μComplexity: Estimating Processor Design Effort

Cyrus Bazeghi   Francisco J. Mesa-Martinez   Jose Renau
Dept. of Computer Engineering, University of California Santa Cruz
http://masc.soe.ucsc.edu

ABSTRACT

Microprocessor design complexity is growing rapidly. As a result, current development costs for top of the line processors are staggering, and are doubling every 4 years. As we design ever larger and more complex processors, it is becoming increasingly difficult to estimate how much time it will take to design and verify them. To compound this problem, processor design cost estimation still does not have a quantitative approach. Although designing a processor is very resource consuming, there is little work measuring, understanding, and estimating the effort required.

To address this problem, this paper introduces *μComplexity*, a methodology to measure and estimate processor design effort. *μComplexity* consists of three main parts, namely a procedure to account for the contributions of the different components in the design, accurate statistical regression of experimental measures using a nonlinear mixed-effects model, and a productivity adjustment to account for the productivities of different teams. We use *μComplexity* to evaluate a series of design effort estimators on several processor designs. Our analysis shows that the number of lines of HDL code, the sum of the fan-ins of the logic cones in the design, and a linear combination of the two metrics are good design effort estimators. On the other hand, power, area, frequency, number of flip-flops, and number of standard cells are poor estimators of design effort. We also show that productivity adjustments are necessary to produce accurate estimations.

1 Introduction

While the ability to fabricate ever larger and denser circuits is still increasing as predicted by Moore's Law, the semiconductor industry is facing several serious challenges. One of them is the cost of new processor development. Current development costs for top of the line processors are staggering, and are doubling every 4 years [12]. Another challenge is the growing difficulty to correctly design and verify the circuits — which has been called the Design and Verification Gaps [2]. As a result, according to the ITRS 2002 update [2], "the increasing level of risk that design cost and design quality present to the continuation of the semiconductor industry" is of serious concern.

The main cause of these two problems is the growing complexity of designs. Designing a top of the line commodity processor requires large teams of engineers working for several years designing, implementing, and verifying the circuits. Ironically for such a resource-intensive endeavor, there is little systematic work (at least in the public domain) on measuring, understanding, and estimating the effort required by each step in this process. If effort estimates were available early in the design process, they would help identify the critical paths in the whole design process, thus allowing resources to be more effectively allocated and procured.

In this paper, we take a first step toward developing a methodology for measuring and estimating processor design effort. Our focus is the effort in person-months required to implement and verify the RTL description of the design. We propose a novel methodology called *μComplexity*, which consists of three parts: a procedure to account for the contributions of the different components of the design, accurate statistical regression of experimental measures using a nonlinear mixed-effects model, and a productivity adjustment to account for the productivities of different design teams.

This paper also applies the *μComplexity* methodology to several freely-available processor designs, and evaluates the accuracy of a series of design effort estimators. These estimators are based on design synthesis and software metrics. Our results show that the number of lines of Hardware Design Language (HDL) code, the sum of the fan-ins of logic structures, and a linear combination of these two metrics are simple, good estimators of design effort. On the other hand, dynamic or static power, logic or storage area, frequency, number of flip-flops and, somewhat surprisingly, the number of standard cells, are inaccurate estimators of design effort. The evaluation quantifies the accuracy of each of the estimators.

The rest of the paper is organized as follows. Section 2 describes the *μComplexity* methodology; Section 3 describes the statistical techniques that allow us to calibrate and evaluate the *μComplexity* regression model; Section 4 describes the setup for our evaluation; Section 5 evaluates each of the designs; Section 6 covers related work; and Section 7 presents conclusions and future work.

2 *μComplexity*: Estimating Design Effort

μComplexity is a methodology to measure and estimate the design effort in person-months required for a processor. *μComplexity* comprises three components. The first one is an accounting procedure whereby the design is partitioned into disjoint modules that can be measured individually. A quantification for the entire processor is obtained by aggregating all the module measurements. The second component is the application of statistical regression to these design measures to obtain an unscaled estimate of the design effort. The final component is the multiplication of this unscaled effort by a

This work was supported in part by the National Science Foundation under grant EIA-0072102 and DARPA under grant NBCH30390004.

productivity factor to obtain the estimation of the design effort for a given design team.

In the following subsections, we first review a typical design flow and define the design effort that we are trying to estimate. Next, we discuss the three-component *μComplexity* methodology in detail. Finally, we examine some concerns about the methodology.

2.1 Design Effort Defined

The processor development timeline can be broken down into several overlapping stages as shown in Figure 1. Note that the duration of the different stages is not drawn to scale. The figure also shows an approximation of the size of the engineering team working on the project during each stage.

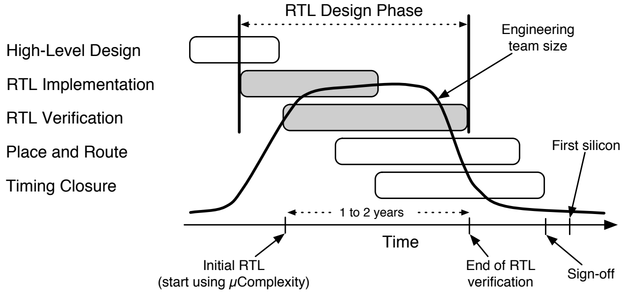

Figure 1: Processor development timeline with the size of the engineering team. Note that the timeline is not drawn to scale.

In the *High-Level Design* stage, architects perform functional simulation and power estimation of multiple candidate designs. Based on that, they select one microarchitecture and produce a complete functional and interface description of each of its components. Examples of such components are the branch predictor, load-store queue, or floating-point unit. These components are then assigned to engineering teams for implementation.

In the *RTL Implementation* stage, engineering teams implement their assigned components in an HDL such as VHDL or Verilog. They continue refining the description until they reach an RTL-level implementation, which can be automatically translated to a gate-level netlist. Functional bugs are fixed as the verification teams discover them. Synthesis is performed to ensure that the timing, area, and power goals are being met.

In the *RTL Verification* stage, engineers create test cases to verify the functionality of individual components and of the whole chip. They perform cycle-accurate simulations and compare the results with the expected values. At this point, the verification team is only concerned with the functional correctness of the design — whether it produces correct answers in a logic-level simulation. Circuit-level verification, in which electrical and timing parameters are verified, comes later. RTL verification is complete when the number of outstanding bugs reaches zero and stays there for a pre-agreed amount of time.

In the *Place and Route* stage, the synthesized netlist is physically placed within the chip-defined core area based on timing constraints. During the placement phase, gates are resized and some additional logical optimization may be performed. After the initial placement, the routing phase will occur and, if needed, minor placement changes will be made. Once the design is successfully placed and routed, clock tree synthesis happens, whereby the clocks in the design have their buffer trees placed and routed.

Finally, in the *Timing Closure* stage, engineers perform timing analysis of the gate-level implementation to determine the maximum clock speed of the design and to identify critical paths. A redesign may be required which could involve RTL or placement–and–route changes. A refine–test–refine loop exists between the Place-and-Route and Timing Closure stages.

As shown in Figure 1, the focus of this paper is the period that includes both the RTL Implementation and the RTL Verification stages. We define *Design Effort* as the number of person-months spent implementing the description of the processor in a hardware design language such as VHDL or Verilog, refining it to an RTL description, and verifying the latter for functional correctness. We exclude any additional time required to revise the design later, during the Timing Closure process. While the period considered excludes some design time, we believe that it includes the bulk of it.

2.2 Accounting Procedure

To estimate the overall design effort, estimates of the effort for each component are obtained, and then added into a compounded index. However, components may be instantiated several times through the design. Some components may also be *parameterized*, and different-sized instances could be generated. Parameters could be the width of the input or output buses, queue depth, or pipeline depth. To address these cases, we use the following two rules.

Account for a single instance of each component. When a design reuses a component (e.g., an ALU), we only count the design effort of one instance of it. The rationale is that, in accordance with the principles of modular design, the effort required to design and verify the component is a one-time cost. Once the component is designed and verified, it can be re-used elsewhere with negligible effort.

Minimize the value of component parameters. To estimate the design effort of a parameterized component, we set each parameter to the minimal value that does not result in a degenerate case. We refer to this minimization of parameters as *scaling*. The rationale is that, while different parameter values can drastically change the size of the component instance (in terms of chip area or number of gates), it is not much harder to write parameterized code than it is to write code for the smallest nontrivial instance.

More formally, consider a VHDL description where the parameterized component is implemented with `GENERATE` loops. We select for each parameter the smallest value that does not cause any loops or conditional statements in the RTL description to be optimized away by traditional program analysis techniques such as constant propagation and dead code elimination. The process for Verilog is more difficult to formalize because Verilog did not have an equivalent of the `GENERATE` construct until Verilog-2001 was introduced.

However, the determination of what constitutes the minimal non-degenerate parameterization is conceptually the same.

2.3 Design Effort Estimator

There are multiple metrics that may be related to design effort. Examples include the number of logic gates or the number of HDL lines in the design description. Consequently, for each component in the design (subject to the constraints of Section 2.2), we measure these metrics. Then, we select a single metric or a set of metrics (e.g., the number of gates and the number of HDL lines) and use statistical regression [13] to find how well they correlate with the person-months design effort reported by the processor designers. For each set of metrics m_1, m_2, $\ldots m_n$, we find the best values for the coefficients w_1, w_2, $\ldots w_n$ in Equation 1. The result is a *Design Effort Estimator* (*eff*):

$$\text{eff} = \frac{1}{\rho} \times \sum_{k=1}^{n} (w_k \times m_k) \qquad (1)$$

The regression model used is described in Section 3. In the equation, ρ is the productivity factor for the design team. It allows the same set of coefficients w_k to be used in different projects. The rationale for ρ is discussed next.

2.4 Productivity Adjustment

In software development projects, it is well known that different development teams have different productivities. For example, it has been shown that the productivity difference between teams can be up to an order of magnitude [7]. We believe that a similar effect occurs between hardware design teams. The productivity differences may be due to multiple factors, including the average experience of the designers in the team and the tools used. In our analysis, ρ captures this effect.

2.5 Issues

Ideally, we would like to use design effort estimators as soon as possible in the processor design timeline. The earlier the estimations can be made, the more useful they are likely to be. Early estimation presents two challenges: how to adjust the coefficients w_i shown in Equation 1 and how to ensure that the values of the early metrics remain relevant (and valid) at later stages of the design. We address the first challenge in Section 3.1.1. We address the second one by using metrics whose value changes little from initial stages of the design until completion of the RTL implementation and verification. Specifically, the metrics analyzed in this paper can be measured once a module has been designed and before it starts to be verified. This corresponds to the point shown with an arrow in Figure 1, which is often 1 to 2 years before completing the RTL verification. The values of the metrics remain largely unchanged until the end of RTL verification. The exception is if the verification finds substantial bugs that require a major re-design.

One potential objection to our accounting procedure is that counting each component only once regardless of its number of instances may not be appropriate. For example, at a very low level, we could consider that the entire processor is made out of logic gates, and that there are only a dozen or so types of gates. The analysis would clearly be inaccurate. However, at the high level of the functional components that we are discussing, the count-only-one heuristic is appropriate. Anyway, any given component is likely to have fewer than ten instances. At this level, scaling the effort estimate linearly with the number of instances does not seem appropriate.

In our discussion of parameter scaling in section 2.2, we argued that writing code for a parameterized component is no more difficult than writing code for the smallest nontrivial instance of it. In practice, however, the parameter values chosen for a given instance may affect the number of test vectors required for verification and, therefore, the verification time. For example, model checking and automatic theorem-proving tools may require more time to run with larger parameter values, since the size of the state space may be larger. However, this issue could be addressed, at least conceptually, by allocating more computational resources to the verification budget — not more engineer-hours.

The parameter scaling rule has another undesirable consequence. Specifically, varying the value of certain parameters may have implications on the difficulty of timing closure and, therefore, on the number of RTL redesign iterations. An example is the degree of associativity of a time-critical structure: higher associativity may make it hard to perform timing closure and may induce several redesigns. This issue suggests the need for future design effort estimators that are aware of back-end physical design and timing concerns.

Finally, our analysis has implicitly assumed that each component in the design is implemented from scratch. In practice, components are sometimes reused from older designs, often with little modifications. Integrating a reused component incurs some design effort, even if it requires no modification at all. The software engineering literature has discussed effort estimation for reused components [3]. We regard the study of reuse in hardware as a subject for future work.

3 Regression Model

As indicated in Section 2.3, given a set of metrics m_1, m_2, $\ldots m_n$, the goal of the regression procedure is to find the w_1, $w_2, \ldots w_n$ values for Equation 1 that provide the best fit for the person-months design effort reported by the designers. Each component in the design for which we know the design effort (e.g., fetch unit or load-store queue), is a data point consisting of the reported design effort and the measured metrics. The more data points we have, the more precise the determination of w_k will be.

The data points for this paper come from several small projects implemented by unrelated design teams at different times. Consequently, in addition to the usual statistical variation across data points, there is variation across teams. In statistical terms, this forces us to introduce a per-project *random effect* (represented by the productivity ρ). Therefore, we use a *nonlinear mixed-effects* model [19], which is able to deal with both *fixed* and *random* effects better than more conventional linear methods [6, 13].

In the following, we describe the mixed-effects model that we use and then consider what would happen if we attempted to fit a simpler model without productivity adjustments.

3.1 A Nonlinear Mixed-Effects Model

When we use Equation 1 with data from multiple projects, we have one data point for each component j designed in project i. The estimated design effort eff_{ij} is given by Equation 2. Note that for each component j from project i, we have a set of n metrics m_{ijk}. There is a productivity factor ρ_i specific to each project. However, the coefficients w_k are assumed invariant across all data points. In reality, of course, the fit is not perfect and the actual (reported by designers) design efforts Eff_{ij} are different from the estimated ones eff_{ij} (Equation 3). The difference is accommodated by the ϵ_{ij} error term, which we assume is multiplicative.

$$\text{eff}_{ij} = \frac{1}{\rho_i} \times \sum_{k=1}^{n} (w_k \times m_{ijk}) \qquad (2)$$

$$\text{Eff}_{ij} = \text{eff}_{ij} \times \epsilon_{ij} \qquad (3)$$

To fit the mixed-effects model and determine the w_k, we need to treat ρ and ϵ as independent random variables. As such, we must provide a probability distribution for each. From software engineering, we know that productivity is determined by the product of a collection of variables (e.g., team cohesiveness, tool quality or process maturity) [3]. Since the sum of a large number of random variables is distributed normally, the product of a number of random variables is distributed *lognormally* — a distribution where the logarithm of the variable is normally distributed [5]. Similarly, software engineering studies tell us that the multiplicative error ϵ is also lognormally distributed [20]. Consequently, we use a lognormal distribution for both ρ and ϵ.

The lognormal distribution is described by two parameters: μ and σ. They represent, respectively, the mean and standard deviation of the *log* of the variable. For the ρ and ϵ distributions, we choose to set $\mu = 0$, and then let the fitting procedure determine the standard deviations σ_ρ and σ_ϵ. The result of setting $\mu = 0$ in both cases is that the median of the distributions is 1. Intuitively, this means that half of the projects will have $\rho > 1$ and half will have $\rho < 1$. Similarly, half of the estimations will have $\epsilon > 1$ and half will have $\epsilon < 1$. Figure 2 shows a lognormal distribution with $\mu = 0$, showing the difference between mean, median, and mode.

Our choice also means that the resulting estimated effort eff that we obtain is the *median* design effort. To determine the estimated mean design effort $\overline{\text{eff}}$ rather than the estimated median design effort, we would apply Equation 4.

$$\overline{\text{eff}} = \text{eff} \times e^{(\sigma_\epsilon^2 + \sigma_\rho^2)/2} \qquad (4)$$

In Section 5, we use σ_ϵ as a measure of goodness of fit. Consequently, it is important to understand what different values of σ_ϵ tell us about the quality of the estimate. Specifically, we say that σ_ϵ determines a *confidence interval* for the estimated effort. The x% confidence interval for eff_{ij} is defined to be the range of efforts (el_{ij}, eh_{ij}) such that $P(el_{ij} < \text{Eff}_{ij} < eh_{ij}) = x/100$. For example, the 90% confidence interval gives us two values a and b such that there is a 90% chance that the actual effort is between a and b. Figure 3 plots the 68% and 90% confidence intervals for a range of σ_ϵ. To compute the confidence interval for a given σ_ϵ and eff_{ij}, find the value y_h corresponding to the top of the interval and the y_l corresponding to the bottom of the interval. The confidence interval is then $(y_l \times \text{eff}_{ij}, y_h \times \text{eff}_{ij})$. For example, if $\sigma_\epsilon = 0.45$ then $y_h \approx 2.1$ and $y_l \approx 0.5$. Therefore, the 90% confidence interval for Eff_{ij} is $(0.5 \times \text{eff}_{ij}, 2.1 \times \text{eff}_{ij})$.

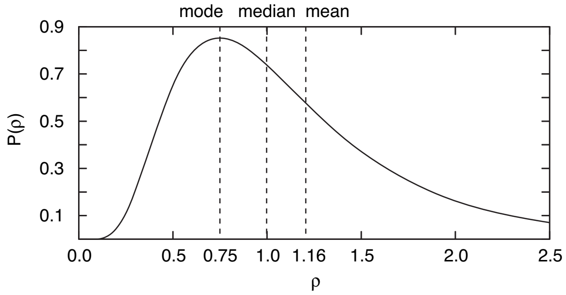

Figure 2: Example of a lognormal distribution with $\mu = 0$.

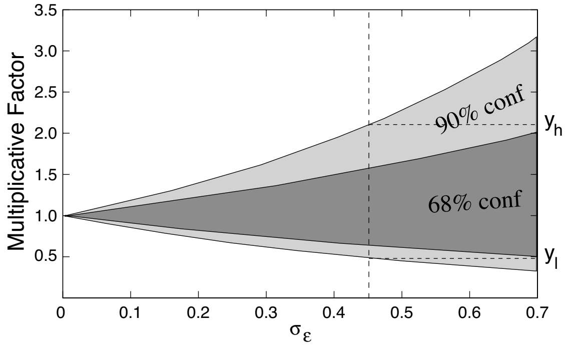

Figure 3: 68% and 90% confidence intervals corresponding to $0 \le \sigma_\epsilon \le 0.7$. The figure demonstates finding the multiplicative factors y_h and y_l for the 90% confidence interval corresponding to $\sigma_\epsilon = 0.45$.

We perform model fitting computation using the `NLMIXED` procedure from *SAS* [19], although we could also use the `nlme` package from *R* [22]. Appendix A shows a listing of the programs that generate estimates for w_k, ρ_i, and σ_ϵ.

3.1.1 How to Use the Model

μ*Complexity* is a methodology for measurement and early estimation of design effort. It attempts to discover what metrics in a design are correlated with the amount of implementation effort required and, to some extent, use them as early estimators of how much effort is required in a design. While we are only beginning to explore some of these issues, here are some guidelines on how to use the methodology in its estimation role.

In applying μ*Complexity*, the basic principle is to use the best possible estimates for w_k and ρ at any time. Ideally, this means maintaining a continuously updated database of component measurements and of reported design efforts, and periodically re-fitting the model to obtain more up-to-date estimates for ρ and, to a lesser extent, w_k.

Initial estimates of w_k can come from data from recent projects. While we assume that the w_k change slowly with time, they obviously do change as new EDA tools or design procedures are introduced. We therefore recommend periodically re-calibrating the model with recent data — including data on some recently-completed components in the current design. Using a large number of data points lends precision to the estimate, while using recent data lends accuracy. Obviously, there is a tradeoff here, although we do not have enough data to make specific recommendations.

An initial estimate of ρ may also be obtained using data from a very recent project or by extrapolating the trend of how ρ changed across projects in the past. Given the potential volatility of ρ, an alternative approach is to assume $\rho = 1$ and use the model to make *relative* effort estimations only. In this case, we can say that a component with an estimated design effort of x is likely to take half as many person-months as one with estimated design effort $2x$. These relative estimates may be useful when allocating engineers to verification teams; they may also allow an early determination of which components are likely to delay project completion.

In either case, as some components in the current project are completely verified, we can re-calibrate the model and obtain successively better estimates of the current ρ. Such ρ can be used to estimate the design effort for the remaining components of the design.

3.2 A Model Without Productivity Adjustments

Eliminating productivity adjustments by setting $\rho_i = 1$ for all i simplifies the statistical model. Instead of using the nonlinear mixed-effects model described in Section 3.1 to fit the weights, we can use a simpler multiple regression technique. Unfortunately, as we show in Section 5, the model without productivity factors fits the data poorly. We present it only for comparison with the recommended nonlinear mixed-effects model of Section 3.1.

A model without productivity adjustments may be acceptable for industrial practitioners with a very large single project, perhaps representing thousands of person-months of effort. In this case, they can set $\rho = 1$, since there is only one project and therefore no need to account for productivity differences across projects.

4 Evaluation Setup

4.1 Designs Evaluated

To evaluate *μComplexity*, we use three processors and two Register Alias Table (RAT) designs. The processors are the in-order Leon3 [11] core and the out-of-order PUMA [8] and IVM [23] cores. The characteristics of these processors are shown in Table 1.

Leon3 [11] is a single issue, in-order Sparc V8 processor designed in VHDL by Gaisler Research. The VHDL code has been released under GPL. Leon3 was originally designed under contract for the European Space Agency, and has been used in many research and commercial applications. It has a 7-stage pipeline with hardware multiply, divide, and MAC units. It has separately configurable instruction and data caches, and implements the Sparc reference MMU.

Characteristic	Leon3	PUMA	IVM
ISA	Sparc V8	PPC subset	Alpha subset
Execution	In-order	Out-of-order	Out-of-order
Pipeline stages	7	9	7
FE, IS width	1, 1	2, 2	8, 4
DI, RE width	1, 1	4, 2	4, 8
Branch predictor	None	Gshare	Tournament
Caches	Blocking	Non-block	Not modeled
Multiproc. support	Yes	No	No
HDL Language	VHDL-89	Verilog-95	Verilog-95

Table 1: Characteristics of the processor designs used in our evaluation. In the table, FE, IS, DI, RE stand for Fetch, Issue, Dispatch, and Retire, respectively.

PUMA [8] is a two issue, out-of-order processor that executes a subset of the PowerPC integer instruction set. It was designed at the University of Michigan as a high-performance processor to be fabricated in a radiation-hardened CGaAs process. PUMA is designed in the Verilog HDL.

The Illinois Verilog Model (IVM) [23] implements a subset of the Alpha 21264 microarchitecture. The cache hierarchy is not modeled. IVM was designed in Verilog at the University of Illinois for fault-tolerance research. IVM was written in a mixture of behavioral and synthesizable formats, and some modifications were made for our analysis.

In addition to these three processors, we also use two RAT designs written in Verilog-2001. Both designs can rename up to 4 instructions per cycle. One of them is a standard design, while the other is enhanced to support sliding register windows [14]. Both designs can run at frequencies over 600MHz in a 180nm technology.

4.2 Reported Design Effort

For the designs analyzed, we interviewed the designers and obtained the rough number of person-months required to design the different components of the project. The data is shown in Table 2. Leon3 was designed by J. Gaisler from Gaisler Research. PUMA was designed by J. Sivagnaname and eight other students from the University of Michigan. IVM was designed by N. Wang and three other students from the University of Illinois in about one year. The RAT was designed by J. Renau from the University of California Santa Cruz.

Reported Design Effort (Person-Months)	
Leon3:	IVM:
Pipeline: 24	Fetch: 10
Cache: 6	Decode: 2
MMU: 6	Rename: 4
Mem Control: 6	Issue: 4
	Execute: 3
PUMA:	Memory: 10
Fetch: 3	Retire: 5
Decode: 4	
ROB: 4	RAT:
Execute: 12	Standard: 0.3
Memory: 1	Sliding: 0.5

Table 2: Reported design effort.

4.3 Metrics

We identify a set of metrics that we feel may be related to design effort. For each component of each design, we gather these metrics shown in Table 3. Also shown in the table is the tool used to obtain the measurement.

Metric	Description	Tool
FanInLC	Total number of inputs of all logic cones	Synplify Pro
LoC	Number of lines in the HDL code	–
Stmts	Number of statements in the HDL code	–
Nets	Number of nets	Design Comp
Cells	Number of standard cells	Design Comp
$Area_L$	Logic area in μm^2	Design Comp
$Area_S$	Storage area in μm^2	Design Comp
$Power_D$	Dynamic power in mW	Design Comp
$Power_S$	Static power in μW	Design Comp
Freq	Frequency for 90nm Stratix-II EP2S90 FPGA	Synplify Pro
FFs	Number of flip-flops	Synplify Pro

Table 3: Metrics gathered for each component.

The concept of FanInLC is as follows. Given a primary output (i.e., a signal that reaches a pipeline latch), we identify the set of logic gates that produces it starting from the preceding pipeline latch (i.e., its logic cone), and count all the primary inputs to the cone (i.e., signals directly coming from the preceding latch). We then repeat the process for all the primary outputs in the design, accumulating the counts. It is expected that components with higher FanInLC tend to require a higher design effort.

The other entries in Table 3 are conceptually straightforward. They include the number of lines or statements in the HDL code, the number of nets, cell gates, or flip-flops in the component, the area devoted to logic or storage, the static or dynamic power dissipated, and the maximum frequency at which the component can cycle using a 90nm Altera Stratix-II EP2S90 FPGA device [4].

To obtain all these metrics, we synthesize the designs. Synthesis is the process of converting an HDL functional description into a gate-level netlist in the case of a standard-cell ASIC, or into a logic-block mapping in the case of an FPGA. For ASIC synthesis, we use Synopsys Design Compiler 2004.12 [15] to synthesize to a 180nm standard cell library; for FPGA synthesis, we use Synplify Pro 8.1 from Synplicity [16] and targeted an Altera Stratix II device.

From the output of Synplify Pro, we obtain three of the metrics in Table 3, namely the frequency, number of flip-flops, and an estimate of the FanInLC. The latter is estimated as follows. Synplify Pro reports the number of Look-Up Tables (LUTs) that use a given number of inputs. We approximate FanInLC by summing all the inputs used in all the LUTs. This may be slightly inaccurate because when the number of inputs to a cone exceeds the eight inputs available on a single LUT, Synplify Pro cascades LUTs to form the cone. In practice, Synplify Pro rarely had to cascade LUTs in our designs.

From the output of Design Compiler, we obtain most of the other metrics in Table 3, including power and area estimates.

5 Evaluation

Our evaluation examines how accurately each of the software and synthesis metrics of Table 3 correlate with design effort. We also examine a few combinations of such metrics. In our analysis, we compare against some of the design effort estimators currently being used. Specifically, Sematech [12] and the SIA Roadmap [2] which use the number of cells and the number of bits or transistors, respectively, to estimate effort. Synthesis tools report most of the metrics listed in Table 3, which are often used to make effort estimations. Finally, in software development, some of the most popular software complexity accounting methods, such as COCOMO [3] and Function Points [1] use the number of lines of code as a proxy for effort.

As indicated in Section 3.1, to assess the accuracy of an estimator, we report the standard deviation of its error (σ_ϵ). Lower values of σ_ϵ are better, and zero is the minimum possible value. Given a σ_ϵ, we can compute the interval for, say, 90% confidence for the true value. For the lognormal distribution used, the mapping between σ_ϵ and the 90% confidence interval is shown in Figure 4. We will use this chart to compare the accuracy of different estimators.

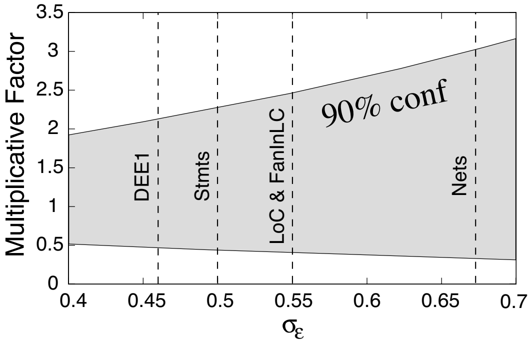

Figure 4: Mapping between the standard deviation of the error (σ_ϵ) and the 90% confidence interval for the lognormal error distribution used. This plot is organized as Figure 3.

We do not report an accuracy measure similar to the multiple-R^2 measure often used in linear models. The reason is that it is not recommended for the nonlinear mixed-effects model that we use. In practice, the statistics community often uses two popular goodness-of-fit measures to compare different non-linear mixed models, namely Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC) [13]. These are rigorous methods, but they are less intuitive than the σ_ϵ approach that we use here. Consequently, we only give the AIC and BIC metrics in a few cases.

In the following, we first measure the accuracy of the different design effort estimators using our model. Then, we repeat the process without the productivity adjustment or without the *μComplexity* accounting procedure.

5.1 Accuracy of Design Effort Estimators

Table 4 shows the accuracy of various design effort estimators. First, Column 2 lists the reported design effort in person-months for each component of each design — this data repeats

Module Name	Effort (Person × Months)	DEE1	Stmts	LoC	FanInLC	Nets	Freq (MHz)	$Area_L$ (μm^2)	$Power_D$ (mW)	$Power_S$ (μW)	$Area_S$ (μm^2)	Cells	FFs
Leon3-Pipeline	24	12.8	2070	2814	10502	4299	56	50199	80	409	68411	3586	1062
Leon3-Cache	6	7.3	1172	1092	6325	1980	94	37456	57	332	12556	3	210
Leon3-MMU	6	4.4	721	1943	3149	1130	84	60136	23	287	112765	246	699
Leon3-MemCtrl	6	5.4	938	1421	2692	853	138	7394	5	2	11938	704	275
PUMA-Fetch	3	2.2	586	1490	5192	1292	68	147096	226	3513	555168	1809	1786
PUMA-Decode	4	6.2	1998	3416	4724	5662	65	78076	11	526	47604	5189	464
PUMA-ROB	4	2.2	503	913	6965	9840	41	82527	733	816	1022	9709	922
PUMA-Execute	12	12.6	3762	9613	18260	10681	49	92473	44	1370	119746	10867	1725
PUMA-Memory	1	3.3	976	2251	5034	1089	60	43418	80	602	115841	4337	1549
IVM-Fetch	10	8	1432	4972	15726	4914	71	212663	8	2	135074	1859	1661
IVM-Decode	2	1.7	391	963	1044	504	104	2022	2	6	73	2	0
IVM-Rename	4	2.7	566	2519	3307	1134	159	70146	1	1	26740	121	510
IVM-Issue	4	3.6	624	2704	8063	4603	60	90388	2	1	68667	3414	2729
IVM-Execute	3	5.4	961	4083	11045	4476	91	619561	5	5	154655	940	0
IVM-Memory	10	11.6	2240	5308	19021	23247	54	267753	73	2	625952	12050	2510
IVM-Retire	5	5	1021	2278	6635	3357	71	36100	2	1	50375	1923	924
RAT-Standard	0.6	0.7	64	250	3889	2905	137	34254	4	275	17603	2596	288
RAT-Sliding	1	1	78	334	5586	4936	119	52210	10	459	60713	4507	612
σ_ϵ	–	0.46	0.50	0.55	0.55	0.67	0.94	1.23	1.34	1.44	2.07	2.09	2.14
σ_ϵ ($\rho_i = 1$)	–	0.53	0.60	0.69	0.82	1.08	1.12	1.35	1.82	3.21	2.07	2.55	2.18

Table 4: Accuracy of various design effort estimators.

the data shown in Table 2. Then, each of remaining columns shows data for one design effort estimator. Most of the estimators are simply the individual software or synthesis metrics listed in Table 3. The only exception is the DEE1 estimator, which is the linear combination of two metrics — we will analyze DEE1 in Section 5.1.1. For a given estimator, the column shows its value for each component of each design and, in the penultimate row, its σ_ϵ. We do not report the productivities ($\rho_{\mathtt{Leon3}}$, $\rho_{\mathtt{PUMA}}$, $\rho_{\mathtt{IVM}}$, and $\rho_{\mathtt{RAT}}$) in the table – they can be calculated using the programs in Appendix A. Finally, we ignore the last row until Section 5.2.

From the table, we see that there are a group of estimators that have a relative high accuracy (i.e., low σ_ϵ). They include Stmts, LoC, FanInLC, and Nets. For example, Stmts and FanInLC have σ_ϵ equal to 0.50 and 0.55, respectively, which, according to Figure 4, correspond to a 90% confidence interval of (0.44,2.28) and (0.40,2.47), respectively. Really, within the margin of error of our study, any one of Stmts, LoC, or FanInLC has the same accuracy.

The other estimators, namely Freq, $Power_D$, $Area_L$, $Power_S$, $Area_S$, Cells, and FFs, have lower accuracy. For example, $Area_L$ has σ_ϵ equal to 1.23, which corresponds to a 90% confidence interval of (0.13,7.56). None of these metrics is a reasonable estimator.

Freq has a 90% confidence interval as large as (0.21,4.69). While increasing processor frequency requires additional design effort, other metrics like Nets or FanInLC have higher correlation with design effort. The reason is that, to increase frequency, it is necessary to add extra pipeline stages or more complex logic. This increased effort is better measured by Nets and FanInLC.

Perhaps unsurprisingly, $Area_S$ and FFs are not well correlated with design effort. Their 90% confidence intervals are (0.03,30.11) and (0.03,33.78), respectively. The reason is that storage structures such as RAM banks are relatively simple to design. Similarly, $Area_L$ and Cells are not well correlated because simple to implement structures can occupy a lot of area and have large numbers of logic cells. Moreover, neither dynamic nor static power is well correlated with design effort as their confidence intervals are (0.11,9.06) and (0.09,10.68) respectively. Larger designs will probably require more power, but are not necessarily more complicated to design.

Overall, our data shows that any one of Stmts, LoC or FanInLC is a good single-metric estimator of design effort. Interestingly, this shows some similarity between hardware and software design efforts. On the other hand, it appears that the hardware estimators used elsewhere such as Cells and transistors used by the SIA Roadmap and Sematech are not so effective. Most of the other synthesis tools metrics such as area, power and frequency are not well correlated with design effort either.

5.1.1 Design Effort Estimator 1 (DEE1)

We have also analyzed the accuracy of estimators generated with the linear combination of groups of two metrics from Table 3. As usual, we use Equation 1 from Section 2.3. We find that two-metric combinations that include Stmts, LoC, FanInLC, and Nets tend to have slightly more accuracy than those with a single metric. The ones that are the most accurate are Stmts plus Nets, and Stmts plus FanInLC. They have the same accuracy, but we prefer the Stmts plus FanInLC estimator because, individually, the metrics are more accurate. We call the resulting estimator Design Effort Estimator 1 (DEE1).

As shown in Table 4, DEE1 has the lowest σ_ϵ, namely 0.46. This corresponds to a 90% confidence interval of (0.47,2.13). The slightly higher accuracy of DEE1 comes from the fact that its two component metrics measure slightly different underlying factors in the design. The corresponding accuracy values using AIC and BIC are 34.8 and 38.4, respectively. As a comparison, the AIC and BIC values of Stmts are 37.0 and 39.7, respectively. For AIC and BIC, lower values mean a better fit.

To see the correlation between DEE1 and the reported design effort better, Figure 5 shows a scatter plot of DEE1 estimations versus reported design effort. The Figure has one data point per component and design. From the figure, we see that most of the DEE1 estimations are very close to the reported design effort. The exception is the data point for the Leon3 pipeline, where the DEE1 estimation is 12.8 months, and the reported effort is 24 months. In practice, most of the estimators in Table 4 underestimate the effort for the Leon3 pipeline. The reason is that this pipeline is more sophisticated than the other components and designs. Indeed, while IVM and PUMA only execute a subset of Alpha and PowerPC, respectively, Leon3 is a full SPARC V8 compliant processor. In addition, Leon3 is highly configurable, for example the user can select different processor and cache parameters.

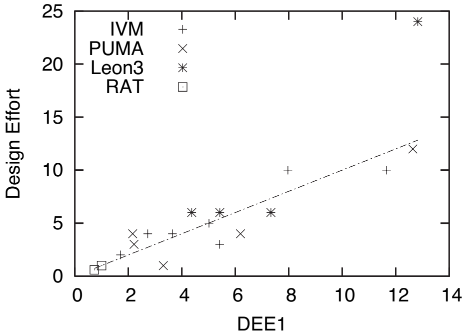

Figure 5: Scatter plot of DEE1 estimations versus reported design effort.

Generating estimators with the linear combination of groups of more than two metrics from Table 3 decreases the AIC and BIC values. Therefore, the small correlation improvement is not recommended unless more data samples are considered.

5.2 Accuracy without the Productivity Adjustment

The last row of Table 4 shows the σ_ϵ values that would be obtained if no productivity factor was used – in other words, if ρ_i was 1 for the Leon3, PUMA, IVM, and RAT teams. This approach was mentioned in Section 3.2.

From the values of σ_ϵ, we can see that practically all the estimators lose a significant amount of accuracy. For example, the σ_ϵ for Stmts and FanInLC becomes 0.60 and 0.82, respectively, which correspond to 90% confidence intervals of (0.37,2.68) and (0.26,3.85), respectively. Similarly, DEE1 expands its 90% confidence interval to (0.41,2.39).

The loss of accuracy for LoC and Stmts is due to several factors. Specifically, while Leon3 uses VHDL, the other designs use Verilog. Moreover, while RAT uses the more compact Verilog-2001, PUMA and IVM use the more verbose Verilog-95. Additionally, different coding styles add much noise to any correlation without productivity adjustment. To compound the problem, it is known from software projects that productivity across teams can vary up to 10 times [7].

The FanInLC and Nets estimators lose accuracy because each processor was designed under a different set of constraints and a different set of tools. For example, since Leon3 was designed for an area-constrained environment (FPGAs), a substantial effort was needed to reduce area and interconnections. On the other hand, PUMA's target was a high frequency CGaAs process. All these effects again add noise to any correlation.

Overall, we conclude that, to have good estimation accuracy, productivity adjustments are required.

5.3 Accuracy without the Accounting Procedure

One aspect of the *µComplexity* methodology is the special procedure followed to account for multiple instantiations of a given component and parameterized components. This procedure is described in Section 2.2. In this section, we compute the σ_ϵ values that would be obtained without such a procedure — namely, no special provisions for these components.

Figure 6 shows the σ_ϵ for our estimators when the measurements are gathered with or without our accounting procedure. Note that the σ_ϵ with the accounting procedure are equal to those listed in Table 4.

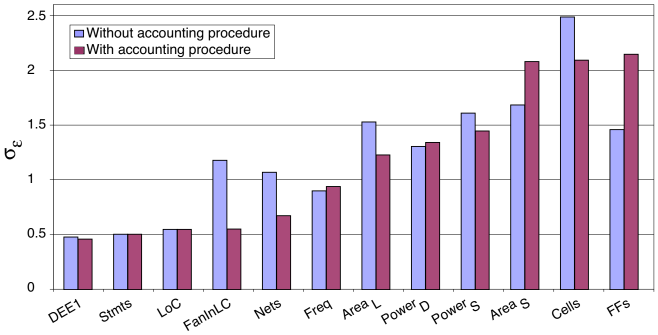

Figure 6: Accuracy of the design effort estimators without or with the *µComplexity* accounting procedure of Section 2.2.

Figure 6 shows that, without our accounting procedure, the accuracy of the estimators is often very low. Specifically, the two estimators generated with a single synthesis metric that had acceptable accuracy before (FanInLC and Nets), now have poor accuracy. FanInLC and Nets have a σ_ϵ equal to 1.18 and 1.07, respectively, which correspond to a 90% confidence interval of (0.14,6.97) and (0.17,5.81), respectively. The other estimators generated with a single synthesis metric (Freq, $Power_D$, $Area_L$, $Power_S$, $Area_S$, Cells, and FFs) have such high σ_ϵ to start with that any variation is uninteresting. For example, the 90% confidence interval for Freq without our accounting procedure is (0.23,4.32).

The accuracy of the estimators without synthesis metrics (Stmts and LoC) does not change because the absence of the accounting procedure does not affect them. Moreover, the accuracy of the DEE1 estimator changes little because one of its metrics is Stmts, and the regression automatically compensates for the inaccuracy coming from the other metric (FanInLC).

While not shown in Figure 6, it can be shown that the main contributor to the difference between the environments with and without the accounting procedure is the IVM design. The IVM design models a 4-issue Alpha superscalar core. The IVM design that we measure has many cases of multiple instantiations of the same component, and of parameterized components. The narrower PUMA superscalar and the 4-way RAT designs are simpler designs; they have fewer cases of such components. Finally, the streamlined, single-issue Leon3 processor has practically no such types of components.

Overall, from our measurements, we conclude that the design effort estimators based on synthesis metrics that we consider should use our accounting procedure. Otherwise, the accuracy is low.

6 Related Work

The work most related to ours is done by Numetrics, a company specializing in enterprise software and services product development [21]. They propose a "complexity unit" to measure the level of project difficulty and to quantify the development team's output. Patent 6,823,294 describes a method to estimate design effort. If we apply the method to our data, the result is considerably less accuracy than DEE1. After discussions with Numetrics, they informed us that the patent represented preliminary work, and that their current models are more advanced. Unfortunately, little detail is available on their current models because it is considered a technological advantage for their company.

The authors are unaware of any other work on the topic of estimating the number of engineering person-hours that will be needed for a processor design. Some related research has proposed productivity optimization by focusing on "design process optimization". For example, METRICS [9] is a system that seeks changes to the EDA tools and focuses primarily on back-end or place-and-route performance, not on design effort.

Recently, some research has focused on reducing the number of RTL redesigns during the timing closure process. To streamline timing closure, new methods have been developed to predict logic criticality [17] and wire congestion [18] early in the RTL design phase. With these predictors, logic designers can focus their attention on the critical logic during the initial implementation, reducing the number of redesign cycles.

Fornaciary *et al.* [10] propose a methodology to predict the final size of a VHDL project on the basis of a high-level description. With this, they seek some indication of development effort by estimating the number of lines of code from starting specifications. While their method was shown to be accurate in predicting lines of code, it did not address the design effort, such as the number of engineering person-months required for the project.

7 Conclusions & Future Work

As processors get ever more complicated, it is becoming very important to have simple, and yet accurate, early estimators of design effort. One of the most important developments in the field of computer architecture has been the introduction of quantitative approaches, in which different aspects of a processor system are measured, to further understand design implications. Most quantitative studies have focused in the measurement of speedup. To address this problem, this paper has made two main contributions.

First, this paper has introduced the novel *μComplexity* methodology to measure and estimate processor design effort. *μComplexity* consists of three main parts, namely a procedure to account for the contributions of the different components, accurate statistical regression using a nonlinear mixed-effects model, and a productivity adjustment to account for the productivities of different teams.

The second contribution has been applying *μComplexity* to four designs and evaluating a series of estimators based on synthesis and software metrics. The evaluation uncovered a few simple, good design effort estimators, namely the number of lines of HDL code (or HDL statements) and the sum of the fan-ins of all the logic cones. A slightly more accurate estimator is DEE1, which is the linear combination of HDL statements and fan-ins of all the logic cones. We recommend this estimator, but using estimators that combine a larger number of metrics may make sense for a practitioner that has access to more data.

The evaluation also revealed several inaccurate estimators of design effort. These include dynamic or static power, logic or storage area, frequency, number of flip-flops and, somewhat surprisingly, the number of standard cells. The number of cells and transistors are two popular design effort estimators used by Sematech and the SIA roadmap. Finally, the evaluation showed that both the productivity adjustment and the *μComplexity* accounting procedure are necessary to produce accurate estimators.

We hope that this paper helps open up the largely-unexplored area of computer architecture that deals with metrics for design effort. There is much work to be done. A methodology for estimating design effort from early measurements would be very useful. It may help identify the critical paths in the development process, thus allowing resources to be more effectively allocated and procured. This paper has studied estimators based on measurements of early RTL code. As our next step, we are now focusing on estimators that can be obtained even earlier. Such early estimators would allow design considerations to be made early, when the costs are low, to better reach the goals and requirements of the project. Such estimators must necessarily be derived from a higher-level description of the design. This paper has provided intuition that might be used to generate such estimators.

This paper has focused only on some aspects of the design process, namely the engineering hours for RTL implementation and verification. In practice, there are many aspects of the development of a processor that we did not address. We are extending our approach to cover a larger fraction of the activities required to develop a processor. We are collaborating with industry design teams to address those areas.

Acknowledgments

Special thanks to Josep Torrellas and Brian Greskamp for their invaluable support to enhance the statistical model and help with the experimental data. We thank Jayakumaran Sivagnaname, Jiri Gaisler, and Nicholas Wang for providing design effort numbers for the PUMA, Leon3, and IVM, respectively. Thanks also go to Donglai Dai (SGI), Pradip Bose

(IBM), Victor Zyuban (IBM), Prabhakar Kudva (IBM), and Emmett Kilgariff (NVIDIA) for their insights on this project. Finally, Maria Muyot from the Illinois Statistics Office helped with the statistical analysis.

A SAS and R Code

This appendix provides the SAS [19] and R [22] code to fit the nonlinear mixed-effects model explained in Section 3.1. Like most statistical analysis software, SAS and R expect the error to be additive and normally distributed. They also require the random effects to be normally distributed. Recall that our model in Equations 2 and 3 has a multiplicative lognormal error and also a lognormal distribution for the random effect ρ. Simply taking the logarithm of both sides of the equation gives us the requisite additive normal error and normal random effect as follows. Note the change of variables from $-ln(\rho_i)$ to log_ρ_i and $ln(\text{Eff}_{ij})$ to log_actual$_{ij}$. $\mathbf{N}(\mu, \sigma^2)$ represents a normally distributed random variable.

$$\begin{aligned} ln(\text{Eff}_{ij}) &= -ln(\rho_i) + ln\sum_{k=1}^{n}(w_k \times m_{ijk}) + \mathbf{N}(0, \sigma_\epsilon^2) \\ \text{log\_actual}_{ij} &= \text{log\_}\rho_i + ln\sum_{k=1}^{n}(w_k \times m_{ijk}) + \mathbf{N}(0, \sigma_\epsilon^2) \end{aligned}$$

After the transformation, entering the model into SAS or R is straightforward. The input data set contains columns labeled “team”, “log_actual”, “FanInLC”, and “Stmts”. The “team” field contains the name of the team that designed the component. The “log_actual” column contains the logarithm of the designer's reported effort for the component. “Stmts” and “FanInLC” are the column labels for the metrics we want to use in the estimator. R lets you choose the fitting method, we choose “ML” for consistency between SAS and R. ML maximizes the log-likelihood.

```
*** SAS Code
PROC NLMIXED;
  log_est = log_rho + log(w1*Stmts + w2*FanInLC);
  MODEL log_actual ˜ NORMAL(log_est, sigma_epsilonˆ2);
  RANDOM log_rho ˜ NORMAL(0, sigma_rhoˆ2)
         SUBJECT=team OUT=prods;
RUN;

# R Code
nlme(model=log_actual ˜
  (log_rho) + log(w1*Stmts + w2*FanInLC)
  ,random = log_rho ˜ 1 | team
  ,fixed  = list(w1 ˜ 1, w2 ˜ 1)
  ,start = c(0.0001, 0.00001)
  ,data=(traw)
  ,method="ML")
```

The SAS code stores the estimated productivity factors in a new data set called “prods”. You may need to remove the string “OUT=prods” to see the rest of the output from the analysis (including the estimated weights and error) on the console.

REFERENCES

[1] A.J. Albrecht. Measuring Application Development Productivity. In *SHARE/GUIDE IBM Applications Development Symposium*, Oct 1979.

[2] Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS), 2002.

[3] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[4] Altera Corporation. Stratix II Device Family Literature, 2005. http://www.altera.com/products/devices/stratix2/.

[5] E.L. Crow and K. Shimizu. *Lognormal Distributions: Theory and Application*. Dekker, 1988.

[6] M. Davidian and M.D. Giltinan. *Nonlinear Models for Repeated Measurement Data*. Chapman & Hall, 1995.

[7] T. DeMarco and T. Lister. *Peopleware Productive Projects and Teams*. Dorset House Publishing, 1999.

[8] A.J. Drake, T.D. Basso, S.M. Gold, K.L. Kraver, P.N. Parakh, C.R. Gauthier, P. S. Stetson, and R.B. Brown. CGaAs PowerPC FXU. In *International Conference on Design Automation*, pages 730–735, New York, NY, USA, 2000. ACM Press.

[9] S. Fenstermaker, D. George, A. B. Kahng, St. Mantik, and B. Thielges. METRICS: A System Architecture for Design Process Optimization. In *International Conference on Design Automation*, pages 705–710, Jun 2000.

[10] W. Fornaciari, F. Salice, and D.P. Scarpazza. Early Estimation of the Size of VHDL Projects. In *International Conference on Hardware/Software Codesign and System Synthesis*, pages 207–212, Oct 2003.

[11] J. Gaisler, S. Habine, and E. Catovic. GRLIB IP Library User's Manual, 2005. http://www.gaisler.com.

[12] R. Goodall, D. Fandel, A. Allan, P. Landler, and H. R. Huff. Long Term Productivity Mechanisms of the Semiconductor Industry. www.sematech.org, 2002.

[13] J.P. Hoffmann. *Generalized Linear Models*. Pearson, 2004.

[14] SPARC International Inc. The SPARC Architecture Manual Version 8, 1992. http://www.sparc.com.

[15] Synopsys Inc. Design Compiler Product Information, 2005. http://www.synopsys.com.

[16] Synplicity Inc. SynplifyPro Product Information, 2005. http://www.synplicity.com.

[17] P. Kudva, B. Curran, S.K. Karandikar, M. Mayo, S. Carey, and S.S. Sapatnekar. Early Performance Prediction. In *Workshop on Complexity-Effective Design*, Jun 2005.

[18] P. Kudva, A. Sullivan, and W. Dougherty. Metrics for Structural Logic Synthesis. In *International Conference on Computer-Aided Design*, pages 551–556, Nov 2002.

[19] R.C. Littell, G.A. Milliken, W.W. Stroup, and R.D. Wolfinger. *SAS System for Mixed Models*. SAS Publishing, 1996.

[20] T. Little. Value Creation and Capture: A Model of the Software Development Process. *IEEE Software*, 21(3):48–53, 2004.

[21] Numetrics Management Systems. Key Performance Indicators of IC Development Capability-A Framework. Technical report, Numetrics Management Systems, Inc., 2005. http://www.numetrics.com.

[22] The R Development Core Team. *The R Reference Manual - Base Package*. Network Theory Limited, 2005.

[23] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *International Conference on Dependable Systems and Networks*. IEEE Computer Society, Jun 2004.