity specified.

For a given FPGA, we can describe its resources using above descriptions. Although one module type can be mapped to several combinations of resources, mapping design module distributions to those of FPGAs' building blocks to minimize the required area can be performed using resource areas. As our objective is to minimize the FPGA area, once such map is obtained for each module type, it will be used throughout the paper.

[1] Although the histogram of all circuits ever implemented in FPGAs may not be a normal distribution, a large set of their representative circuits used in FPGA architecture design is a normal distribution by the central limit theorem. The area of each resource type is at least an order of magnitude lower than an FPGA area. Thus, assuming continuous distribution is reasonable as the quantization error is limited.

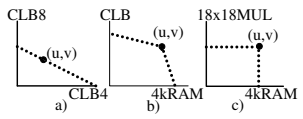

Fig. 1. Interactions between two types of resources. (See Section 2.2 for details)

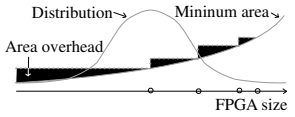

Fig. 2. Effects of a finite-member FPGA family.

2.2. Interactions between different types of resources

Some resource types are partially compatible, resulting in one being able to implement the other. Examples of resource usage interactions are shown in Figure 1 for two types of resources. Part (a) shows an example in which an FPGA has two types of resources: CLB4 and CLB8, each with 4 and 8 LUTs respectively. An FPGA having u resources of type CLB4 and v units of CLB8 can be shown as Point (u, v). A design requiring $u - 2$ and $v + 1$ resources can still fit on the FPGA because two CLB4 blocks can be combined to implement a CLB8. Thus, the interaction between these two resources is a line passing through (u, v) with the slope of -0.5 as shown in Figure 1(a). Any design that falls under the line can fit on the FPGA.

Now, assume that the resources are CLB and 4kRAM as shown in part (b) of the figure. If we move from (u, v) toward the y-axis, we use some 4kRAMs to implement CLBs. If we move toward the x-axis, some CLBs are used to implement 4kRAMs. However, the efficiency of using CLBs to implement 4kRAMs is different from that of using 4kRAMs to implement CLBs. Thus, the two lines have different slopes as shown in the figure. If the resources are 18x18MULs and RAMs which cannot implement each other, designs have to use no more than u and v of RAMs and 18x18MULs respectively to be able to fit in the FPGA and the interaction is shown in Figure 1c).

The resources exhibit a relation in Figure 1(a) if the two resources can implement each other with the same silicon area efficiency, or Figure 1(b) for different efficiencies. If they cannot implement each other, we have Figure 1(c). In general, the interaction of one resource with others can be described by a hyperplane, one for each resource type, as follows.

$$x_i + \sum_{j \neq i} a_{ij} \cdot x_j = b_i \tag{1}$$

, where $a_{ij} \in (0, \infty]$ is the efficiency of using x_i to implement x_j and b_i depends on a particular FPGA under consideration. For example, if the resource of type i and j are 4-LUT and 9x9 multiplier, respectively, one 9x9 multiplier can be implemented using 81 LUTs. Thus, $a_{ij} = 1/81$. If $a_{ij} = 0$, the resource of type j cannot be implemented using type i.

The maximum number of hyperplanes is n, where n is the number of resource types on an FPGA. However, it is possible that two resource types will have the same hyperplane, such as that in Figure 1(a). Thus, the minimum number of hyperplanes is one. In conclusion, interactions among n resource types can be described by m n-variable linear equations, where $m \leq n$.

3. FPGA FAMILY COMPOSITION

Assume that an FPGA has n types of functional blocks (resources). An FPGA can be represented as an n-tuple vector, $\vec{x}$, in which x_i is the number of blocks of type i available in the FPGA. Let $pdf(\vec{x})$ be the distribution of all mapped designs and $A(\vec{x})$ be the minimum area of an FPGA that can fit a design requiring $\vec{x}$, in this case x_i represents the number of resource of type i that the design needs. The total area of all FPGAs to implement all designs in $pdf(\vec{x})$ is

$$TotalArea = N \cdot \int A(\vec{x}) \cdot pdf(\vec{x}) d\vec{x} \tag{2}$$

where N is the number of designs, which is supposedly a large number. The probability density of the distribution toward the upper end is small because few applications exist that require huge resource requirements. Thus, it may not be practical to provide FPGAs for designs in that range. As a result, there is a specific amount of design coverage, say PMAX that our FPGA architecture should be able to implement.

As $pdf(\vec{x})$ is assumed given, thus the total area is determined by $A(\vec{x})$. The effect of $A(\vec{x})$ on the total area can be seen as follows. For simplicity, let's consider an FPGA with one type of resources. Consider Figure 2 which implements all designs using an FPGA family containing 4 FPGAs, shown as circles on the X-axis. The area overhead is shown as shaded regions. It can be easily seen that as the number of FPGAs in the family increases, the area overhead reduces and vanishes if there are infinitely many FPGAs in a family. FPGA family composition problem can be stated as follows.

Problem statement

For a given distribution of all designs, find a set of FPGAs containing at most M FPGAs that requires the minimum total area for covering PMAX of all designs.

Solving the problem is equivalent to choosing M points (FPGAs) in an n-dimensional space that collectively minimize the area overhead. Thus, due to its high complexity, the problem is solved in two steps: 1) Finding candidate FPGAs detailed in Section 4. The FPGA architecture solution space is multi-dimensional and hard to optimize. To reduce the number of architectural candidates, the solution space is projected to a one dimensional solution space by mapping the multi-dimensional space to a number indicating application coverage percentage. For each selected coverage percentage, a minimum area FPGA is determined. 2) Choosing at most M FPGAs from the candidates that minimize the total area. (See Section 5)

The above steps may not result in an optimal solution. Although the second step guarantees an optimum solution, its solution can be affected by the selected candidates. The overall solution may not be optimum because 1) there are a limited number of selected coverage points, and 2) the minimum area FPGA for a given coverage may not be unique, due to the given probability distribution and / or the introduction of errors in the numerical algorithm used.

4. A MINIMUM AREA FPGA COVERING A GIVEN PERCENTAGE OF DESIGNS

For a given set of hyperplanes obtained from Section 2.1, the total probability of all designs being covered by the FPGA can be computed by multiple integrals. However, such an approach would be complicated and inefficient. Thus, in this section, the transformation of the problem space to be efficiently solved will be discussed first and the optimization algorithm that can be applied in the transformed space will be presented next.

4.1. Transforming the hyperplanes

The percentage coverage by a specific size FPGA can be computed by multiple integrations with limits from hyperplanes. If hyperplanes are transformed to be axis-aligned ones, the percentage coverage becomes a cumulative distribution function (CDF), which can be computed efficiently. Although the CDF of a multivariate normal distribution has no closed form, an efficient computation for it is known [9].

The transformed hyperplanes will become axis-aligned if the normal vectors of the original hyperplanes are used in the linear transformation. This transformation will change the probability distribution. The transformed RVs may be correlated even if the original RVs are uncorrelated. The transformed distribution is a normal distribution if and only if the resulting covariance matrix is positive definite. This can be guaranteed if the linear transformation is linearly independent. However, if there are hyperplanes similar to case b) of Figure 1, it may happen that the covariance matrix is not positive definite and the obtained probability is not normal anymore.

4.2. Finding the minimum area FPGA

After the transformation, the probability that an FPGA with a specific amount of resources, $\vec{y}$, covers designs can be computed using a CDF. Furthermore, the area of the FPGA which is the summation of areas of all resources in the original problem, *ie*., $\vec{y} \cdot \vec{1}$, becomes $\vec{x} \cdot \vec{C}$, where $\vec{x}$ and $\vec{C}$ are $\vec{y}$ and $\vec{1}$ after the transformation. Thus, the minimum area FPGA that covers $a\%$ of all designs can be formulated as

$$\begin{aligned} minimize \quad f(\vec{x}) &= \vec{x} \cdot \vec{C} \\ s.t. \quad CDF(\vec{x}) &= a \\ B^{-1}\vec{x} &\succeq 0 \end{aligned} \tag{3}$$

where B is the linear transformation used in Section 4.1 and $\succeq$ means $\geq$ for every component. Since there is no closed form for the CDF of a multivariate normal distribution, it has to be computed numerically [9]. Thus problem (3) is in an oracle form. Thus, the optimization problem is difficult to solve because an initial feasible solution is difficult to obtain. The problem can be extended to (4). Later on we will show that the solution to (4) is the same as that of (3).

$$\begin{aligned} minimize \quad f(\vec{x}) &= \vec{x} \cdot \vec{C} \\ s.t. \quad 1 - CDF(\vec{x}) &\leq 1 - a \\ B^{-1}\vec{x} &\succeq 0 \end{aligned} \tag{4}$$

Let $x_2 = \mathcal{C}(x_1, \rho)$ defined by $P\{X_1 \geq x_1, X_2 \geq x_2\} = \gamma$, where (X_1, X_2) has a bivariate normal distribution with mean (0,0), variances $\sigma^2_{X_1}, \sigma^2_{X_2}$ and correlation ρ. It is shown that $\mathcal{C}(x_1, \rho)$ is strictly concave over x_1 for $\gamma \in (0, 1)$ and any value of ρ [10].

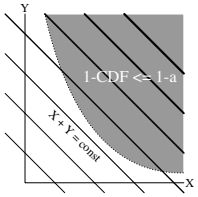

Fig. 3. Demonstration of (4) for two-variable case.

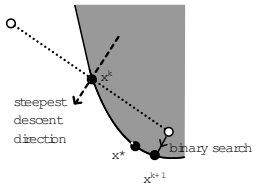

Fig. 4. Show how to compute x^{k+1}.

Furthermore, $\mathcal{C}(x_1, \rho + \epsilon) > \mathcal{C}(x_1, \rho)$, for $\epsilon > 0$ [10]. However, $CDF(x_1, x_2) = P\{X_1 \leq x_1, X_2 \leq x_2\}$. By linear transformation, a contour of $CDF(x_1, x_2) = \beta$ is strictly convex over x_1. As a result, $1 - CDF(\vec{x}) \leq 1 - a$ is also convex for 2-dimensional case as shown in Figure 3. Any marginal distribution of a multivariate normal distribution is also a normal distribution. Thus, we can infer that the feasible set is also convex in a higher dimension.

In the original problem, $\vec{y} \succeq 0$ is convex and thus its transformed counterpart is also convex. Because intersection of convex sets yield a convex set, the constrain of (4) is convex. A local optimum point x^* must satisfy

$$\nabla f(x^*)'(x - x^*) \geq 0, \quad \forall x \in X. \tag{5}$$

If f is convex over a feasible set X, x^* is also a global minimum. In our problem, (4), the objective function is linear, thus convex, and the feasible set is strictly convex, a local optimum x^* is the global optimum.

The algorithm we will present belongs to the feasible direction method which proceeds in an iterative manner. Let x^k be the current feasible solution, at iteration k.

$$x^{k+1} = x^k + \alpha^k d^k. \tag{6}$$

d^k is a feasible and descent (regarding objective function) direction, ie., $\nabla f(x^{k+1})' d^k < 0$ and α^k is a step size. If $x^{k+1} = x^k$, it means that $d^k = 0; \nabla f(x)' d^k > 0, \forall x \in X | \|x - x^k\| < \epsilon$. Therefore, x^k is the optimum solution. Traditionally, x^{k+1} is obtained by solving a subproblem. However, since we have the feasible set only in oracle form, we have to find x^{k+1} numerically by searching around x^k. Using the fact that our objective function is linear, a simple, yet effective search can be performed. Consider Figure 4. A hyper-ball of radius ϵ in $n-1$ dimensions centered at x^k can be drawn, where n is the dimension of the solution space. x^{k+1} is determined in 2 steps : 1) Randomly select a point on the ball. There are two points, shows as white circles, on the hyper-ball in Figure 4 and 2) Find a point x' along the contour $1 - CDF(\vec{x}) \leq 1 - a$ by performing a binary search along the steepest descent direction, $\nabla f = C$.

These two steps will be repeated until $x' \cdot \vec{C} < x^k \cdot \vec{C}$ and B^{-1}. $x' \succeq 0$ and that x' is used as x^{k+1}. Note that the chosen x^{k+1} may not yield the most cost reduction. However, the optimum solution is guaranteed to be found as the feasible set is convex, but may take more steps. It is possible that x^* is in an obscure location such that the feasible points oscillate between x^k and x^{k+1}. To ensure convergence, ϵ must be small and gradually decrease in size.

If the feasible region is contained by the perpendicular plane in the $-\vec{C}$ direction, it implies that we are at the optimal solution because $C' \cdot (x - x^k) > 0, \forall x \in X$ (C' is the transpose of C). Therefore, the optimality checking can be combined with x^{k+1} finding. In particular, if there is no feasible point during x^{k+1} finding, x^k is the optimal solution. However, to check for optimality, ϵ must be small. In contrast, using a large ball (i.e., using a large ϵ) in finding x^{k+1} may reduce the number of steps to get to the optimum solution. Thus, we choose to use two separate balls.

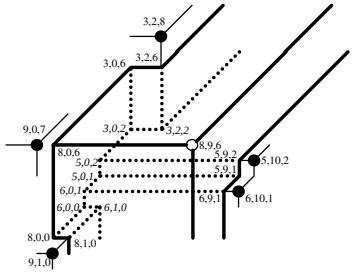

Fig. 5. The increase in coverage by adding one more FPGA (8,9,6) into a family is shown with bold solid lines. (The boundary of the previous coverage is shown using dashed lines.)

An initial solution can be any point within the feasible region. The choice will not affect optimality, but may affect run time. Note that not all the points in the solution space correspond to a valid FPGA due to discretization but in practice discretization had negligible effects in our implementation.

5. FINDING A GOOD FPGA FAMILY

The selected minimum area FPGAs for various percent coverage points may not be well-formed. As a result, adding a minimum FPGA that covers 80% of the applications to a family covering 60% of the applications does not necessarily mean that the family will cover 80% of the applications. Consider Figure 5. Imagine that only point (3,2,8), which covers 60%, is included in the family. Adding point (8,9,6) that covers 80% to the family will result in a total coverage of more than 80% because (3,2,8) covers some space which is not covered by (8,9,6).

The selection can be performed in a branch-and-bound manner. When one more FPGA is added into the family, the increase in coverage must be computed. Consider Figure 5 again. Assuming that (3,2,8),(9,0,7),(9,1,0),(6,10,1) and (5,10,2), all in black dots, are included, and (8,9,6) is going to be included. The volume of the increased coverage is enclosed by bold lines and dash lines, which are parts of the boundary of the coverage of the partial solution. As this volume is not a hyper-box, the plane sweep technique is used to partition it down to several hyper-boxes so that the probability they represent can be computed as CDFs. In general cases, included points may reside in the volume. Thus, the plane sweep technique is also used to break down these inner volumes so that their probability content can be computed, using CDF, and subtracted from the outer volume. For brevity, the algorithm is omitted. It is important to note that a minimum FPGA that covers PMAX applications must be included in a family.

6. EXPERIMENTS

The design basic modules used in our experiments were LUT, RAM and 9x9MUL (they were also used as RV names for the correspond-

Table 1. Parameters of normal distributions of design modules.

	LUT	RAM	9x9MUL
mean	5.824×10^3	1.743×10^4	4.460×10^1
sigma	2.055×10^4	2.042×10^4	8.862×10^2

ing modules) which are in units of number of LUTs, memory bits and multipliers, respectively. Many designs were collected from [3, 5, 1] and histograms of basic modules were plotted and fitted with normal distributions. The parameters of obtaining distributions are shown in Table 1. We assume that these distributions are independent. However, our proposed technique does not pose any restrictions on the dependency of distributions.

In the experiments, two types of FPGAs are considered: one Virtex-like FPGA containing only CLBs and block RAMs, and another with additional 18x18 bit multipliers. A CLB has 4 4-input LUTs and a block RAM is of size 4k bits. The RVs of FPGA resources are named CLB, 4kRAM and 18x18MUL for CLBs, block RAMs and multipliers, respectively. In the FPGA without the 18x18MUL, we have $CLB = E\{4 * LUT, MUL/20.5\}$ and $4kRAM = E\{4096 * MEM\}$. Thus, mapping from design basic module distributions to those of FPGA resources can be shown in Table 2. In contrast, for the FPGA with 18x18MUL, basic modules can be directly mapped to their corresponding blocks.

The logic part of a Virtex CLB implemented in $0.13\mu m$ technology takes $3660\mu m^2$ [11]. Projected to $65nm$, it takes 0.000915 mm^2. Based on the Virtex routing architecture description, routing resource per CLB is about $0.0011mm^2$. The CLB area is the summation of the logic and routing area. Block RAM of 4k bits takes $0.0208mm^2$ at $65nm$, projected from [4]. An 16x16 bit Booth multiplier takes $0.837mm^2$ in $0.6\mu m$ technology [12]. We believe that using an industrial, more aggressive design flow, 18x18 multiplier[2] should fit in the same area. Projected to $65nm$, the embedded multiplier should take $0.0098mm^2$.

Modules usually implemented in one resource type can also be mapped to other types. Although slower, particular logic functions can be implemented in multipliers. In the case that there are multipliers already on critical paths, MUXs not on critical paths can be implemented using multipliers without performance degradation [5]. On average, the number of LUTs reduces by 70, with the increase of 15 9x9 multipliers. A 4-input LUT is essentially a 16x1 memory. Thus, LUTs can be packed into larger memory blocks [6, 13]. One 9x9 multiplier can be implemented using 81 LUTs (with associated logic)[3]. Note that a multiplier can be implemented using memory but it requires exorbitant amount of memory [14]. Thus, implementing a multiplier using memory blocks is not considered in our experiments. Using this information, a linear transformation of FPGAs with 18x18 multipliers can be set up as shown in Table 3. For FPGAs without multipliers, the same transformation is used, but with the last column and row removed.

The algorithm detailed in Section 4.2 was applied to find a minimum area FPGA for each selected coverage point. The resulting resource area of the minimum area FPGAs with and without 18x18 multipliers are shown in Table 4. Data in each row corresponds to one percent coverage point. It can be observed that minimum area numbers increase with the percent coverage for both architectures. Furthermore, the minimum area increases faster for higher coverage percentages. The most important observation is about the area difference (column 9) between the two architectures (columns 4 and 8). We can see that FPGAs with 18x18 MUL require less area to cover the same percentage compared to FPGAs without multipliers. In addition, the difference tends to increase with percent coverage.

[2]Although benchmark circuits contain only 9x9 multiplier, recent commercial FPGAs provide dedicated 18x18 multipliers. Thus, an 18x18 multiplier will be used to implement one 9x9 multiplier

[3]Although there are several compact multiplier implementations, they are irregular and not suitable for implementation on LUTs. The carry save multiplier construction is the most common implementation.

Table 2. Mapping from design module distributions to FPGA resource distributions.

without MUL18x18	CLB	=	0.25*LUT + 20.25*MUL
	4kRAM	=	MEM/4096
with MUL18x18	CLB	=	0.25*LUT
	4kRAM	=	MEM/4096
	MUL18x18	=	MUL

Table 3. The linear transformation used in the experiments.

	CLB	4kRAM	18x18MUL
CLB	1	32/4096	1/20.25
4kRAM	15/4	1	0
18x18MUL	70/60	0	1

As a basis for a comparison, a baseline family selection was performed as follows. The designs collected were divided into 4 categories: 1) those use only CLB, 2) those use CLB and RAM, 3) those use CLB and MUL and 4) those use all 3 resources. Within each category, 3 designs were selected in such a way that other designs are dominated by the selected designs in term of resource usage. In some categories, only 2 designs were enough. But to reduce the area overhead (recall Figure 2), one more design is selected from the design at the median of the category. The module usages of the selected designs were mapped to resource usages of FPGAs with and wihout MUL using data from Table 2. At this point, there are 12 FPGAs for each architecture. For each FPGA, its coverage over the distribution shown in Table 1 was computed. Some FPGAs far away from the center of the distribution are large but have the comparable coverage as other small FPGAs. For example, one selected FPGA of size 22.7 mm^2 covers 61%, while another FPGA of size 0.27 mm^2 covers 62%. Thus, those FPGAs were eliminated from the family. As a result, the baseline family for each architecture contains 10 FPGAs. The baseline family of FPGAs with and without MUL covering 98% has an expected area of 6.266 and 6.475 mm^2, respectively.

Table 4. A minimum area FPGA for selected percent coverage points.

%	W/O 18x18 MUL			With 18x18 MUL				
	CLB	4kRAM	tol area	CLB	4kRAM	MUL	tol area	area dif.
cover	area	area	(mm^2)	area	area	area	(mm^2)	(mm^2)
0.35	0.237	0.063	0.300	0.014	0.222	0.018	0.254	0.046
0.40	0.902	0.009	0.911	0.289	0.051	0.428	0.767	0.144
0.45	1.528	0.135	1.663	0.872	0.121	0.391	1.384	0.279
0.50	2.162	0.082	2.244	1.476	0.235	0.257	1.969	0.275
0.55	2.786	0.020	2.807	1.990	0.020	0.534	2.545	0.262
0.60	3.427	0.032	3.459	2.660	0.009	0.138	2.807	0.652
0.65	4.072	0.177	4.248	3.163	0.287	0.537	3.987	0.262
0.70	4.771	0.004	4.775	3.898	0.099	0.112	4.109	0.666
0.75	5.514	0.119	5.634	4.562	0.412	0.190	5.164	0.470
0.80	6.350	0.082	6.433	5.344	0.178	0.182	5.704	0.729
0.85	7.328	0.245	7.573	6.216	0.373	0.208	6.797	0.776
0.90	8.547	0.150	8.696	7.432	0.154	0.183	7.770	0.927
0.95	10.358	0.020	10.378	9.085	0.073	0.129	9.287	1.091
0.98	12.384	0.074	12.458	10.936	0.204	0.501	11.641	0.817

Table 5. Effects of the number of FPGAs in a family.

# in a family M	without 18x18 MUL in a family				with 18x18 MUL in a family				over w/o MUL
	family	expected area (mm^2)	over 1 FPGA	over conventional	family	expected area (mm^2)	over 1 FPGA	over conventional	
2	00000100000001	6.805	0.454	-0.051	00000001000001	6.101	0.476	0.026	0.103
4	10000100001001	4.479	0.640	0.308	10000100010001	3.943	0.661	0.371	0.120
6	10001001010101	3.891	0.688	0.399	10010001010101	3.335	0.713	0.468	0.143
8	10010101010111	3.668	0.706	0.434	10000100111111	2.889	0.752	0.539	0.213
10	11010101011111	3.545	0.715	0.453	11010011011111	2.770	0.762	0.558	0.219
12	11101101111111	3.471	0.721	0.464	11111101101111	2.752	0.764	0.561	0.207
14	11111111111111	3.418	0.726	0.472	11111101101111	2.752	0.764	0.561	0.195

Using minimum area FPGAs listed in Table 4 , the algorithm, highlighted in Section 5, was used to solve the FPGA family composition problem, stated in Section 2, for family sizes, M, between 2 to 14. The result is shown in Table 5. The table is divided into 2 similar sections, for FPGAs with and without 18x18MULs. Each row contains data for a specific value of M, shown in the first column. The second column lists FPGAs without MUL from Table 4 selected for the family. The leftmost and rightmost bits represent FPGAs covering 35% and 98%, respectively. If the minimum area FPGA covering a particular percent coverage is included, the corresponding bit is marked 1 and 0 otherwise. The third column shows expected areas (eq. (2)/N). Area improvements, computed by $(ref - new)/ref$, of the families over having only the FPGA covering 98% are shown in the fourth column. The fifth column shows area improvements over the baseline selection. Column 6-9 have similar meanings to Column 2-5. To highlight the effect of having 18x18MUL in FPGAs, area improvements for families of the same size from the two architectures are shown in the last column. In general, we can see that there are more $1s$ around FPGAs with higher percent coverage. This is because the area increases faster for the higher coverage (See Table 4). Thus, to minimize area overhead, more FPGAs were selected from the higher percent coverage (See Figure 2). Interestingly, for FPGAs with 18x18MUL, the same family is obtained for both $M = 12$ and $M = 14$ because the volume in the transformed space covered by the minimum area FPGAs for 65% and 80% coverage can be collectively covered by FPGAs with smaller areas. For both architectures, the expected area decreases as M increases. Hence, the area improvement over one FPGA and conventional selection are also increased. The effectiveness of the proposed technique can be observed from the area improvements over the baseline (Column 5 and 9) even when M is smaller than that of the baseline family, whose M is 10. The benefit of having MUL in FPGAs is confirmed by the area improvement (column 10) and it increases with M from 10% at $M = 2$ to about 20% when $M \geq 8$.

7. CONCLUSION

An FPGA family composition problem was formulated in this paper. Due to the difficulty of the problem, it was solved in two steps. Minimum area FPGAs of selected percentage coverage points were chosen from the multi-dimensional solution space. For each percentage coverage point, a minimization problem was set up and shown to be convex, although in an oracle form. Thus, we presented an algorithm to solve it optimally within numerical errors. Among the resulting FPGAs, a family of limited FPGAs was formed by a simple branch-and-bound technique. However, as the solution space is multi-dimensional, minimum area FPGAs are not totally ordered. Thus, an algorithm using a plane sweep technique was created to compute the probability increase for including one more FPGA into the family. The result showed that the technique can produce families with the same number of members as the baseline families, but with area improvement of 0.453 and 0.558, for an architecture with and without multipliers, respectively. Although the experiment was carried out on a distribution projected from a small set of designs, the methodology proposed can be used for the more accurate distribution and the same benefit should be obtained. The process of mapping design distributions to those of FPGA resources was also discussed. Interactions among resources were described by hyperplanes making the FPGA family composition efficiently solvable after the transformation. However, the interactions may be nonlinear and should be address in future work.

References

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Febuary 2007.

[2] K. Rajagopalan and P. Sutton, "A flexible multiplication unit for an FPGA logic block," in *Proc. of Int. Symp. on Circuits and Systems*, 2001, pp. 546–549.

[3] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *Proc. of IEEE Int. Conf. on Field Programmable Technology*, 2006, pp. 1–8.

[4] J. R. Tony Ngai and S. Wilton, "An SRAM-programmable field-configurable memory," in *Proc. of IEEE Custom Integrated Circuits Conference*, 1995, pp. 499–502.

[5] P. Jamieson and J. Rose, "Mapping multiplexers onto hard multipliers in FPGAs," in *Proc. of IEEE-NECAS*, 2005, pp. 323–326.

[6] S. J. Wilton, "Heterogeneous technology mapping for FPGAs with dual-port embedded memory arrays," in *Proc. of Int. Symp. on Field Programmable Gate Arrays*, 2000, pp. 67–74.

[7] K. Eguro and S. A. Hauck, "Resource allocation for coarse-grain fpga development," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 10, pp. 1572–1581, October 2005.

[8] A. M. Smith, "Heterogeneous reconfigurable architecture design : An optimisation approach," *Ph.D Dissertation, Imperial College*, 2006.

[9] A. Genz, "Numerical computation of multivariate normal probabilities," *Numerical Computation of Multivariate Normal Probabilities*, vol. 1, no. 2, pp. 141–149, June 1992.

[10] D. P. Tihansky, "Properties of the bivariate normal cumulative distribution," *J. of the American Statistical Association*, vol. 67, no. 304, pp. 903–905, December 1972.

[11] V. Garg, V. Chandrasekhar, M. Sashikanth, and V. Kamakoti, "A novel CLB architecture and circuit packing algorithm for logic-area reduction in SRAM-based FPGAs," in *Proc. of Asia South Pacific Design Automation conference*, 2005, pp. 791–794.

[12] A. A. Katkar and J. E. Stine, "Modified booth truncated multipliers," in *Proc. of Great Lakes symposium on VLSI*, 2004, pp. 444–447.

[13] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. of Int. Symp. on Field Programmable Gate Arrays*, 1998, pp. 179–188.

[14] M. Wirthlin, "Constant coefficient multiplication using look-up tables," *J. of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, January 2004.