A Systematic Approach to Automated Construction of Power Emulation Models

Benjamin A. Bjørnseth, Asbjørn Djupdal, Lasse Natvig
Department of Computer and Information Science
Norwegian University of Science and Technology
{benjambj,djupdal,lasse}@idi.ntnu.no

***Abstract*—Efficient estimation of power consumption is vital when designing large digital systems. The technique called *power emulation* can speed up estimation by implementing power models alongside a design on an FPGA. Current state-of-the-art power emulation methods construct models using various custom techniques, but there is no study on how the existing methods relate to each other nor how their differences impact the final quality of the model. We propose a methodology which describes the breadth of current approaches to automated construction of power emulation models. We also evaluate the current methods, finding that there is significant variation in accuracy and complexity. In 32.8 % of all tests, the average accuracy of the least complex method is better than that of the most advanced method at less than 0.3 % the hardware overhead. This result fuels the hope that further innovation may yield models with high accuracy at low implementation cost. Our software frameworks and experimental data are made available to promote continued work on the field.**

I. Introduction

The proliferation of power as a first class design constraint in computer architecture research makes it increasingly important to evaluate the energy efficiency of architectural innovations. A precise evaluation of power consumption may require the development of HDL implementations of the proposed hardware designs, which enables the use of ASIC design tools such as Synopsys PrimeTime [1]. These tools yield highly accurate power estimates, but the run-time of such tools may prevent evaluations using real-world workloads.

FPGAs have long been a vessel for high-speed prototyping of ASIC hardware designs. Previous work has proposed a technique named *power emulation*, where regression models relating RTL activity to energy or power consumption are created and implemented on an FPGA alongside the HDL being prototyped [2]. The slow ASIC design tools are thus only used once, to gather activity and power data for creating the regression models. Energy consumption can then be estimated using the FPGA at orders of magnitude higher speed.

Several different semi- or fully automated methods have been proposed to make the regression modelling step in power emulation simpler [3], [4], [5]. We will call such methods *energy model synthesis* (EMS). These methods demonstrate the viability of automated power emulation. However, no work has thus far considered what trade-offs exist in the construction of the modelling method itself; several method variants work, but how their differences impact model quality is unclear. Furthermore, no detailed comparison between these state-of-the-art methods exist.

In this article, we provide a description of the current methodology[1] for automated model creation for power emulation. The explicit methodology description simplifies systematic comparison of EMS method alternatives. Next, we evaluate the trade-offs between state-of-the-art EMS methods. In order to perform these evaluations, we extend the semi-automatic methods to be fully automated. We find that the modelling time, prediction accuracy and hardware cost of the different methods vary significantly, but that the hardware cost savings of the least complex method is disproportionately high compared to its loss of accuracy. This suggests that more accurate methods can be created at lower costs by judiciously applying the most successful aspects of the more complex methods as extensions to the simplest one.

The remainder of this article is organized as follows. Section II presents background material on power modelling of RTL designs in general and the more specific EMS methods. Section III describes the EMS methodology, and identifies how various methods deviate from one another. An evaluation of the state-of-the-art EMS methods is presented in Section IV, including a discussion of the impact the results may have on EMS method design practices. Finally, we conclude and present ideas for future work in Section V.

II. Background

A. RTL Power Models and Power Emulation

Power models of RTL designs have been used to extend various software simulators with power estimation capabilities; examples include regular RTL simulation [6], SystemC models [7] and HLS binaries [8]. The different methods showcase a broad set of statistical techniques for crafting models from a sample set of RTL activity and corresponding power values. Bogliolo *et al.* use regression trees, where different regression models are used based on the values of splitting variables [6]. The splitting variables are design signals selected based on the variance in power due to the signals. The regression models are computed using the method of least-squares. Bansal *et al.* apply significance testing to determine relationships between signals and power consumption [7]. They also use regression trees, but select splitting variables based on which signals introduce the fewest cases. Regression models are created using symbolic regression, based on the selected signals and a pre-defined set of operators. Lee *et al.* create different models

[1]Note the distinction between method and methodology: a method is a description of steps taken to reach a goal, whereas a methodology is the set of methods applicable within a domain.

978-3-9815370-6-2/DATE16/

for different states of a control pipeline [8]. For each state, they use decision trees to select a subset of signals as predictors, calculate four models using different regression modelling techniques, and select the best-performing one.

The idea of implementing the power models in hardware, called *power emulation*, was first proposed by Coburn *et al.* [2]. This has a calculation speed benefit, but the hardware overheads put stricter constraints on model complexity. Coburn's method is based on creating and accumulating models of all RTL components, with several optimizations which reduce the area requirements. Still, the area overheads reported were on average $3\times$. An approach for scaling power emulation to full chip-multiprocessor systems was proposed by Bhattacharjee *et al.* [9]. They use performance counters as predictors, reducing per-cycle prediction accuracy for massive area reductions. Their strategy requires a design with performance counters, and the designer knowledge to select among them. A case study of how power emulation can be applied in a prototyping platform may be found in [10].

B. Energy Model Synthesis Methods

Recent methods have focused on partial or complete automation of the modelling [3], [4], [5]. Ideally, a user would only need to input an HDL design to the method in order to synthesize a version augmented with a model of the per-cycle energy consumption.

Sunwoo *et al.* propose creating a hierarchical power model semi-automatically [3]. A subset of modules in the hardware design and a subset of inputs for each of these modules are selected manually. For each module, the method automatically creates a regression model using the method of least-squares. The sum of model predictions form the model of the top-level module. The formulae have the fixed format $\sum HD(B) \times S^k$, where HD denotes a Hamming-distance computation, B is the selected input buses, S is the selected input single-bit signals, and k is a parameter which is set by the modeller to balance modelling run-time with attainable model accuracy. To reduce overheads, the final model uses only the N terms with the highest maximum values in the training data.

Bachmann *et al.* present a semi-automatic method for creating a model for an entire hardware design [4]. The method uses single-bit signals as predictors, selected based on a manually created name pattern list. The initial list of signals is filtered based on inter-signal correlation and power-signal auto-correlation values. The filters remove heavily correlated signals, and only keep signals correlating with the power consumption. The final regression model is the sum of the remaining signals, with coefficients fit based on a non-negative least-squares solution. In a follow-up work, the authors determine that the exclusive use of single-bit signals is inadequate for accurately predicting the power consumption in data-dependent situations [11]. This work extends the method with the option of including manually selected bus variables

The most recent work, conducted by Yang *et al.* [5], features a completely automated modelling method. They include one predictor variable for each flip-flop in the design, where each predictor is the toggle-activity of the corresponding flip-flop. Since there may be a significant number of flip-flops in a design, they use a rank-k-limited singular-value decomposition technique to reduce the modelling time and calculate an approximated least-squares solution for this formula.

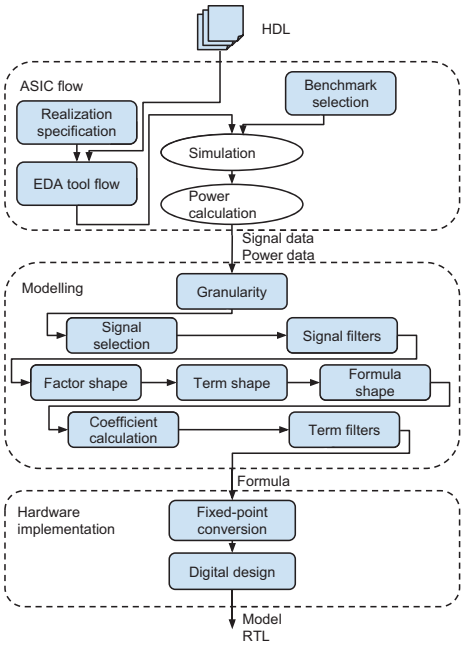

Fig. 1. The steps in the EMS methodology. The dashed lines demarcate methodology categories, and the blue boxes indicate steps in which there are significant opportunities for method variation. Ellipses are invariants.

III. The Energy Model Synthesis Methodology

As is clear from Section II-B, there is significant variation in the design of existing EMS methods. To facilitate systematic method design experimentation, we developed a unifying methodology description illustrated in Figure 1. The methodology is split into three categories, which are further comprised of a series of steps. The internals of each step are what separate one method from the other. The method design decisions addressed in the different parts of the methodology are as follows:

a) ASIC Flow: The ASIC flow category steps produce training data for modelling. Apart from using simulation and ASIC power calculation tools, EMS methods must include:

Realization specification: ASIC power calculation requires realization details such as the target cell library and operating conditions. Although often a manual responsibility, complete automation would require EMS method involvement.

EDA tool flow: The EDA tool flow synthesizes the HDL design into an ASIC realization. Methods can differ in whether or not the synthesis manages physical layout.

Benchmark selection: The benchmark selection impacts signal activity quality because models are optimized for the activity in the training data. Typical alternatives are writing benchmarks customized to the design, or picking applications from a benchmark suite.

b) Modelling: The modelling category steps form the actual regression modelling. This is the primary focus of current EMS methods. The following list describes the questions

TABLE I. IMPLEMENTATION VARIATIONS FOR METHODOLOGY STEPS IN THE MODELLING CATEGORY. (LEGEND: SUNWOO, BACHMANN YANG)

Granularity	*Signal Selection*	*Signal Filters*	*Factor Shape*	*Term Shape*	*Formula Shape*	*Coefficient Calculation*	*Term Filters*
Top-level module only	Single-bit signals	Name filters	Raw signals	Single factors	Sum of terms	Least-squares	Maximum term value
Hierarchical All RTL components [2]	Module inputs	Auto-correlation with power	Hamming distance	$HD(B) \cdot S^k$	Regression trees [6], [7], [8]	Non-negative least squares	
	Flip-flops Performance counters [9]	Inter-signal correlation	Toggle Count of zeros [7]	Bit-wise `and` [7]		Rank-truncated SVD-based least squares	
		Manual selection Significance tests [7]	Count of ones [7]	Bit-wise `or` [7] Multiplication [7]		Bayes Ridge [8] Gradient boosting [8]	

methods address when implementing a given step. For each step, some alternatives previously used are presented in Table I.

Granularity: Should the top-level model be hierarchical, and if so at what granularity should the design be modelled?

Signal selection: What kind of HDL design signals should model predictors be based on?

Signal filters: Can the usefulness of signals as predictors be determined, in order to filter out less useful signals to reduce the modelling complexity?

Factor shape: How should the model predictors be derived from the selected signals?

Term shape: How should the predictors be combined to form terms of the model?

Formula shape: Is the final formula a regression tree, or a single sum of terms?

Coefficient calculation: What method is used to calculate coefficients for the model terms?

Term filters: Should the model complexity be reduced by selecting the most useful terms in the final formula?

c) Hardware Implementation: The hardware implementation category steps generate an implementation of the mathematical model, balancing accuracy and hardware overhead. The steps are as follows:

Fixed-point conversion: The floating-point coefficients from the regression modelling must be converted to fixed-point by selecting an appropriate scaling factor and a bit width.

Digital design: The final step creates the hardware based on some predefined digital design. Simple models can be trivially implemented; complex models may require tailor-made designs for sufficient efficiency, as in the work by Coburn *et al.* [2].

IV. EVALUATING THE STATE OF THE ART

Having described the methodology, we will next consider how the current state-of-the-art embodiments of it compare to each other. Such a comparison should yield a better understanding of the current methods, which is useful both for determining which option is the best in a given scenario and for research on improvements to the methods.

A. Evaluation Framework

We have created an evaluation framework which manages technical tasks in the execution and evaluation of EMS methods. This includes HDL analyses; sampled benchmark simulation; EDA tool flow scripts; data management; modelling and hardware implementation steps and evaluation utility functions in R [12]; and hardware cost calculation. The HDL analyses and sampled simulation are based on the open-source Chisel HDL [13], but other languages can be supported as the rest of the framework is independent of the chosen HDL. All the utility software and our data material is publicly available [14].

TABLE II. METHODOLOGY STEPS FIXED DURING EVALUATION.

Methodology Step	**Configuration**
HDL	Rocket chip [15] with one core.
Realization Specification	Commercial 65 nm 1.0V cell library, 500 MHz, no clock-gating.
EDA tool flow	Synopsys DC [17], front-end synthesis only.
Benchmark selection	Custom 30K-cycle program.
Fixed-point conversion	Fixed resolution of 2^{-16}.
Digital design	Sum of terms.

TABLE III. METHOD PARAMETERS USED DURING EVALUATION.

Sunwoo [3]	**Bachmann** [4]	**Yang** [5]
$k = 2$	Signal-signal correlation threshold $= 1$	$k = 100$
$N = 10$	Signal-signal correlation lag $= 5$	
	Power-signal auto-correlation threshold $= 0.40$	
	Power-signal auto-correlation lag $= 5$	

B. Evaluation Method

1) Methodology Configuration: To enable a fair comparison between the EMS methods, we use the methods to model the same HDL design: a Rocket chip [15] with one core. The power consumption of on-chip SRAM resources is subtracted from their containing modules, as these resources are better modelled with specialized SRAM tools like CACTI [10].

We also fix the steps in the ASIC flow and hardware implementation categories of the methodology. The benchmark is a custom 30K-cycle program, written to exercise various parts of the Rocket chip. We use a simple digital design in which the final result is calculated using an adder tree. The individual terms are calculated from factors using either bit-wise `and` if a factor has bit-width 1, or multiplication otherwise. Most factors are trivially calculated; for Hamming distance calculation, we use the FPGA-specific implementation proposed by Skylarov *et al.* optimized for wide buses [16]. All the fixed settings are summarized in Table II.

The steps in the modelling category are implemented separately for each method in the way described in the original article. Each method parametrizes certain steps in a way which allows varying the trade-off between accuracy, implementation cost, and modelling time. We set the parameters to the values reported in the original articles. Not all parameter values were reported; in these cases, their values were set to the values which yielded the best results after systematic experimentation. The method parameters are listed in Table III.

2) Evaluation: The accuracy of the models is evaluated by using each model to predict power based on activity traces

TABLE IV. LIST OF BENCHMARKS USED FOR VALIDATION.

MiBench	Samples [cycles]	MachSuite	Samples [cycles]
automotive/basicmath	$5 \times 20K$	fft/strided	$5 \times 20K$
automotive/bitcount	$5 \times 20K$	fft/transpose	$6 \times 20K$
automotive/qsort	$6 \times 20K$	bfs/bulk	$10 \times 10K$
network/dijkstra	$4 \times 20K$	bfs/queue	$10 \times 10K$
office/stringsearch	$2 \times 20K$	aes/aes	$1 \times 103K$
security/sha	$5 \times 20K$	stencil/stencil3d	$5 \times 20K$

from a set of validation benchmarks, and comparing the predictions to actual values calculated by Synopsys PrimeTime. We use selected benchmarks from MachSuite [18] and MiBench [19], listed in Table IV. Signal activity is sampled at periods customized to the length of each benchmark, after skipping a benchmark-dependent initialization phase, to keep the data volume manageable.

The modelling time reported for each method is the wall-clock time spent running it, excluding the time required for evaluating its accuracy. Although the modelling is implemented in R, the majority of the time is spent on coefficient calculation which is implemented with calls to efficient libraries. The run-times should therefore be representative of what one may expect from more thoroughly optimized implementations.

The hardware cost is calculated by synthesizing the generated models using Xilinx Vivado, targeting the Xilinx Virtex-7 FPGA.

C. Fully Automating Existing Methods

As explained in Section II-B, not all the modelling steps in Bachmann's and Sunwoo's methods are fully automated. We extend the methods to be fully automated to remove subjectivity as a factor in the execution of the methods.

a) Bachmann: We remove the manually-created signal name filter. The absence of a name filter causes a large number of signals to be considered. In order to mitigate the impact on modelling run-time, we re-order the power filter and signal filter to execute in that order in contrast to the original article. This substantially reduces the run-time, since the power-signal filter has lower complexity than the inter-signal filter.

b) Sunwoo: First, we include a search for an appropriate granularity. The search proceeds by creating both cumulative and hierarchical models for each module bottom-up, keeping the one with the best results. Second, we select all module inputs as signals when possible. However, certain modules have prohibitively many inputs. We therefore use correlation with power to sort the inputs, selecting the top of the list such that the total number of terms is below a given limit. For our results, we used a term limit of 2000.

D. Results

In this section, we present our results. The accuracy is reported using the following metrics:

- **Average error:** The error in prediction of the average power consumption.
- **Average per-cycle error:** The average of the absolute error of predictions each cycle $= mean(\frac{|pred_i - actual_i|}{actual_i})$

TABLE V. AVERAGE ACCURACY, HARDWARE COST AND RUN-TIME RESULTS

Metric	Sunwoo [3]	Bachmann [4]	Yang [5]
Average error	-12.9%	-6.1%	-3.2%
Average per-cycle error	19.4%	16.2%	11.6%
NRMSE	12.3%	10.4%	8.1%
CV(RMSE)	24.0%	20.5%	16.2%
95^{th} absolute error percentile	39.9%	39.7%	25.3%
99^{th} absolute error percentile	49.7%	48.2%	38.6%
Largest underestimation	-119%	-66%	-55%
Largest overestimation	270%	89%	300%
Hardware cost, LUTs	7566 (70 %)	76 (0.7 %)	27223 (253 %)
Hardware cost, registers	1272 (12.7 %)	0	8164 (81.4 %)
Hardware cost, DSPs	4 (33.3 %)	0	0
Modelling time	17.72 hours	80 seconds	9.77 hours

TABLE VI. DISTRIBUTION OF BEST-PERFORMING METHODS.

Metric	Granularity	Sunwoo	Bachmann	Yang
Average error	Per sample	1 (1.6 %)	21 (32.8 %)	42 (65.6 %)
	Per benchmark	0	5 (41.7 %)	7 (58.3 %)
Avg. per-cycle error	Per sample	2 (3.1 %)	0	62 (96.9 %)
	Per benchmark	0	0	12 (100 %)

- **NRMSE**: The root-mean-square error of predictions each cycle, normalized by the range of observations $= \frac{RMSE}{actual_{max} - actual_{min}} = \frac{\sqrt{mean((pred_i - actual_i)^2)}}{actual_{max} - actual_{min}}$
- **CV(RMSE)**: The RMSE normalized by the mean observation $= \frac{RMSE}{mean(actual_i)} = \frac{\sqrt{mean((pred_i - actual_i)^2)}}{mean(actual_i)}$

1) Comparison of State of the Art: The overall method results are presented in Table V. The accuracies are calculated as the arithmetic mean of their respective metrics for all benchmark samples. Yang's method clearly has the best overall accuracy, with the lowest errors in all categories except the largest overestimation. However, the hardware overhead of the method is also significant, requiring 253 % of the logic resources used for the design being modelled. At the other complexity extreme, we find Bachmann's method. The implementation of the final model requires only 0.7 % of the logic resources of the design itself, thus being practically free. The modelling time is also the lowest among the alternatives. The accuracy is considerably lower than Yang's method, however, with twice the average error. Sunwoo's method does not have convincing overall performance, with the lowest accuracy, highest run-time and significant hardware cost.

Figure 2 plots the average error and per-cycle error for each benchmark. In all cases, Yang's method has the lowest average per-cycle error. For the average error, however, Bachmann's method is often on par and sometimes better. Table VI lists how often each method performs the best in terms of having either the lowest average error or average per-cycle error. Bachmann's method performs well for its low cost, having the best average prediction in one third of the benchmark samples. As is clear from Figure 2, however, the margins by which Bachmann's method outperforms Yang's is often small in comparison to the margins in favour of Yang's method. For average per-cycle errors, Yang's method has the lowest average per-cycle error in all but two benchmark samples where Sunwoo's method performs the best. Comparing only Bachmann's and Sunwoo's method reveals that Bachmann's method has a lower average per-cycle error in 85.9% of the benchmark samples; Sunwoo's method is better primarily for the *qsort_small* benchmark.

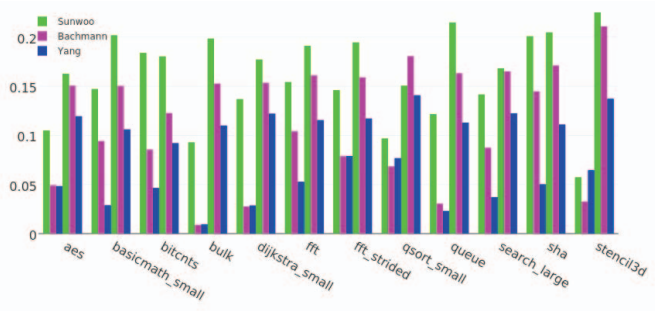

Fig. 2. The average error and average per-cycle error for each benchmark.

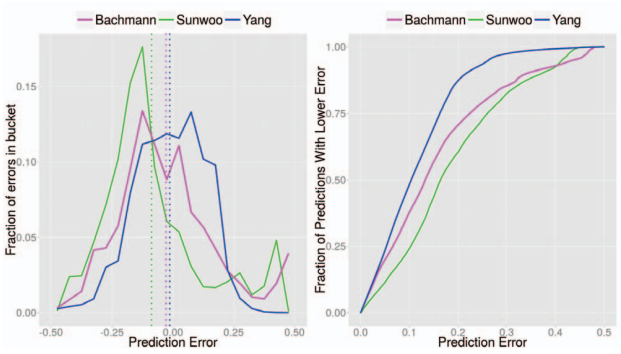

Fig. 3. Histograms and cumulative distributions of per-cycle errors for Sunwoo's, Bachmann's and Yang's method. The dotted lines indicate average prediction error. The bucket size is 0.05.

Figure 3 plots the prediction error distribution over all the benchmarks. Yang's method has stable predictive performance, but a significant tally of prediction errors occur up until the 25% error mark. This is also visible from the cumulative distribution function, which increases almost linearly with prediction error until the 90th percentile. Bachmann's method is less consistent, with a higher count of larger errors and a less steep cumulative distribution. The error frequency of Sunwoo's method has a clearer central tendency, but a larger negative mean value.

Bachmann's method shows comparable performance to Yang's method for average prediction in certain benchmarks, but is consistently outperformed on average per-cycle error. We investigate the relationship between the size of intervals over which predictions are averaged, the best-performing method and interval prediction accuracy in Figure 4. Interestingly, the fraction of intervals in which Yang's method performs the best increases until the interval reaches 100 cycles. This can be understood by looking at the interval prediction error, represented with dashed lines in the figure: for interval sizes below 100 cycles, all methods benefit from an increased interval size. Past 100 cycles the improvements in accuracy of Yang's method are lessened, which explains the relative improvement to Bachmann's method which keeps improving.

2) Impact of Toggle Activity as Factor: Both Bachmann's and Yang's methods use signals which are one bit wide to form their model. However, Bachmann uses the raw signal values as predictors whereas Yang uses their toggle activity. An interesting question is whether either of these choices

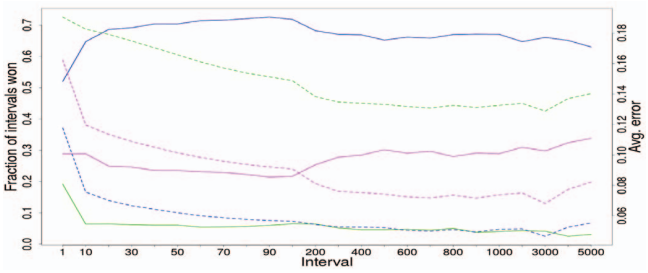

Fig. 4. The relationship between prediction interval and method accuracy. The left-hand axis and solid lines show the fraction of intervals in which a given method has the lowest average error; the right-hand axis and dashed lines show the average interval error. The colour scheme is the same as in previous figures.

TABLE VII. DATA FROM EXPERIMENTS ON ALTERNATIVE FACTORS IN YANG'S AND BACHMANN'S METHODS.

	Yang, $k = 50$		**Bachmann**	
Metric	With `xor`	Without `xor`	Without `xor`	With `xor`
Average error	−4.4%	−30.1%	−6.1%	−12.0%
Avg. per-cycle error	11.1%	31.2%	16.2%	22.6%
NRMSE	8%	20.7%	10.4%	14.9%
CV(RMSE)	16.2%	38.3%	20.5%	27.6%

is inherently more accurate than the other. We investigate this issue by modifying Bachmann's method to use toggle activity as predictors, and vice versa for Yang's method. The results from these experiments are presented in Table VII. In both cases, the modification to the original method leads to substantial accuracy degradations. Removing `xor` from Yang's method does reduce its hardware cost, but not without making it less accurate than the other methods. These results demonstrate that the choice between using toggle activity or raw signal values as factors is dependent on the implementation of other steps in the methodology.

3) Impact of Method Extensions: Our results use extensions of Sunwoo's and Bachmann's methods, as described in Section IV-C. The extensions of Bachmann's method only impact run-time, since the power filter and inter-signal filter are independent of each other. The extensions of Sunwoo's method, on the other hand, may impact the accuracy of the method as the manual interventions ensuring quality are replaced with automatic methods which have not been rigorously developed. To estimate the impact on quality caused by the removal of human expert knowledge as a method element, we experimented with slight variations in how our automated steps worked.

For the granularity selection, we tested two alternatives: one finer in which all modules are modelled, and one coarser in which models down to hierarchical level three are modelled. Table VIII presents the results. Both finer and coarser granularities worsen the accuracy of the final model , indicating that the automated step hierarchy selection is reasonable accuracy-wise. Since the difference in accuracy is substantial when using coarser granularity, the potential for reductions in hardware overhead appears limited.

For the model signal selection, we test alternative term limits of 1000 and 3000 instead of the original 2000. These results are also in Table VIII. Reducing the term limit to 1000 significantly increases both average and per-cycle error.

TABLE VIII. VARYING STEP EXTENSIONS TO SUNWOO'S METHOD.

	Granularity		Term limit		
Metric	Finer	Coarser	1000	3000	**Original**
Avg. error	−13.2%	−16.7%	−15.6%	−12.0%	−12.9%
Avg. per-cycle err.	19.7%	25.4%	23.7%	22.7%	19.4%

As only ten terms are eventually used, it is apparent that the correlation between factors and power is not perfect as a discriminator for the factors' relative eventual merits as constituents of model terms. Setting the term limit to 3000 interestingly worsens the overall model quality: the average per-cycle error rises, and graphical comparison reveals that the method quality in terms of trend prediction suffers a sharp decline. These unforeseen variations illustrate the importance and difficulty of signal selection, and indicate that automated steps must be developed with rigour before a human expert can be supplanted.

E. Discussion

In this section we highlight the key findings from our results, and discuss their implications for method design.

1. The least costly method has remarkable accuracy. Bachmann's method has on average twice the average error of Yang's method, at 0.28% the LUT cost. Being virtually free, Bachmann's method admits moderately complex additions targeting accuracy improvements in specific scenarios. A reasonable approach for future method designs is therefore to consider how much of the accuracy difference can be bridged with modest cost investments.

2. The workload impacts method prediction quality. The attainable accuracy depends on the workload. For instance, the benchmark *qsort_small* is the one where Yang's method has its highest average per-cycle error (14.1%) and Sunwoo's method has its lowest (15.1%). Method designs should include different elements which address variations in the expected workload of the design for stable accuracy.

3. Methodology steps are interdependent. As the best factor shape for Bachmann's method is different from that in Yang's method, it is apparent that other aspects of the method design influence this step. Bachmann's method can be viewed as a characterization of power states of the system, which is better described with raw signal values; Yang's method, using all flip-flops as predictors, will include signals which are not suitable as state predictors leading instead to an over-fit for the training model. It is therefore important to consider all steps of the method as a whole when designing a method, and experiment with several steps simultaneously.

4. Automation should be rigorously developed. Without a good selection of input signals, Sunwoo's method is unable to perform well. The negative central tendency in Figure 3 indicates that the method is over-fit to the training data. Our extension performs adequately, but is brittle as it depends unpredictably on the term limit parameter. This demonstrates that all steps must be rigorously developed during method design if full automation is to be accomplished.

V. CONCLUSION AND FUTURE WORK

This article presented a methodology encompassing the current approaches to automated power emulation. We described how the different state-of-the-art methods all follow the same general approach, and how the methods relate to one another. We also presented an evaluation of these methods, revealing a considerable span in prediction accuracy, modelling time and hardware overhead. The least costly method had a prediction error in average power twice that of the most accurate method, with less than 0.3% of the hardware overhead.

This work only constitutes a step towards thorough understanding of the methodology for automated power emulation model creation. There is ample opportunity for further research on steps in the methodology, or even extensions to the methodology itself. One option is to investigate whether the least costly method can be extended with elements which improve its accuracy at reasonable hardware cost. Another topic is the applicability of more powerful statistical modelling methods such as regression trees. Finally, one may target full method automation by addressing EDA tool chain dependencies and design stimuli generation. Our open source software and data material facilitate continued research on these areas [14].

REFERENCES

[1] "Synopsys PrimeTime," www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx, Synopsys, Oct. 2014.

[2] J. Coburn *et al.*, "Power emulation: a new paradigm for power estimation," in *DAC*, 2005, pp. 700–705.

[3] D. Sunwoo *et al.*, "PrEsto: An FPGA-accelerated power estimation methodology for complex systems," in *FPL*, 2010, pp. 310–317.

[4] C. Bachmann *et al.*, "Automated power characterization for run-time power emulation of SoC designs," in *DSD*, 2010, pp. 587–594.

[5] J. Yang *et al.*, "Early Stage Real-Time SoC Power Estimation Using RTL Instrumentation," in *ASP-DAC 2015*, 2015, pp. 779–784.

[6] A. Bogliolo *et al.*, "Regression-based RTL power modeling," *TODAES*, vol. 5, no. 3, pp. 337–372, 2000.

[7] N. Bansal *et al.*, "Automatic power modeling of infrastructure IP for system-on-chip power analysis," in *VLSID*, 2007, pp. 513–520.

[8] D. Lee *et al.*, "Dynamic Power and Performance Back-Annotation for Fast and Accurate Functional Hardware Simulation," in *DATE*, 2015, pp. 1126–1131.

[9] A. Bhattacharjee *et al.*, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," in *ISLPED*, 2008, pp. 335–340.

[10] B. A. Bjørnseth, "Enabling Research on Energy-Efficient System Software Using the SHMAC Infrastructure," *Master thesis*, 2015.

[11] A. Krieg *et al.*, "Accelerating early design phase differential power analysis using power emulation techniques," in *HOST*, 2011, pp. 81–86.

[12] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: http://www.r-project.org/

[13] J. Bachrach *et al.*, "Chisel: Constructing Hardware in a Scala Embedded Language," in *DAC*, 2012, pp. 1212–1221.

[14] Benjamin A. Bjørnseth, "Repository for software framework and data material," https://bitbucket.org/benjambj/energy-model-synthesis.

[15] Y. Lee, "RISC-V Rocket Chip SoC Generator in Chisel," http://riscv.org/tutorial-hpca2015/riscv-rocket-chip-generator-tutorial-hpca2015.pdf, UC Berkeley, Mar 2015.

[16] V. Skylarov and I. Skliarova, "Digital Hamming Weight and Distance Analyzers for Binary Vectors and Matrices," *Intl. Jrnl. of Innov. Comp., Inform. and Control (IJICIC)*, vol. 9, no. 12, pp. 4825–4849, 2013.

[17] "Synopsys Design Compiler," http://www.synopsys.com/tools/implementation/rtlsynthesis/pages/default.aspx, Synopsys, Nov. 2014.

[18] B. Reagenakun *et al.*, "MachSuite: Benchmarks for Accelerator Design and Customized Architectures," in *IISWC*, no. Section III, 2014.

[19] M. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *WWC-4*, 2001, pp. 3–14.