LSSVM-Based Power Macro-Modelling

ABSTRACT
We investigate the use of support vector machines (SVMs) to determine simpler and better fit power macromodels of functional units for high-level power estimation. The basic approach is first to obtain the power consumption of the module for a large number of points in the input signal space. Least-Squares SVMs are then used to compute the best model to fit these set of points. We have performed extensive experiments in order to determine the best parameters for the kernels. Moreover, we propose a fundamental modification to the basic kernel that greatly improves the accuracy of the model. It is well-known that not all inputs have the same impact on the power consumption of a module. We propose a new norm that takes into account the weight of each input for the power consumption in the computation of the kernels. We present results that confirm the excellent modelling capabilities of the proposed kernel-based methods. The macromodels obtained provide not only excellent accuracy on average ($< 2\%$ error), but more importantly, thanks to our proposed modified kernels, we were able to reduce the maximum error to values close to 20%. The LSSVM models compare favorably with state-of-the-art 4D table lookup models, having, on average, a third of the relative error.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Modeling techniques

General Terms
Algorithms, Performance, Design

Keywords
Power Estimation; Macro-Model; Support Vector Machines

1. INTRODUCTION
Electronic systems have become pervasive in our daily lives, work environments and even our social habits. Their design and verification prior to fabrication are challenging tasks due to inherent size and complexity. Rising costs of design and market pressure for fast delivery of error-free systems drive the need for reliable verification. A common approach towards easing the design process and enabling verification is to replace large and complex design blocks by smaller more abstract macromodels that accurately represent relevant characteristics of the system. The resulting macromodelled system can then be verified and design exploration can be accomplished, leading to better, more efficient designs, with lower costs and added features.

Power consumption has become one of the most important parameters in the design of VLSI circuits and accurate power estimation a requisite of any design exploration framework and verification environment. Many high-level power estimation tools have been proposed before to enable the evaluation of different architectures in an early stage of design [4]. The general approach is to use power macromodels for each functional unit. These macromodels are obtained in a precharacterization phase, stored in a library for later use and represent the power dissipation of the unit as a function of the input statistics. Much research has been devoted to this topic [5] and many commercial tools are available for this task. The main drawback of existing approaches is that the models generated present fairly high maximum error.

Kernel methods provide a powerful and unified framework for pattern discovery, motivating algorithms that can act on general types of data and look for general types of relations (*e.g.* rankings, classifications, regressions, clusters) [10]. The application areas range from neural networks and pattern recognition to machine learning and data mining. In this paper, we investigate the use of learning algorithms, in particular support vector machines (SVMs), to determine simpler and better fit power macromodels. The basic approach is first to obtain the power consumption of the module for a large number of points in the input signal space. Least-Squares SVMs are then used to compute the best model to fit these set of points. The statistics for each of the module's output signals can be computed in a similar manner, thus providing a means of propagating the switching probabilities through the circuit. We have performed extensive experiments in order to analyze the possible kernels which are the basis of the SVM formulation and to determine the best parameters for these kernels.

Moreover, we propose a fundamental modification to the basic kernel that greatly improves the accuracy of the model.

In general, kernels treat every dimension uniformly. In our case, each input to the functional module defines a dimension for the kernel (although not necessarily so after optimizations, as discussed in Section 4.5). It it well-known that not all inputs have the same impact on the power consumption of a module. We propose a modification to the basic RBF kernel that takes into account a measure of the contribution of each input for the power consumption in the computation of the kernels.

We present results that confirm the excellent modelling capabilities of the kernel-based methods. The macromodels obtained provide not only good accuracy on average (all below 2% average error and close to 1.5% on average), but, more importantly, thanks to our proposed modified kernels, we were able to reduce the maximum error to values close to 20%. The LSSVM models compare very favorably with state-of-the-art 4D table lookup models, having, on average, a third of the relative error.

The paper is organized as follows. In Section 2, we review previous work on power analysis techniques at the RT level and provide some background on kernel methods. Section 3 discusses the kernel parameters and presents the new norm we are proposing. The implementation of the power macromodelling process is described in Section 4. We present our results in Section 5 and draw conclusions and future work in Section 6.

2. RELATED WORK

2.1 Power Macro-Modeling

There has been a fair amount of work on generating models for power dissipation at higher levels of abstraction. Our method follows a bottom-up approach [4], where the model is obtained from an actual circuit implementation and which offers the best level of accuracy. These methods build their models from data points, which consist on a power value for the circuit under some input conditions. From this set of data points, different strategies exist for generating a model that not only fits these data points, but offers the best possible generalization ability.

Lookup tables have been successfully proposed [2]. N-dimensional tables are used, where each dimension represents an input parameter. Several strategies exist for reducing the number of dimensions and for interpolating among table points. Alternatively, regression can be used to compute the coefficients of an expression [1, 5]. A combination of both of these methods has also been proposed [2].

We believe the model we propose, based on LS-SVMs, is more robust than previously proposed approaches: it is generic, fully-automated, and uses an underlying methodology with properties that have been proven both theoretically and in practice in many different fields. Our results demonstrate just that.

2.2 LS-SVMs

Consider a general problem where we are given N input/output data points, $\{\mathbf{x}_k, z_k\}_{k=1}^N \in \mathbb{R}^p \times \mathbb{R}$. These data points follow an unknown function $z(\mathbf{x}) = m(\mathbf{x}) + e(\mathbf{x})$, where $m(\mathbf{x})$ is the target function we wish to estimate and $e(\mathbf{x})$ is a sampling error. Support Vector Machines (SVMs) are a method of obtaining $y(\mathbf{x})$, the estimate of $m(\mathbf{x})$, from the given data set, referred to as training set. SVMs achieve regression by nonlinearly mapping the input space into a higher dimensional feature space where a linear approximant hyperplane can be found. This is implicitly made by the use of a kernel function.

Table 1: Common kernels.

Linear:	$\Psi(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
Polynomial:	$\Psi(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + \theta)^n$
Exponential (RBF):	$\Psi(\mathbf{x}_i, \mathbf{x}_j) = \exp(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2})$
Hyperbolic Tangent	$\Psi(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\phi \mathbf{x}_i^T \mathbf{x}_j + \theta)$

A version of a SVM for regression was proposed by Vapnik el al [10]. This method is called support vector regression (SVR). The model produced by SVR only depends on a subset of the training data, because the cost function for building the model ignores any training data that are close (within a threshold ε) to the model prediction. The relevant data points form a set of support vectors and they immediately lead to a sparse representation. On the other hand, computing the model is a Quadratic Programming (QP) problem. To simplify this QP problem, Least Squares Support Vector Machines (LS-SVM) were introduced [9], reducing the problem to solving the following linear system:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \mathbf{\Omega} + C^{-1}\boldsymbol{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix} \qquad (1)$$

where $\mathbf{\Omega}$ is the kernel matrix, $\Omega_{kl} = \Psi(x_k, x_l)$, $k = 1, \ldots, N$, $l = 1, \ldots, N$, and Ψ is the kernel function. C is a weighing factor that represents the importance of training errors, allowing a tradeoff between the training error and the smoothness of the solution. The resulting LS-SVM is given by

$$y(\mathbf{x}) = \sum_{k=1}^{N} \boldsymbol{\alpha}_k \Psi(\mathbf{x}, \mathbf{x}_k) + b \qquad (2)$$

While SVMs have a built in way of selecting the most significant data points from their training set, LS-SVMs do not. Sparseness is lost due to the usage of all the data points as support vectors (a large N). This adds a new complexity to the problem. It becomes necessary to carefully choose the training points used to effectively cover the input space.

There are many kernels from which to choose from, some of which are shown in Table 1. In this work, we will focus exclusively on the RBF kernel since it has good empirical results and has a nice smooth behavior. However, as we shall see, much of what we propose extends trivially to the other kernels indicated.

3. KERNEL TUNING

The regression process is not totally automated. For a given class of problems, one must select a kernel function, choose the appropriate kernel parameters, and also choose the value of constant C. In this section, we describe first how we tuned these parameters for the case of power macromodelling, and

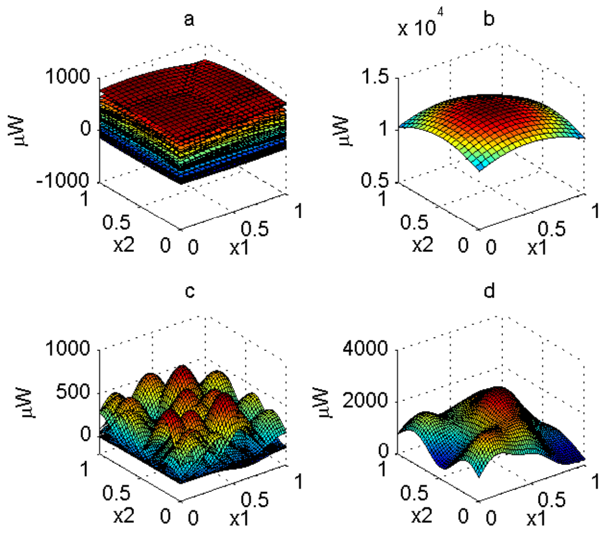

Figure 1: Power surfaces for: a) $\sigma = 1$, components; b) $\sigma = 1$, sum; c) $\sigma = 0.02$, components; d) $\sigma = 0.02$, sum.

then we present the proposed new norm to give different weighs to the inputs of functional units.

3.1 Kernel Behavior

There is some work on how to choose the 'optimal' parameters, C and σ, for the RBF kernel [6]. We begin with a simple interpretation of the RBF kernel abilities and limitations, motivating our proposed modification to the kernel.

In the exponential RBF kernel (Table 1), σ represents a width factor. If it is large, then the influence of each support vector (SV) spreads, smoothing the solution. If σ is small, each SV has a small influence over the space around it, which reduces the information the model has over all the input space. At the extremes, if $\sigma \to \infty$ we obtain a constant function, and if $\sigma \to 0$ our model is only able to estimate input $\mathbf{x}$ equal to its SVs, having null generalization ability.

Figure 1 shows how each support vector contributes to the final solution in an artificial two-dimensional problem, for two different values of σ, $\sigma = 1$ and $\sigma = 0.02$, respectively the top (a, b) and bottom (c, d) graphs. The graphs on the right (a, c) present a separate curve for each support vector and the graphs on the left (a, d) present the resulting model (summation of all components). Three important conclusions are drawn:

1. for large σs the resulting surfaces are very smooth, and that might make it impossible to follow a brisk function.
2. small σs allow more complex and steep surfaces, but unless all input space is well covered, bad generalization will occur.
3. as there is only one σ for every SV, the only degree of freedom LS-SVMs have is the selection of the $\boldsymbol{\alpha}$ weights of each SV, which serve as a scaling factor to the respective component (the bias term, b, is another degree of freedom, but it is only an added constant).

Issues 1 and 2 could be solved by choosing an 'optimal' σ value, but it would still imply that all dimensions of the input space have the same behavior (all smooth or all steep). Issue 3 seems to be a major restriction of the models using a RBF kernel. The model output, $y(\mathbf{x})$, is computed based on the distance between the input vector $\mathbf{x}$ and all of the model SVs. The norm generally used is given by

$$||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\frac{\sum_{l=1}^{p}(x_{i_l} - x_{j_l})^2}{p}} \tag{3}$$

which provides no distinction between each of the p $\mathbf{x}$ dimensions.

3.2 Weighted Norm

Consider as an example a particular situation where:

1. $\mathbf{x} \in R^p$ are samples of the p inputs of a functional unit in a logic circuit.
2. one of them, x_r, has a huge effect on the power dissipated (for instance, a RESET signal).
3. we have a trained LS-SVM model with N support vectors, $N > p$.
4. two test vectors are given, $\mathbf{x}_1$ and $\mathbf{x}_2$, which are exactly the same except in their x_r component.

Since the only information the model uses to differentiate the output values $y(\mathbf{x}_1)$ and $y(\mathbf{x}_2)$ is their distance to all of the N support vectors, it is natural to assume that for $p \gg 1$ their distances to the N SVs would be almost the same which results in $y(\mathbf{x}_1) \approx y(\mathbf{x}_2)$. We know that their real outputs are very different and so the LS-SVM model can never give a good prediction of these values. A possible solution to this problem would be to get a very large number of SVs that would cover that critical input space where x_r varies. This is very costly and would prove to be extremely difficult since that means that there would be many SV close to each other, implying small σs and, thus, poor generalization ability.

To solve this problem, we propose modification to the RBF kernel which spawns from a simple adaptation of the distance measure used. By adding weights, $\boldsymbol{\beta}$, to each dimension of the input space, we add significantly more flexibility to the LS-SVM training procedure. The norm becomes

$$||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\frac{\sum_{l=1}^{p}\beta_l(x_{i_l} - x_{j_l})^2}{\sum_{l=1}^{p}\beta_l}} \tag{4}$$

Parameters $\boldsymbol{\beta}$ must be computed before training the model. In the case of power macromodelling, a formal method to obtain $\boldsymbol{\beta}$ would be to resort to the Shannon expansion for each circuit input [7]. In Section 4.3, we present a more expedite method.

Note that a similar weighing of the different dimensions can be applied to other kernels (Table 1), where the internal product needs to be modified to use a coefficient for each dimension.

4. IMPLEMENTATION

In this section, we describe the methodology for computing the power macromodel. We start by presenting how we obtain the data points used as the training set, then we describe the experiments we performed to tune the kernel parameters, and finally we discuss the size of the model and methods to reduce it.

4.1 Input Space Analysis

To generate the desired black-box macromodel it is necessary to obtain a set of data points to be plugged into the LS-SVM, $\{\mathbf{x}_k, z_k\}$. To analyze the performance of the LS-SVM method for power estimation, we need data points where z_k represents the power dissipation of a circuit under inputs $\mathbf{x}_k$. For this purpose, we can either use experimental values obtained from actual circuit measurements, or values computed by a simulator. Naturally, the accuracy of the model will be directly related to the quality of the data points.

Note that there is some flexibility in terms of what $\mathbf{x}_k$ represents. In this work, we are using the switching probability of each of the p inputs to the functional unit, hence a vector of size p with values between 0 and 1. Alternatively, we could aggregate all inputs and have their distribution probability (for example, in the case a set of bits represent some numerical value). Additionally, if specific information is available regarding joint probability distributions, it can be used to bias the choice of data points.

We should also observe that z_k can represent both static or dynamic power, or total power. The results we present in the next section were obtained from a logic simulator which only accounts for dynamic power. Yet, the results should easily extrapolate for static, and hence, total power.

We use the following error functions to provide some insight into LS-SVM model's performance:

Relative error: $E_R = \{\frac{|z_k - y(\mathbf{x}_k)|}{z_k}\}_{k=1}^{N}$

Average relative error: $E_1 = \frac{1}{N}\sum_{k=1}^{N} E_{R_k}$

Maximum relative error: $E_2 = \max_k E_{R_k}$

Fraction below 10% : $E_3 = \frac{|\{a \in E_R : a < 10\%\}|}{N}$

Ultimately we aim to achieve an E_3 of 100% and an E_1 smaller than 5%. Although the error values are not bounded, E_2 is a good representation of the worst-case scenario.

As data points used during training represent the total knowledge LS-SVMs have for model construction, their selection method is of crucial importance. For each circuit, the objective is to generate N vectors $\mathbf{x}$ of size p with values within $[0, 1]$. To effectively cover the input space three distributions were tested:

1. UNIFORM, 100% follow an uniform distribution between 0 and 1.
2. NORM, 100% follow a normal distribution with 0.5 mean and a chosen variance γ (every value not contained in the [0,1] interval is resampled).

Table 2: Comparison of input distributions.

Distribution	E_1	E_2	E_3
UNIFORM	1.68	23.1	99.7
UNMIX ($\gamma = 0.1$)	1.53	24.8	99.5
UNMIX ($\gamma = 0.3$)	1.5	24.2	99.6
NORM ($\gamma = 0.1$)	2.24	34.2	97.7
NORM ($\gamma = 0.3$)	1.57	26.5	99.4

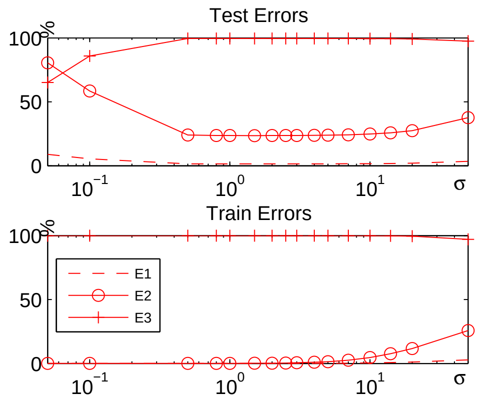

Figure 2: Errors for different values of σ.

3. UNMIX, a mix of 1 and 2, 50% follow an uniform distribution and 50% follow a normal distribution.

For the purpose of testing the generalization ability of our model, test sets of 8,000 points were constructed using equal quantities of each of the 3 methods presented, for each of the benchmark circuits. In Table 2 the errors defined earlier are shown, averaged over all the circuits.

We conclude that there isn't a great difference in error performance between the tested methods, but the UNMIX distribution seems to lead to slightly smaller errors. Adding to that, in practice, circuit input probabilities should be around 0.5 so we opted to use UNMIX with a variance $\gamma = 0.3$ to get a better representation in that area.

4.2 LS-SVM Parameters

The parameters that we have to tune are C, σ and the number of support vectors. As referred in Section 2.2, C is a constant that permits a tradeoff between the training error and the smoothness of the model. Our tests with different values of C and for all circuits indicate that the error decreases as C increases, but for values above $C = 10^4$ the error remains almost constant. Hence, and in order to potentially avoid numerical errors, we have set $C = 10^4$.

Figure 2 presents the error obtained for different values of σ, on test and training data, on top and bottom respectively. As it was expected, our experiments show that small values of σ allow very small training errors, but bad generalization ability. On the other hand, large values of σ made the training errors several orders of magnitude higher. Empirically our tests show that the optimum value should be $\sigma = 1.1$.

To determine the number of SVs, we performed similar experiments with increasing size of the training data set. We observed that errors decreased significantly up to a number

Table 3: LS-SVM test results.

Circuit	Circuit Information			Training			Testing- Usual Norm			Testing- Weighted Norm		
	Ins	Outs	Nodes	E_1	E_2	E_3	E_1	E_2	E_3	E_1	E_2	E_3
`add_cla_16`	32	16	214	0.0	0.0	100.0	1.5	15.3	99.9	1.4	14.8	99.9
`add_rpl_16`	32	16	214	0.0	0.0	100.0	1.3	14.4	99.9	1.3	13.0	99.9
`apex6`	135	99	411	0.0	0.1	100.0	5.0	30.8	90.9	1.7	12.8	99.9
`frg2`	143	139	522	0.0	0.1	100.0	5.6	48.6	86.3	1.4	15.8	99.9
`i5`	133	66	161	0.0	0.0	100.0	1.8	11.2	100.0	1.7	10.2	100.0
`i6`	138	67	318	0.0	0.1	100.0	7.4	72.0	78.4	1.4	26.0	99.4
`i7`	199	67	406	0.0	0.2	100.0	7.5	75.2	77.2	1.4	23.4	99.4
`i8`	133	81	1183	0.0	0.2	100.0	9.7	117.3	63.3	1.9	41.4	99.2
`mult8`	16	16	176	0.0	0.0	100.0	1.0	20.8	99.8	0.9	18.8	99.9
`pair`	173	137	877	0.0	0.0	100.0	2.7	15.9	99.6	1.6	10.2	100.0
`prolog`	36	73	424	0.0	0.0	100.0	3.7	25.8	96.7	1.3	12.5	100.0
`rot`	135	107	390	0.0	0.0	100.0	2.8	18.6	99.1	1.5	12.7	100.0
`vda`	17	39	304	0.0	0.0	100.0	1.9	50.7	98.2	1.1	39.2	99.3
`x1`	51	35	174	0.0	0.0	100.0	3.0	36.0	98.3	1.5	21.9	99.8
`x3`	135	99	332	0.0	0.0	100.0	3.9	26.3	96.6	1.1	10.6	100.0
`x4`	94	71	211	0.0	0.0	100.0	5.5	43.2	88.6	1.9	16.3	99.7
						average:	**4.0**	**38.9**	**92.1**	**1.4**	**18.7**	**99.8**

of SVs around 2,000. Since larger values of SVs translate linearly to the size of the model and its computation time, we settled the number of SVs to 2,000.

4.3 Computing the Input Weights

In order to gauge the relative importance of each input to the functional unit in terms of the impact in power consumption, we performed a set of experiments where we set all other inputs to a fixed value and measure the power as we change the value of the input under evaluation. Naturally the results obtained depend on the values assigned to the other inputs. Hence, we repeat this procedure for a set of 20 different combination of values for the remaining inputs.

From these experiments, we compute the power range for each input, as the difference between the maximum and minimum power values. We use this value directly as the weight for this input in the computation of the modified norm.

4.4 Model Size

From Equation 2, we know that evaluating the LS-SVM model to compute the power requires the sum of N elements, and each element contains a norm (Equation 4) which is a sum of p elements. Hence, the model requires $O(Np)$ operations to compute $y(\mathbf{x})$. In our case, we have set $N = 2,000$, thus the computation time of the model is linear in the number of circuit inputs. In any case, it should be extremely fast in practice and should not be an issue.

It terms of memory, we need to store: N SVs, each of size p; N α values; and the constant b. Hence, the non modified kernel has $O(N(p + 1) + 1) = O(Np)$ memory complexity. Our modified kernel adds p input weight coefficients (β), which has a negligible impact on memory usage. If we use the float data type to store these values (usually 4 bytes long), a model of a circuit with 200 inputs ($p = 200$) and $2,000$ SVs ($N = 2000$) will need $(2000 \times (200 + 1) + 200 + 1) \times 4 = 1.53MB$ of storage space. It is affordable, but still expensive. Next, we discuss methods to reduce this size.

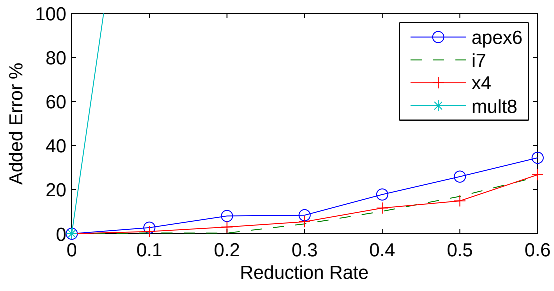

Figure 3: Error increase as p is reduced.

4.5 Model Pruning

There are two parameters that define the model size, N and p. There are methods that search "redundant" SVs, *i.e.*, SVs whose removal has minimal impact on the error [3].

In this paper, we further investigate the reduction of p. The strategy is to remove input dimensions with the lowest β values, as these are the ones which have less impact in defining the power characteristics of the circuit. Figure 3 shows the increase in error with the reduction of a fraction of p.

Test results show that we can effectively reduce the model size by selectively ignoring input dimensions. However, this approach is highly dependent on the circuit to which it is applied to since it depends solely on β values. If βs range over a small interval then removing the lowest ones is still removing important elements of the norm calculation. This is excellently demonstrated by the behavior observed on the chosen circuits. As `mult8` is the only circuit which did not benefit from the weighted norm (see Table 3), that means its β values should all have about the same values, and so are all equally important, resulting in the biggest performance hit in Figure 3. To solve this problem, the weight should be normalized and only those below a given threshold should be removed.

Table 4: Comparison with the results of the table-lookup method of [3].

Circuit	Circuit Information			Testing-Usual Norm			Testing- Weighted			4D Lookup Tables	
	Ins	Outs	Nodes	E_1	E_2	E_3	E_1	E_2	E_3	E_1	E_2
C499	41	32	202	3.47	21.3	98.0	0.42	4.0	100	3.95	16.3
C880	60	26	383	6.94	69.2	75.8	1.56	16.0	99.8	3.62	14.0
C1355	41	32	546	3.11	17.2	98.9	0.49	5.3	100	4.03	15.0
C1908	33	25	880	4.90	41.7	91.6	1.01	10.8	99.9	3.73	15.7
C2670	233	140	1193	5.32	36.2	87.4	1.26	9.2	100	2.18	10.2
C3540	50	22	1669	5.86	50.4	85.4	1.08	14.0	99.9	3.22	15.6
C5315	178	123	2307	4.03	21.5	96.5	0.96	5.9	100	2.08	12.2
C6288	32	32	2406	3.86	44.7	95.7	0.45	9.1	100	2.22	17.4
						average:	**0.90**	**9.3**	**100**	**3.13**	**14.5**

5. RESULTS

In Table 3, the performance of the LS-SVM models computed from training sets of $N = 2,000$ samples generated by the UNMIX method (Section 4.1), with $\gamma = 0.3$, is evaluated, using test sets of $8,000$ samples. Weighted norm tests were done using $C = 10^4$, $\sigma = 1.1$ versus our best achieved results using non-weighted norm ($C = 10^4$, $\sigma = 3$).

We can observe that the results obtained with the original kernel were already very good, with, on average, average error (E_1) close to 4%, maximum error (E_2) below 50% and more than 92% of all results with error below 10% (E_3).

However, notice that the proposed modified norm still has a significant impact in reducing these errors. On average, E_1 is reduced to a third while maximum relative errors (E_2) were reduced to half. Nearly 100% of the estimations have relative errors below the 10% mark (E_3).

In Table 4 we compare the accuracy of the LSSVM-based macromodels with the macromodels based on table-lookups, as proposed in [2]. For the circuits with results provided in [2], we can observe that our models achieve below 1% average-error over all circuits while the average-error of the table-lookup is above 3%. Similarly, the average maximum error of our method is below 10%, compared with 14.5% of the table-lookup. Another interesting observation is that this set of circuits seems to be easier to model than the circuits of Table 3, as the errors obtained are lower. An explanation for this effect is that the larger size of the circuits of Table 4 may lead to a less skewed distribution of the power dissipation with respect to input probability distribution.

6. CONCLUSIONS

Our experiments show that LS-SVM are a viable method for the generation of power macromodels. Modifying the basic RBF kernel so that it takes into account the impact of different circuit inputs on dissipated power proved to result in a huge improvement in model accuracy. It also opened doors to a new approach to model size reduction based on input dimension weights and the use of the weighted norm on other kernels. Further work involves applying kernels composed of more than one element [8] and finding a way of using different variances σ values for each support vector of the RBF kernel.

7. REFERENCES

[1] A. Bogliolo, L. Benini, and G. De Micheli. Regression-Based RTL Power Modeling. *ACM Transactions on Design Automation of Electronic Systems*, 5(3):337–372, 2000.

[2] S. Gupta and F. Najm. Power Modeling for High-level Power Estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8:18–29, 2000.

[3] Luc Hoegaerts. *Eigenspace Methods and Subset Selection In Kernel Based Learning.* PhD thesis, Katholieke Universiteit Leuven, June 2005.

[4] E. Macii, M. Pedram, and F. Somenzi. High-level Power Modeling, Estimation, and Optimization. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, pages 1061–1079, 1998.

[5] A. Raghunathan, S. Dey, and N.K. Jha. High-level Macro-modeling and Estimation Techniques for Switching Activity and Power Consumption. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(4):538–557, Aug. 2003.

[6] W. Shang, S. Zhao, and Y. Shen. Application of LSSVM with AGA Optimizing Parameters to Nonlinear Modeling of SRM. *Industrial Electronics and Applications, 3rd IEEE Conference on*, pages 775–780, June 2008.

[7] C. Shannon. The Synthesis of Two-Terminal Switching Circuits. *Bell System Technical Journal*, 28:59–98, Jan. 1949.

[8] G. Smits and E. Jordaan. Improved SVM Regression using Mixtures of Kernels. *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, 3:2785–2790, 2002.

[9] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines.* World Scientific Pub., 2002.

[10] V. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, 1998.