

Soft-core Processor Customization using the Design of Experiments Paradigm

Abstract

Parameterized components are becoming more commonplace in system design. Customizing parameter values for a particular application, or tuning, can be a challenging designer task. We focus on the problem of customizing a parameterized soft-core microprocessor for best performance of a particular application, subject to a size constraint. We map the problem to the well-established Design of Experiments (DoE) paradigm, which involves careful creation of a set of experiments and sound analysis of generated results to yield maximum information about input factors of the experiment. We apply DoE methods to obtain maximum information regarding a soft-core's parameters' impacts on performance for a particular application, using only a small number of synthesis/execution runs. That information in turn guides a soft-core tuning heuristic. We show that the multi-factor-based DoE method, results in application speedups of 6x-17x versus an un-tuned base core, representing 3x-6x speedups compared to a previous single-factor-based tuning method.

1. Introduction

Soft-core processors are becoming increasingly common in modern technology. A soft-core processor is a programmable processor that can be synthesized to a circuit, typically integrated into a larger system existing on an application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA). Popular commercially available soft-cores include ARM [1], Tensilica [15], Microblaze [16], and Nios [2]. Several trends of modern technology catalyze soft-core usage, including more stable synthesis tools, higher-capacity ASICs and FPGAs, new commercial toolsets for application-specific instruction-set processors, and increasing demands for high-performance and low-power embedded processing.

Whereas traditional pre-fabricated processors must be optimized for good performance across an entire domain of applications, soft-cores can instead be customized to the particular applications they execute. For example, a particular application may perform best on a processor having a large cache and a floating-point unit, while another application may instead require a hardware multiplier and have no need for a cache. Nearly all soft-core providers therefore include parameters that may be customized by a soft-core user. Common parameters include instantiatable coprocessor units such as hardware floating-point, multiplier, divider, or barrel shifter units; cache architecture settings such as the use of one or two levels of cache, separate versus unified cache, cache size, line size, associativity, and write back policy; and processor micro architecture features such as pipeline depth, bypass logic, or register-file size. The above-referenced soft-core processors each contain more than 10 customizable parameters, with the trend in newer core versions being towards more parameters. In this work, we only consider customizations that consist of tuning parameter values; such customizations are in a different category than are application-specific instruction-set extensions [15], which involves design of new hardware and introduction of new

instructions. Of course, an instruction-set extension approach could wrap extensions using automation such that extensions are presented to the user as a parameter (e.g., choose to use instruction-set extensions A, the set B, or none).

Customizing a soft-core by tuning parameter values can yield improved performance, lower-energy, and/or smaller size. A particularly interesting newly evolving size-constrained scenario involves dozens of FPGA soft-cores competing for limited hardware-resources [10].

Soft-core providers provide little or no automated support to assist users in customizing a soft-core's parameters to a particular application, other than providing simulation tools. Thus, users must manually select and simulate (or implement) a user-determined set of candidate soft-core configurations to find the best configuration for a particular application. Each such simulation may require tens of minutes, limiting the number of candidate configurations that can be examined. Some recent research addresses automated soft-core configuration using custom heuristics developed for a particular soft-core's parameters [14]. However, those custom heuristics do not scale to handle a broader range of parameters, as we will show.

In this paper, we propose to apply the well-established scientific paradigm known as *Design of Experiments (DoE)* [9][12] to the problem of soft-core customization. DoE has been developed for over 80 years by scientists seeking to optimize parameterized physical phenomena through a minimum number of carefully chosen experiments, where such experiments are costly and/or time-consuming and thus should be kept to a minimum. The main idea is to design the set of experiments such that subsequent analysis of the resulting data provides maximum information about each parameter's impact on metrics of interest. We claim that the work in this paper makes two contributions. First, the work results in a more robust soft-core customization methodology, which can not only handle a wider variety of parameter types than could previous approaches, but which also results in better-customized soft-cores having improved performance than obtained by previous heuristics. Second, to the best of our knowledge, this work represents the first employment of the DoE paradigm in design automation research literature. With the increasing appearance of highly-parameterized intellectual property components for use in system-on-chip designs, use of the robust DoE paradigm may prove useful for a variety of design automation problems involving customization any of a large spectrum of parameterized components or platforms.

2. Previous Work

Kumar [7][8] showed the benefit of multi-core general-purpose processor chips having heterogeneous rather than homogenous cores. They considered superscalar processor parameters related to cache, instantiations of floating-point, multiply, and arithmetic-logic units, and sizes of the register file, translation lookaside buffer, and load/store queue, yielding 480 possible single-core configurations. Via exhaustive search, they showed an optimally-configured four-core system to have up to 40%

better performance for a given workload, versus the best homogeneous four-core system for that workload.

Givargis [5][6] developed a tuning approach for parameterized system-on-a-chip platforms, considering parameters related to cache, bus, processor voltage, and a few parameters in peripherals. They used a user’s denotation of independent subsets of parameters to extensively prune the configuration space before searching dependent parameters exhaustively or using heuristics. They showed roughly 5x tradeoffs between power and performance for different applications.

Sekar [13] discusses trends toward highly-parameterized platforms, including parameterized processor cores, peripherals, caches, etc., and then describes a technique for dynamically tuning a processor’s voltage and frequency.

Yiannacouras [17][18] developed a framework for generating and customizing a soft-core for FPGAs, with parameters including hardware versus software multiplication, different shifter implementations, and pipeline depth. They showed 30% improvements obtained by optimally tuning soft-core parameters to an application, using exhaustive search to carry out the tuning. Their work motivated the need to develop efficient automated customization heuristics.

Sheldon [14] developed heuristics for soft-core parameter tuning. Their approach assumes that synthesis and execution (or simulation) of soft-core configurations, rather than pure estimation approaches, is essential for accurate evaluation of FPGA soft-cores, due to the tremendous variation of soft-core performance for different applications and across the hundreds of different FPGA devices by an FPGA vendor. Because synthesis/execution runs are costly, requiring tens of minutes or more, they developed several tuning heuristics that utilized only about a dozen synthesis/execution runs, thus executing in 1-2 hours. They considered a Xilinx Microblaze soft-core processor whose parameters each involved the option of instantiating a hardware component, including a hardware multiplier, barrel shifter, divider, floating point unit, or a fixed-sized cache. That work showed 2x application speedups of a customized core versus a base core having no optional components instantiated.

Sheldon’s best heuristic (as well as his other heuristics) used what we will call a “single factor” analysis. The heuristic was guided by the speedup versus the base core when instantiating *exactly one* (single factor) of the core’s optional hardware components. The heuristic then sorted each component by the ratio of its speedup over size, yielding an “impact-ordered tree” of parameters, which the heuristic then descended (encountering two choices per tree level) to find a solution. While a single-factor analysis is effective for on/off-type parameters, such an approach lacks an obvious extension for parameters that have two non-zero values (which value would be the base value?) or that have three or more values. Furthermore, a single-factor analysis may be inaccurate if parameters are interdependent. For example, neither of two components may individually yield speedup, but the two together may; conversely, two components may individually each yield speedup, but instantiating both may yield little benefit beyond instantiating just one, due to functionality overlap. In contrast, the approach we introduce performs a multi-factor analysis, supporting multi-valued parameters and considering interdependent parameters, as will be described.

3. The Design of Experiments Paradigm

Design of Experiments (DoE) [9][12] is a scientific paradigm for carefully creating a small set of experiments that provide maximum information on how the experiment’s parameters, or *factors*, influence output and interact with one another. The basic assumption in DoE is that experiments are costly (with respect to money and/or time) and thus only a few can be conducted. DoE was originally created by Sir Ronald Fisher in the 1920s for agriculture (in which factors might for example involve different watering schedules or fertilizer types, and output might be crop quality), and has since expanded to a huge variety of domains, including even management of workers.

DoE produces three types of statistics. The first identifies the extent of positive or negative impact that each factor has on the output, where a factor is an input variable of the experiment. The second indicates whether each factor is beneficial to the experiment. The third describes how factors interact with one another. For a given number of allowed experiments, where each experiment involves setting each factor to a specific value and then measuring the output, DoE seeks to design experiments providing maximum quality of statistical information.

DoE experimental design is based on four main principles: randomization, replication, blocking, and orthogonality. Randomization means that the experiments are performed in a random order. Replication means that each experiment is repeated. Blocking is the process of grouping different experiments into a group and running those experiments in the same environment. Orthogonality is the process of creating the experiments so that one factor can be analyzed independently of the other factors. Orthogonality is the principle of key interest to our work.

Several experimental design methods have evolved to create a set of runs that satisfy the orthogonality, as well as the other, principles. The *factorial* method, typically called the exhaustive search method by the design automation community, examines all possible combinations of the factors, thus providing maximum information, but obviously being infeasible in most situations. The *fractional factorial* method runs only one half to one eighth of the total number of runs of the factorial design, increasing feasibility to more situations, but still sometimes involves too many runs. In contrast, the *Taguchi* and the *Plackett-Burman* methods generate just a few more runs than there are factors, thus using a linear rather than exponential number of runs with respect to the number of factors, at the expense of less accurate information obtained about the factors.

DoE uses various statistical techniques to analyze the data resulting from a set of runs. *Analysis of variance* (ANOVA) techniques separately analyze the variation of each of factor. *Multiple regression* techniques determine whether a factor is statistically significant.

We observed that soft-core customization problem can be mapped to the DoE paradigm. In that mapping, the parameters of a soft-core correspond to factors of an experiment. The synthesis and execution/simulation of an application on a configured soft-core corresponds to an experiment, with such an experiment being “costly” due to the time required¹. The application’s

¹ Although in DoE “costly” typically means costing thousands of dollars or requiring weeks or months of time, cost is a relative subject – in design automation, 10 minutes is costly.

performance on a configured soft-core corresponds to the experiment’s output.

Therefore, our idea is to use DoE experimental design and analysis methods to customize a soft-core for an application. In contrast to Sheldon’s single-factor analysis approach [14], a DoE approach considers interactions among parameters. We thus hypothesized that a DoE approach would yield better results than the previous single-factor method.

4. Experimental Methodology

We describe the features of our experimental setup for soft-core tuning using DoE methods. Our soft-core tuning objective is to find a soft-core configuration that maximizes performance of a given application, subject to a size constraint.

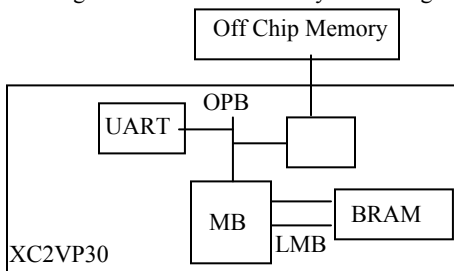
4.1 Soft-Core Framework

We use a Xilinx Microblaze soft-core processor on a Virtex 2 Pro device, illustrated in Figure 2. The system consists of a UART for external communication, an on-chip BRAM accessed through the Xilinx’s LMB (local memory bus), and access to off chip memory via the Xilinx’s OPB (on-chip peripheral bus). We used Xilinx’s Virtual Platform tools to simulate designs.

The soft-core parameters were:

- Instantiatable datapath units – The Microblaze has six on/off parameters: barrel shifter, floating point unit unit, multiplier, divider, MSR instructions, and comparator. Each parameter could be set to on, which would instantiate the hardware unit, or to off, which would carry out the corresponding function in software.
- Cache type – The Microblaze has two cache types for instruction and data caches: OPB cache and CacheLink cache. The OPB cache is the standard cache. The CacheLink cache is more advanced by using FSL (fast simplex link) technology to communicate with off-chip memory, as opposed to using the OPB, allowing other resources to use the bus. The CacheLink cache uses less BRAM (block RAM) for a given cache size at the expense of a higher LUT (lookup table) count.
- Cache size – Each cache’s size may range from 0K to 64K. We considered sizes of 0K, 4K, 16K, and 64K.
- Instruction/data location – The instruction and data segments of an application can be stored in either of two memory locations: on-chip BRAM, or off-chip memory.

Figure 2: Diagram of the Microblaze system being configured.



When both instructions and data are stored off-chip, the size of the BRAM is reduced to 2K.

- Compiler optimizations – An application can be compiled using one of four optimization levels: none, O1, O2, or O3.

We define a *base Microblaze core* as a core configured with all instantiatable datapath units turned off, all caches off, instruction and data segments on chip, and no compiler optimization.

4.2 Equivalent LUTs

The term “size” must be qualified in the context of FPGAs. FPGAs contain different hardware resources, including lookup tables (LUTs), multipliers, and block RAM. However, having a single number for size is desirable during design optimization, so that two designs may be directly compared. The most straightforward method of creating a single size number is to combine the various hardware resource numbers of a particular design – number of LUTs used, number of multipliers used, bytes of block RAM used – using a weighted sum function. We use the equations shown in Figure 1, where weights for multipliers and block RAMs are determined from the total number or size of those items in a maximally configured soft-

Figure 1: Equations for calculating *Equivalent LUT* value of a configured MB System.

$$LUT_{Equivalent} = LUT_{Regular} + LUT_{Equivalent(Mult)} + LUT_{Equivalent(BRAM)}$$

$$LUT_{Equivalent(Mult)} = \#Mult_{Used} / \#Mult_{full MB} * LUT_{full MB}$$

$$LUT_{Equivalent(BRAM)} = sizeBRAM_{Used} / sizeBRAM_{full MB} * LUT_{full MB}$$

core. For configured Microblazes, we found this equivalent LUT approach to correlate almost perfectly (0.995) to the “equivalent gate” concept defined by Xilinx for general circuits. From this point forward, we use the term LUTs to refer to equivalent LUTs, unless otherwise stated.

4.3 Size Estimation

Although our DoE usage will keep small the number of soft-core configurations that must be synthesized, we still found it necessary to use estimation to reduce the number of such synthesis runs during our own experiments for this paper. Synthesizing one soft-core configuration can range from 10 minutes to over one hour, depending on the parameter values of the configuration. Running 20 or 30 configurations could thus require 1-2 days. In an attempt to reduce that time during our experiments, in which we were tuning not just one application but rather ten, we investigated the accuracy of adding incremental sizes (versus the base) of each parameter’s hardware. In other words, if a multiplier adds X LUTs to the base core’s size, and a divider adds Y LUTs, then we sought to determine the accuracy of estimating the size of a core having both a multiplier and divider by simply adding X + Y to the base core’s size, rather than actually synthesizing a base core with a multiplier and divider.

Figure 6: Obtained application speedups when tuning a Microblaze soft-core using the Single Factor and the Design of Experiment approaches to build the impact ordered tree, with a size constraint of 10,000 LUTs.

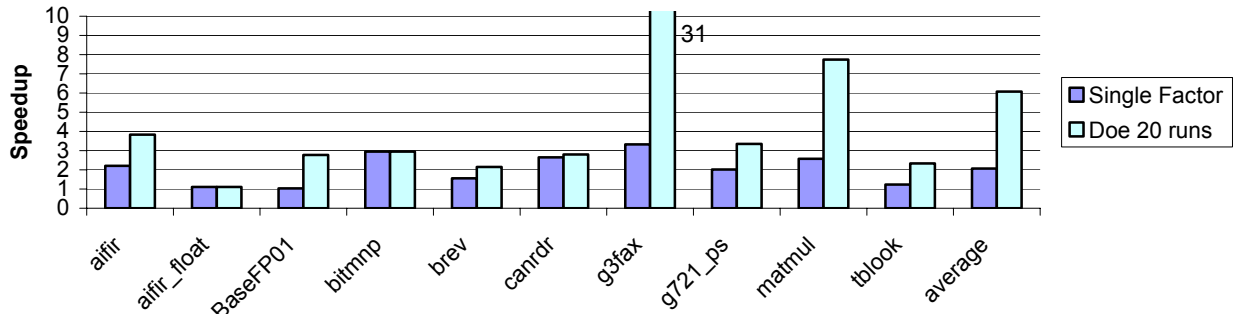
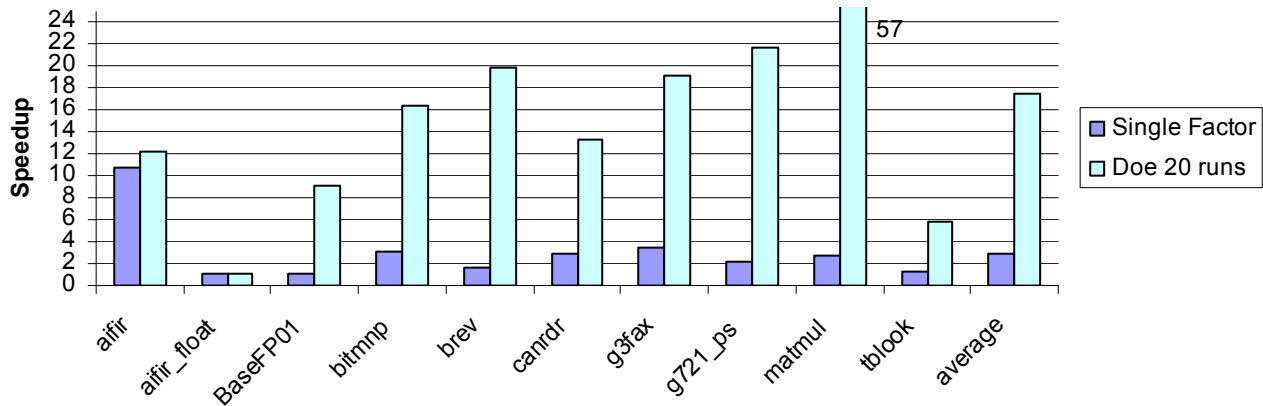


Figure 7: Obtained application speedups when tuning a Microblaze soft-core using the Single Factor and the Design of Experiment approaches to build the impact ordered tree, with a size constraint of 30,000 LUTs.



The DoE approach required 37 runs per application (20 for the DoE part, and 17 for the impact-ordered tree part), while the single-factor analysis required 34 runs – roughly the same number for both approaches. Thus, the difference in upcoming results can be attributed mostly to quality of the search process, rather than to more runs by one approach or the other.

5. Results

5.1 DoE for Determining Impact Ordering

We ran the DoE and single-factor approach on 10 EEMBC benchmarks [4], for three size-constraint scenarios. Figure 6 shows the results comparing the Single Factor and the DoE approaches with a small size constraint 10,000 LUTs, while Figure 7 shows data for a size constraint of 30,000 LUTs. (A base Microblaze uses 7,376 LUTs). Results are shown in terms of the speedup obtained by running the application on a tuned core compared to running the application on a base core. The results show that, while both approaches yield speedups, the DoE approach vastly outperforms the single-factor analysis approach. Figure 6 shows on average that the Single Factor has an average speedup of 2 times, while the DoE approach yields a 6 times speedup; Figure 7’s speedups are 3 versus 17. Thus, DoE improves on a single-factor approach by $6/2=3x$, and by $17/3\approx 6x$.

We also obtained results with a 50,000 LUT size constraint. Because that size was large enough to support most optional core

components, the approaches achieved the same speedups on all benchmarks.

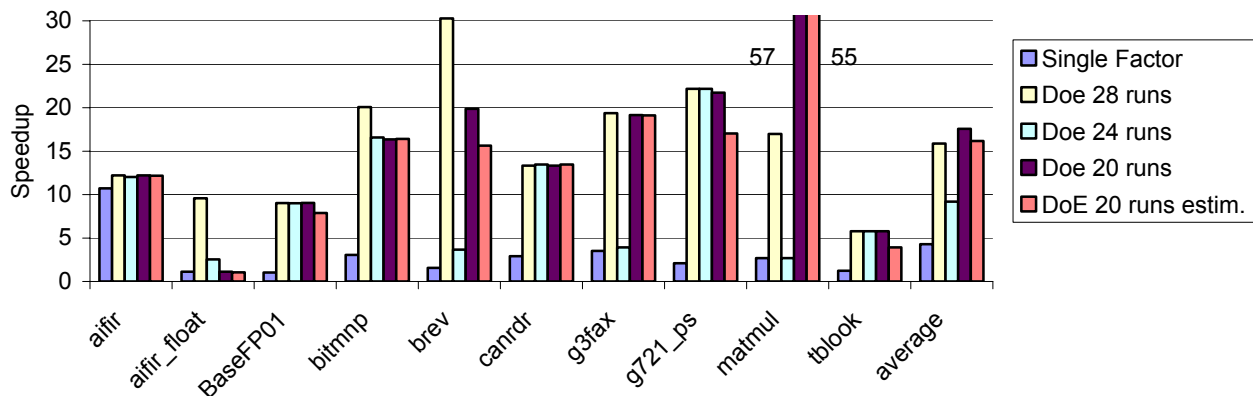
The single-factor approach’s success can depend strongly on selection of a good base system, and we were concerned that we might have chosen a poor base, thus hurting the approach’s results. We therefore modified the base case to place instruction and data segments off-chip (the most important factor in most application), and repeated the entire set of experiments. We found no significant difference in results compared to the results in Figure 6 and Figure 7. This subject does bring up the point that the DoE paradigm does not require definition of a base system, instead considering multiple factors turned on (or at the high level) simultaneously.

We also generated DoE results using 24 and 28 run options provided by the DOE Pro XL tool. The data appears in Figure 8. We saw only mild improvements compared to the 20 run option, and in some cases slight worsening, due to the approaches being heuristic in nature

5.2 Using DoE’s Built-In Tuning Approach

The previous section used DoE to determine the impact of each factor, and then used a tree-based heuristic to actually tune the parameter values. Interestingly, DoE tools often contain additional algorithms that further analyze the experimental data, and seek to predict the factor values that would predict the best output. These algorithms are based on sophisticated statistical prediction techniques whose details are beyond the scope of this

Figure 8: Speedups obtained by the single-factor approach; by 20, 24, and 28-run DoE approaches for creating an impact-ordered tree, and 20-run DoE approach using DoE's built-in statistical prediction of the best configuration.



paper². The advantage of using the DoE tool's predicted best values would be to eliminate the time required to run the impact-ordered tree tuning heuristic, reducing the total experiments from 37 down to just the 20 DoE-generated runs.

Figure 8 shows the speedups obtained using DoE's predicted best configuration ("estim."), compared to our earlier approach of using DoE only to create the impact-ordered tree followed by a heuristic search of that tree, for 20-run DoE. The figure shows that the predicted configuration method is nearly as good as the impact-ordered tree heuristic. The single factor approach required 34 runs, the DoE 20-run approach followed by the tree heuristic required 37 runs, while the DoE with prediction approach required only 20 runs (with the prediction algorithm running in negligible time). This finding represents a nice additional benefit of using DoE for tuning parameterized softwares, namely that of about 40-45% faster tuning runtimes.

6. Conclusions

Using the well-established Design of Experiments paradigm for tuning a microprocessor soft-core to an application can result in tune cores having 6x-17x speedup compared to a base core. Those speedups are 3x-6x better than obtained by a previous non-DoE-based core tuning approach. The key benefit of DoE is the multi-factor analysis, proven to yield near-maximum information from a given small number of experimental runs after decades of intense development by various scientific communities. As modern technologies result in more parameterized components, applying DoE techniques to find the best parameter settings may become increasingly important in more areas of system-level design automation.

References

- [1] Arm <http://www.arm.com>
- [2] Altera Corp. Nios II Processors. <http://www.altera.com/products/ip/processors/nios2/ni2-index.html>, 2005.
- [3] DOE Pro XL http://sigmazone.com/doepro_faqs.htm
- [4] EEMBC. <http://www.eembc.org/>, 2005.

- [5] Givargis, T., F. Vahid. Platune: A Tuning Framework for System-on-a-Chip Platforms. IEEE Transactions on Computer Aided Design, Vol. 21, No. 11, Nov. 2002, pp. 1317-1327.
- [6] Givargis, T., F. Vahid. Parameterized System Design. IEEE/ACM International Workshop on Hardware/Software Codesign (CODES), 2000.
- [7] Kumar, R., Dean Tullsen, and Norman Jouppi. "Core Architecture Optimization for Heterogeneous Chip Multiprocessors". International Conference on Parallel Architectures and Compilation Techniques, PACT, Seattle, April 2006
- [8] Kumar, R., Dean Tullsen, Partha Ranganathan, Norman Jouppi, Keith Farkas. "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance". In 31st International Symposium on Computer Architecture, ISCA-31, June 2004.
- [9] McLean, R., V. Anderson, Applied Factorial and Fractional Designs. Marcel Dekker, Inc. New York, New York, 1984.
- [10] Moyer, B., Tune *Multicore Hardware for Software*. Xcell Journal, Issue 58. 2006, pp 55-57
- [11] Mohanty, S., Prasanna, V. K., Neema, S., and Davis, J. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. Joint Conference on Languages, Compilers and Tools For Embedded Systems, 2002
- [12] Petersen, R., *Design and Analysis of Experiments*. Merceel Dekker Inc. New York, New York, 1985.
- [13] Sekar, K., Kanishka Lahiri, Sujit Dey. Dynamic Platform Management for Configurable Platform-Based System-on-Chips. Intl. Conf. on Computer-Aided Design (ICCAD), 2003.
- [14] Sheldon, D., R. Kumar, R. Lysecky, F. Vahid, D. Tullsen. Application-Specific Customization of Paramaterized FPGA Soft-Core Processors. Intl. Conf. on Computer-Aided Design (ICCAD), 2006
- [15] Tensilica, Inc. The XPRES Compiler: Triple-Threat Solution to Code Performance Challenges. http://www.tensilica.com/pdf/XPRES-Triple-Threat_Solution.pdf, 2005.
- [16] Xilinx, Inc. MicroBlaze Soft Processor Core. http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze, 2005.
- [17] Yiannacouras, P., J. G. Steffan, and J. Rose. Application-Specific Customization of Soft Processor Microarchitecture. FPGA 2006.
- [18] Yiannacouras, P., Jonathan Rose, J. Gregory Steffan. The Microarchitecture of FPGA-based soft processors International Conference on Compilers, Architecture, and Synthesis For Embedded Systems (CASES), 2005.

² And frankly, not entirely understood by this paper's authors. DOE Pro XL fully understands the techniques so it is not required that the authors have a complete understanding.