Statistical Design Space Exploration for Application-Specific Unit Synthesis*

Davide Bruni
DEIS – University of Bologna
Viale Risorgimento 2
40136 Bologna, Italy

Alessandro Bogliolo
DI – University of Ferrara
Via Saragat 1
44100 Ferrara, Italy

Luca Benini
DEIS – University of Bologna
Viale Risorgimento 2
40136 Bologna, Italy

ABSTRACT

The capability of performing semi-automated design space exploration is the main advantage of high-level synthesis with respect to RTL design. However, design space exploration performed during high-level synthesis is limited in scope, since it provides promising solutions that represent good starting points for subsequent optimizations, but it provides no insight about the overall structure of the design space. In this work we propose unsupervised Monte-Carlo design exploration and statistical characterization to capture the key features of the design space. Our analysis provides insight on how various solutions are distributed over the entire design space. In addition, we apply extreme value theory [11] to extrapolate achievable bounds from the sampling points.

1. INTRODUCTION

In complex systems-on-chip, behavioral synthesis (from C or HDLs) can be used as a basic tool for designing application-specific units (ASUs) for boosting performance beyond the limits of embedded software running on on-chip core processors [1]. An alternative, and more common [1], approach is to specify the ASU at the register-transfer level and use mature RTL synthesis technology to achieve high-quality results. The main shortcoming of the RTL-based methodology is that it requires manual translation of the behavioral specification (used by system designers) into a synthesizable RTL specification that can be fed to RTL synthesis tools. This process is fairly slow and error-prone. Behavioral synthesis advocates claim that the unwieldy behavioral-to-RTL translation may hinder design optimality, by slowing down the iterative process of exploring many alternative ASU implementations, and choosing the best one in the system context.

One of the main advantages claimed by behavioral synthesis with respect to RTL approaches is the capability of rapidly exploring many alternative implementations (a process known as *design space exploration*), with little or no designer intervention. The exploration can either lead to a satisfactory ASU design, or at least provide good insight and a few promising starting points for manually specifying (or refining) an optimal final implementations. The main contribution of this work is a technique for automatic design space exploration that effectively exploits behavioral synthesis coupled with statistic analysis and inference techniques.

The classical approach to behavioral synthesis (often called high-level synthesis or HLS for brevity), as summarized various textbooks [3, 4], is oriented towards optimization. Given a set of K cost metrics $f^{(i)}$, $i = 1, 2, \ldots, K$, behavioral synthesis targets the optimization of one of the metrics, say $f^{(1)}$ with constraints specified on the others. The cost metrics can be seen as functions defined over a complex space $\mathcal{A}$ (the *design space*), which represents the alternative choices in the implementation of the target specification. In symbols, $f^{(i)}(\mathbf{a}) : \mathcal{A} \rightarrow \mathcal{R}$, if we assume, without loss of generality, that the cost metric is real-valued. Clearly, fast and accurate estimators of the cost metrics are needed to drive constrained optimization towards optimal, feasible solutions.

The first generation of high-level synthesis tools [3, 4] adopted a *top-down* optimization approach. It relied on the assumption that the various $f^{(i)}$ could be approximated or bounded during synthesis even with partial knowledge, when the design point $\mathbf{a}$ is not completely defined yet. This assumption was motivated by the relative simplicity of the cost metric adopted (e.g., area of functional units, number of control steps), for which simple and local composition rules hold (e.g., additivity).

HLS-based design space exploration in these favorable assumptions is highly informative. In fact, it is possible to compute Pareto curves [4] for studying tradeoffs. Pareto curves for problems with two cost metrics $f^{(1)}$ and $f^{(2)}$ can be plotted by fixing $f^{(2)}$ as constraint, and running HLS to optimize on $f^{(1)}$. Besides Pareto curves, also design space bounding is significant piece of information for a designer [6, 7]. Bounding on a cost metric $f^{(1)}$ can be obtained by constructing an upper bound metric $f_M^{(1)} > f^{(1)}$, and a lower bound $f_m^{(1)} < f^{(1)}$, over the design space domain (i.e., $\forall\ \mathbf{a} \in \mathcal{A}$). The availability of Pareto curves and bounds provides deep insight on what performance level should be targeted for a given area budget or on what is the minimum latency that can be expected for the implementation of a given behavior.

Unfortunately, the optimistic assumptions on cost metrics made in first-generation HLS tools did not hold for complex designs and real-life design metrics. Real-life cost metrics such as average or worst-case execution time and power are not well behaved nor easily predictable under partial knowledge. This situation has worsened as technology scaled. As a result, design exploration based on behavioral synthesis has not fully met expectations, and it has found limited use in industrial design flows. The second generation of HLS tools [5, 2] has adopted a more conservative approach to-

*This work is sponsored by Mentor Graphics Co. through the Codesign Consortium

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

ward cost metric estimation. Second-generation commercial HLS tools do not claim the capability of performing constrained optimization, but they focus on the capability of synthesizing a functional designs that satisfies requirements and constraints specified up-front by the user (e.g. functional unit constraints, or synchronization requirements). Design exploration is just semi-automated: the designer has the responsibility of driving it by constraining the design in various ways and running the tool to obtain alternative implementations.

The third generation of HLS tools, which is currently the target of various academic research efforts [8, 9, 10], tries to gain back some automatic design space exploration capability by adopting iterative approaches. HLS is not a top-down optimization process, but an iterative search that prunes out sub-optimal solutions, and regions of the design space that are unlikely to contain optimal solutions. The outcome of the iterative process is a pool of "promising" solutions that warrant further investigation. Several heuristic search techniques have been tested (genetic algorithms, simulated annealing, etc.), and have produced promising results. The main shortcoming of these approaches is that they do not provide much insight on the overall structure of the search space and its bound. The output of the design space exploration process is a set of promising solutions, but there is no information on how sparse they are in the design space, how hard is to find them, how hard is to improve them, and how much can we expect to be able to improve them.

Our work is a first step in providing an answer to these open questions, by taking new point of view. Instead of providing on-line support to drive a supervised iterative design process towards optimal solutions, we propose unsupervised Monte-Carlo design exploration and statistical characterization to capture the key features of the design space. Our design exploration is conceived to be performed off-line, in order to collect statistical information before actually starting the design. Unbiased uniform sampling allows us to explore the entire space without making arbitrary assumptions about the position of the Pareto points. The purpose of the exploratory analysis is not (primarily) to find optimal solution, but to characterize the design space. More specifically the analysis provides insight on how various implementations are clustered in the design space (i.e. distributions) and how changes in a cost metric are likely to impact another metric (i.e. tradeoff scatter-plots and regressions). In addition, we show how to estimate achievable bounds by applying the extreme value theory [11] to the sampling points.

It is important to stress that the statistical analysis flow is fully automated and can be run with minimal supervision. Our tool embeds a second-generation commercial HLS tool, Mentor Graphics' Monet [12], as the behavioral synthesis engine, and it includes a complete RTL synthesis back-end [13] to support design cost metric evaluation at various levels of accuracy. Currently, area, execution speed and average power (or energy) are the supported metrics, however the tool can be easily extended to deal with other metrics, such as testability. To illustrate the supported analyses and the insight that can by gained by using the tool, detailed results are reported and discussed.

2. MONTE-CARLO EXPLORATION

Monte-Carlo *sampling* of design space $\mathcal{A}$ is the process of randomly generating design points $\mathbf{a} \in \mathcal{A}$. Generating a single sample point is a three-phase process that closely tracks the steps done by a designer using a behavioral synthesis tool for manual design space exploration. The step performed by a Monet user to instantiate a design can be outlined as follows:

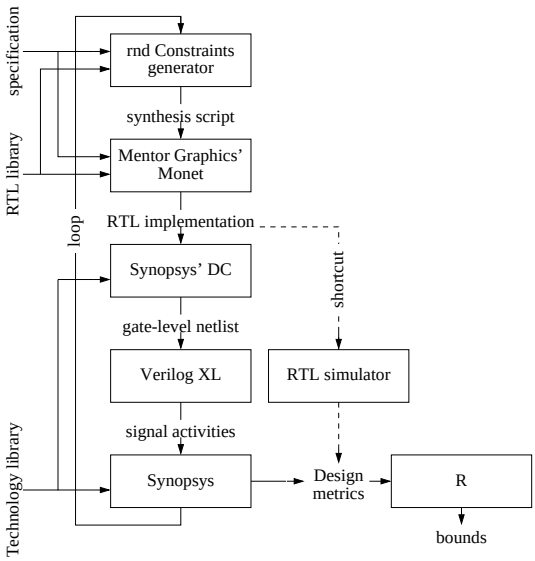


Figure 1: Monte-Carlo design space exploration flow

- *Constraint specification.* In this step the designer specifies constraints to drive behavioral synthesis.
- *Push-button behavioral synthesis.* Monet is run through the various behavioral synthesis sub-steps (allocation, scheduling, control-datapath separation, binding) to produce a final RTL netlist. No user intervention is required.
- *Design evaluation.* The RTL netlist is evaluated for various cost metrics (area, delay, power). This step may employ RTL estimators, or it may involve RTL synthesis followed by gate-level estimation.

Notice that the choices made in the first step fully define, in an implicit fashion, the final implementation. In other words, a sample design **a** is automatically constructed once the synthesis constraints are specified. The high-level synthesis tool can be viewed as a push-button mapping from a set of allocation constraints and synthesis options to an RTL implementation of the given specification. According to this view, random sampling is actually performed on the constraint space.

The iterative Monte-Carlo sampling process is detailed in Figure 1, and can be summarized as follows. First allocation constraints are randomly generated, then Monet is run to obtain a RTL netlist, which is passed to the cost estimation engine. Currently, two estimation engines are supported: the first runs directly on the RTL netlist, while the second calls RTL synthesis, and estimates costs on the final mapped netlist. Clearly, the second estimation flow is much more computationally intensive than the first one, but it is more accurate. In the case studies described in the following sections, we used the accurate estimation engine.

In more detail, the Monte-Carlo exploration tool contains 3 commercial tools coordinated by Perl scripts for driving the flow from high level synthesis to final gate-level (or RTL) implementation and cost metric estimation. Behavioral synthesis is performed by Monet. RTL synthesis (when needed) is performed by Synopsys design compiler. Accurate gate-level estimation of power, area and speed are still performed in the Synopsys environment (area, timing, power reports). Switching activity data, required for accurate power analysis, is obtained by logic simulation with Verilog-XL. At

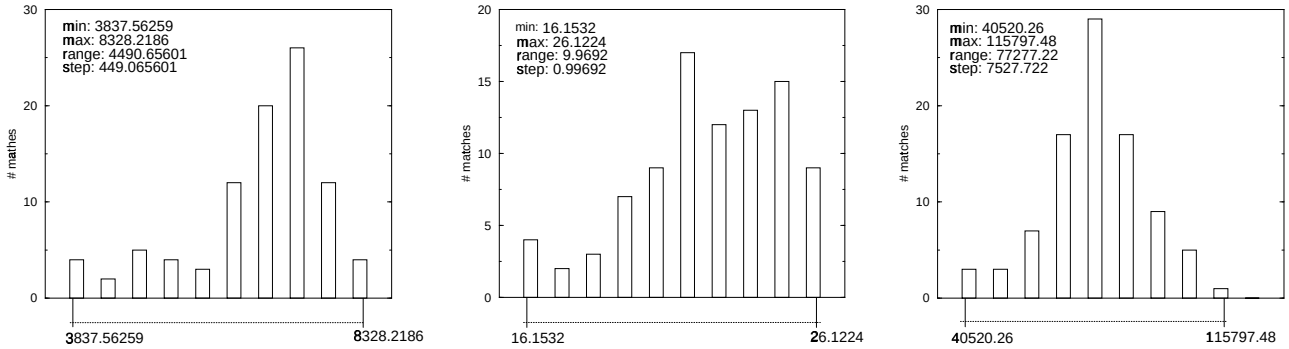


Figure 2: Probability densities of Area, Performance and Power metrics on the design space of `ellipf`.

a reduced level of accuracy and computational effort, RTL estimation employs Monet's timing and area estimates. For power analysis, we use an academic RTL power estimator based on macro-modeling [15].

With reference to Figure 1, the first step of a single Monte-Carlo run is the generation of a command file (*behavioral synthesis script*) for Monet that contains the constraint parameters for components (quantity and type) and clock period. The generation of component constraints is random within a compatibility list for behavioral operations provided by Monet. From the output of the behavioral synthesis tool, several useful information are collected, namely, the RTL implementation of the circuit, the estimated area and cycle time.

At the end of this phase, there are two choices. If the high-effort estimation path is chosen, the flow goes through RTL synthesis and Verilog simulation of gate level implementation, with switching activity collection. Back-annotation of switching activities by Synopsys design power is the last step of a single Monte-Carlo iteration that produce the final gate-level power estimation for the considered random trial. Area and critical path are estimated on the mapped netlist as well. If the low-effort estimation is chosen, RTL synthesis is bypassed. Area and speed estimation are provided directly by Monet. For power estimation, the RTL netlist is passed to a RTL power estimator implemented as a PLI add-on to Verilog-XL [15]. Clearly, the low-effort path is considerably faster than the high-effort one, but accuracy is reduced. In quantitative terms, accuracy is in average within a 25% error bound and speedup is approximatively within a factor of 5. Details on how RTL power, speed and area estimation is performed are out of the scope of this work. Refer to [15] and [12] for more information. It is important to stress that the flow is fully automated, given a sample size specification and a behavioral description of the target design in Verilog/VHDL.

3. DESIGN SPACE CHARACTERIZATION

The tool flow described in the previous section generates a random sample $\mathcal{S} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}$ of feasible designs, taken from the design space $\mathcal{A}$ of a given specification, together with accurate estimates of the corresponding cost metrics $f^{(i)}(\mathbf{a}_j)$, $i = 1, 2, ..., K$, for all $\mathbf{a}_j \in \mathcal{S}$. The K-tuple of estimates $\mathbf{f}_j = (f^{(1)}(\mathbf{a}_j), f^{(2)}(\mathbf{a}_j), ..., f^{(K)}(\mathbf{a}_j))$ associated with a given implementation $\mathbf{a}_j$ provides the coordinates of the point that represents $\mathbf{a}_j$ in the K-dimensional design space. In this section we study the distribution of the sampling points in order to capture some features of the design space they belong to.

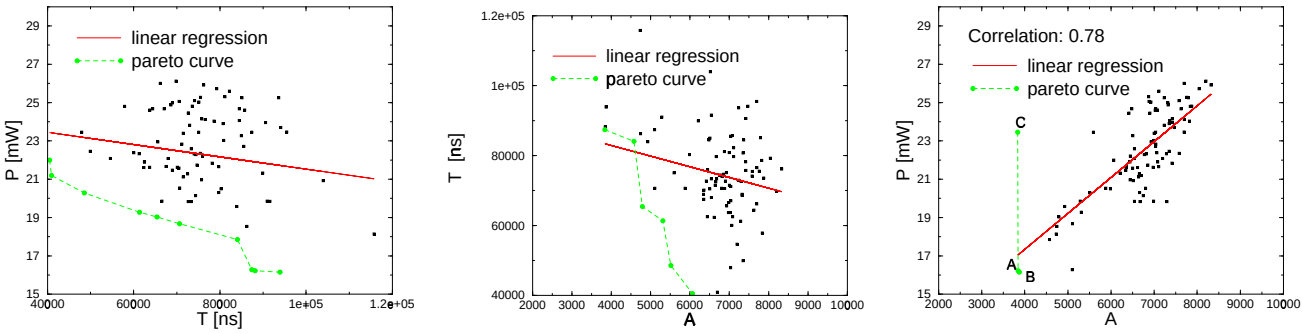
For each implementation we collect a large set of data, including the complete set of HLS constraints (providing the information required to possibly reproduce the solution for further analysis), the latency, the critical path, the number of clock cycles needed to complete execution, the number of registers, the number of MUXes, the type and number of resources instantiated during HLS, the total area and the total energy. Although all these data can be used to parameterize the design space, in the following we focus only on three parameters representing typical area, power and performance metrics:

- A, representing area as the total gate count;
- T, representing performance as the total execution time: $T = Nclk \cdot Tclk$, where $Nclk$ is the total number of clock cycles, and $Tclk$ is the minimum clock period, evaluated as the sum of the combinational critical path and a technology-dependent overhead taking into account registers setup time and design margins;
- P, representing average power as the ratio between total energy and total execution time.

Figure 2 shows the distribution of the values of A, T and P over a sample of 100 feasible implementations of `ellipf`. We observe that all parameters are spread over a wide range of values, indicating that the choice of HLS constraints has a great impact on all design metrics, clearly motivating design space exploration.

Furthermore, notice that although all distributions have the expected bell shape, in some cases (e.g. figure 2 (a),(b)) the bell is skewed or it has long tail. This indicates that the distribution is not a simple Gaussian, and that the search space can have clusters of solutions not only around the mean value. Distribution analysis can be used for driving iterative optimization process (such as the one proposed in [9]). For instance, if the distribution is symmetric and short-tailed, we can expect a fairly smooth design space, and we can tune aggressively the local convergence parameters of iterative search. On the contary, in presence of long tails and multiple clusters, we should expect multiple local minima with large attraction regions around them. Hence we should emphasize randomization in iterative search, to help excaping from local minima. Cluster analysis on the distribution data may even provide information on the approximate location of local minima.

Scatter-plots of Figures 3 (a), 3 (b) and 3 (c) represent the projection of the sampling points on the power-performance plane, on the performance-area plane and on the power-area plane, respectively. Solid lines on the plots represent linear regressions, while dashed lines represent in-sample Pareto curves, i.e., the piece-wise-linear



(a) (b) (c)

Figure 3: Design space projections of `ellipf`

interpolations of the best tradeoffs in the sample. All plots refer to the `ellipf` benchmark, hereafter used as a representative case study for reporting and discussing detailed experimental results.
Notice that most of the points lie very far from the in-sample Pareto curve, motivating automation of the exploration and optimization efforts at this level of abstraction. A manual constrain-evaluate process, as supported by second-generation behavioral synthesis tools, just produces a few sample points which may be very far from the Pareto curve.

Moreover, Figures 3 (a) and 3 (b) are substantially different from Figure 3 (c). In fact, while there is a weak negative correlation between P and T (Figure 3 (a)) and between T and A (Figure 3 (b)), there is a strong positive correlation between P and A (Figure 3 (c)). This is shown by the regression curves on the plots and quantified by cross-correlation coefficients -0.16 for P and T, -0.24 for T and A and 0.78 for P and A.
It is also worth discussing the peculiar shape of the Pareto curve reported in Figure 3 (c). For positively-correlated metrics, Pareto curves should in principle reduce to a single point representing the solution that simultaneously optimizes both parameters. In our sample there were two points, denoted by A and B in Figure 3 (c), that were good candidates to approximate the degenerate Pareto curve. However, there was also a third point, denoted by C in the plot, associated with a completely different solution providing minimum area at a cost of a 50% increase in terms of power. Hence, the scatter plot of Figure 3 (c) gives rise to two observations. On one hand, the high correlation between power and area suggests that area-reduction techniques could be applied to obtain low-power designs. On the other hand, the presence of point C clearly states that area and power optimization are not the same problem and cases exist were unaware area minimization may impair power reduction.

4. DESIGN SPACE BOUNDING

In this section we focus on the tails of the distributions of the design parameters in an attempt of extrapolating from the sample realistic bounds for the design space. Bounding the design space is a challenging task for two main reasons: first, the probability of observing the tails of the distributions during uniform sampling is small by definition; second, non-trivial feasible bounds are usually beyond the largest (smallest) observed values. We address these issues in the framework of extreme value theory [11] (EVT), a branch of statistics devoted to the characterization of the extreme behavior of probability distributions.

4.1 Extreme Value Theory (EVT)

The value taken by each metric (e.g., power P) over the entire design space can be viewed as a random variable with an unknown distribution $F(x) = Pr\{P \leq x\}$. Let us denote by $P_{M(n)}$ the maximum of n independent observations of P:

$$P_{M(n)} = max\{P_1, P_2, \ldots, P_n\}$$

It can easily be shown that $P_{M(n)}$ is a random variable with distribution $F^n(x)$. We denote by $p_{n,\varepsilon}$ the value of P for which $Pr\{P_{M(n)} > p_{n,\varepsilon}\} \leq \varepsilon$, i.e., a point that has a probability less or equal than ε of being passed by at least one of n observed values of P. For large values of n and small values of ε, $p_{n,\varepsilon}$ provides a good estimate of the extreme point of the distribution of P.

In our case, however, the parent distribution $F(x)$ is unknown, making it impossible to compute $F^n(x)$ and, ultimately, to apply the above methodology. Here is where EVT [11] comes into the picture by demonstrating that, regardless of the unknown parent distribution $F(x)$, the distribution of its extreme values converges to a family of known distributions (hereafter denoted by $G(x)$) as n tends to infinity. Hence, distribution $G(x)$ can be characterized and used in place of $F^n(x)$ to compute $p_{n,\varepsilon}$. Characterization of $G(x)$ requires a sample of m observed values of $P_{M(n)}$.

4.2 EVT and Design Space Bounding

Monte-Carlo design exploration provides a sample $\mathcal{S}$ of N independent values of each design metric (we keep using P as an example metric). We partition the original sample $\mathcal{S}$ in clusters of size n and we take the maximum value of P from each sample. What we obtain is a sample of m observed values of $P_{M(n)}$, with $m = N/n$:

$$\mathcal{S}_{M(n)} = \{P_{M(n),1}, \ldots, P_{M(n),m}\}$$

Assuming that n is large enough to consider $G(x)$ a good approximation of the actual distribution of $P_{M(n)}$, $\mathcal{S}_{M(n)}$ can be used to characterize the mean (μ) and the variance (σ) of $G(x)$. Then, $G(x)$ can be used to obtain $p_{n,\varepsilon}$, for a given ε.
The quality of $p_{n,\varepsilon}$ as a bound for the design space depends on the order n of the observed maxima, and on the number m of available observations. In particular:

- for small values of n, $P_{M(n)}$ does not represent a significant extreme point;
- for small values of n, $G(x)$ is not guaranteed to be a good approximation of $F^n(x)$;
- for small values of m, the size of $\mathcal{S}_{M(n)}$ does not guarantee the statistical significance of the estimated values of μ and σ.

In summary, the quality of the bound depends on the number of available points in the original sample: $N = n \cdot m$.

We performed several experiments to test the strength and robustness of EVT when applied to design bounding. For this purpose the Splus `excode2.s` code package [16] was adapted for use with R [14]. This package provides several functions for GEV fitting and extreme value profiling. Results discussed hereafter refer to the minimum-achievable power consumption of benchmark circuit `ellipf`. Figure 4 plots the estimated lower bound as a

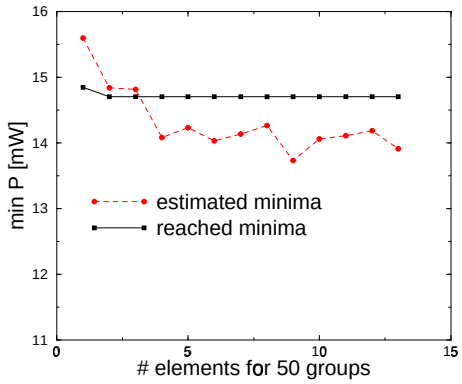


Figure 4: Plot of the estimated minimum power of `ellipf` as a function of the number of order (n) of extreme sample points.

function of the order n, for fixed $m = 50$ and $\varepsilon = 0.01$. The total number of samples required to compute each lower bound were $n \cdot m = 50n$. The observed minimum power is also plotted for comparison. We remark that the lower bound is a decreasing function of n that starts above the observed minimum (for $n = 1$ and 2) and then stabilizes below the minimum observed value. For $n = 1$, the population of order-1 minima coincides with the parent population of P. The reason why the estimated minimum is above the observed minimum is two-fold: first, for $n = 1$ the asymptotic tail equivalence of $G(x)$ and $F^n(x)$ is not demonstrated; second, $p_{n,\varepsilon}$ with $\varepsilon = 0.01$ represents a value that can be reached and passed by the 1% of the observed values. Nevertheless, for values of n larger than 3, $p_{n,\varepsilon}$ starts providing useful information by extrapolating a lower bound that is below all observed data.

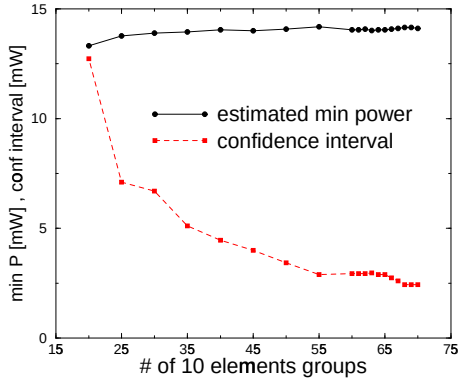


Figure 5: Plot of the estimated minimum power of `ellipf` as a function of the number of m extreme order-10 values.

Figure 5 plots the estimated bound as a function of m for fixed $n = 10$ and $\varepsilon = 0.01$. In this case, what changes is the statistical

Benchmark	max (reached)	max (estimated)	d%
ellipf	26.12 mW	28.58 mW	+ 8.6
dft	2.56 mW	2.98 mW	+ 14.1
fir	13.12 mW	14.11 mW	+ 7.0
my_sample	8.12 mW	8.25 mW	+ 1.57

Table 1: EVT applied to maximum power estimation for different benchmarks.

significance of the estimates of μ and σ, that ultimately impacts the confidence interval of the lower bound. The decreasing behavior of the 95% confidence interval is also plotted in the graph. We observe that it is rapidly decreasing for values of m from 20 to 50, while it is almost constant for larger values of m, indicating that $m = 50$ is a good trade off between sampling size and statistical significance. Figure 6 shows the joint effect of the two previous graphs by using

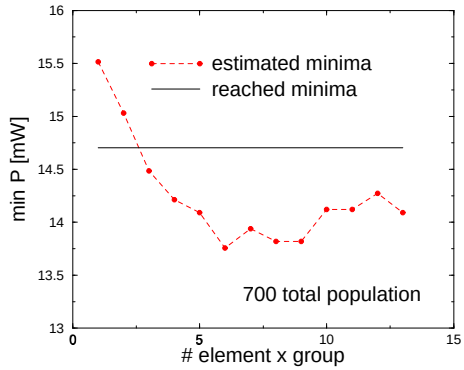


Figure 6: Plot of the estimated minimum power of `ellipf` as a function of the number of order (n) of extreme sample points, for a fixed number of samples in $\mathcal{S}$.

a constant number of $N = 700$ samples, while changing both n and m in such a way that $n \cdot m = N$. For small values of n (left-hand side of the graph) the low order makes the lower bound close to the observed values, while for large values of n the reduced number of observations makes the estimate unstable.

It is important to remark that the impact of m and n on the extreme value estimates does not depend on the benchmark, so that the case study discussed in this section provides general guidelines for the application of EVT to design bounds.

Finally, Table 1 shows the results of application of EVT to compute power upper bounds for a set of behavioral benchmarks. The second column reports the largest power value observed in the sample, the third shows the EVT upper bound, and the fourth reports the percentage difference between the two. The upper bound estimates are obtained with the same sample size. The table shows that the tightness of the bounds does not depend on sample size, but it is a function of the sample distribution. In other words it is a property of the topology of the design space.

5. CONCLUSION

Optimization-oriented design exploration usually performed by HLS tools provides a partial view of the design space, affected by the need of efficiently finding optimal solutions. In this paper we perform unbiased design exploration to provide a complete view of the design space and capture its main features. We have developed a tool flow that automatically performs sampling and characteriza-

tion of the entire design space before actually starting the design process, so that efficiency is not a critical issue.
We have reported and discussed representative experimental results, showing examples of the type of results and insights that can be obtained by the statistic design space exploration approach. Moreover, we have shown how to apply extreme value theory to the estimation of feasible bounds for the design space. Design space characterization and bounding can be used to find optimal trade-offs, to decide how to escape from local minima, to evaluate how far we are from the global optimum and to evaluate how hard it will be to get it.

6. REFERENCES

[1] R. Gupta, G. De Micheli (eds.), "Hardware/Software Codesign," Special Issue, *IEEE Proceedings*, vol. 85, no. 3, March 1997.

[2] D. Knapp, *Behavioral Synthesis: Digital System Using the Synopsys Behavioral Compiler.* Prentice Hall, 1996.

[3] D. Gajski, *High-Level Synthesis: Introduction to Chip and System Design.* Kluwer, 1992.

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

[5] Y. Lin, "Recent Developments in High-level Synthesis," *ACM Transaction on Design Automation of Electronic Systems*, vol. 2, no. 1, pp. 2–21, Jan. 1997.

[6] J. Rabaey, M. Potkonjak, "Estimating implementation bounds for real time DSP applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 6, pp. 669–683, Jun. 1994

[7] S. Chaudhuri, R. Walker, "Computing lower bounds on functional units before scheduling," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 4, no. 2, pp. 273–279, Jun. 1996.

[8] A. Raghunathan, N. Jha, "SCALP: an iterative-improvement-based low-power data path synthesis system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1260-1277, Nov. 1997.

[9] K. Khouri, G. Lakshminarayana, N. Jha, "High-level synthesis of low-power control-flow intensive circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1715–1729, Dec. 1999.

[10] C. Chantrapornchai, E. Sha, X. Hu, "Efficient Design Exploration based on Module Utility Selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 19–29, Jan. 2000.

[11] R. Reiss, M. Thomas, *Statistical Analysis of Extreme Values*, Birkauser, 1997.

[12] Mentor Graphics, *Monet Data Sheet*, http://www.mentor.com/monet

[13] Synopsys, *Logic Synthesis Products*, http://www.synopsys.com/products/logic/

[14] R. Gentleman and R. Ihaka, *The R Project*, http://www.R-project.org/

[15] M. Barocci, L. Benini, A. Bogliolo, B. Riccó and G. De Micheli, "Lookup table power macro-models for behavioral library components," *IEEE Alessandro Volta Memorial International Workshop on Low Power Design*, pp. 173–181, March 1999.

[16] S. Coles *Extreme value theory and applications*, http://www.maths.lancs.ac.uk/∼coless