Power Macro-Modelling using an Iterative LS-SVM Method

António Gusmão L. Miguel Silveira José Monteiro

INESC-ID / IST, TU Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
{antoniog,lms,jcm}@algos.inesc-id.pt

***Abstract*—We propose a new method for power macromodelling of functional units for high-level power estimation based on Least-Squares Support Vector Machines (LS-SVM). Our method improves the already good modelling capabilities of the basic LS-SVM method in two ways. First, a modified norm is used that is able to take into account the weight of each input for global power consumption in the computation of the kernels. Second, an iterative method is proposed where new data-points are selectively added as support-vectors to increase the generalization of the model. The macromodels obtained provide not only excellent accuracy on average (close to 1% error), but more importantly, thanks to our proposed modified kernels, we were able to reduce the maximum error to values close to 10%.**

I. Introduction

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 [8]. The general approach is to use power macromodels for each functional unit. These macromodels are obtained in a pre-characterization phase, stored in a library for later use and represent the power dissipation of the unit as a function of its input statistics.

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) [16]. 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 (LS-SVM) 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.

Our proposed methodology improves the already good modelling capabilities of the basic LS-SVM method in two ways. In general, kernels treat every dimension uniformly. In the problem at hand, each input to the functional module defines a dimension for the kernel (although not necessarily so after optimizations, as discussed in Section IV-E). It is 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. Secondly, an iterative method is proposed where new data-points are selectively added as support-vectors to better generalize the model

We present results that confirm the excellent modelling capabilities of the kernel-based methods. The macromodels obtained provide not only excellent accuracy on average (all below 2% average error and close to 1% on average), but, more importantly, thanks to our modified kernels, we were able to reduce the maximum error to values close to 10%.

The paper is organized as follows. In Section II, we review previous work on power analysis techniques at the RT level and provide some background on kernel methods. Section III discusses the kernel parameters and presents the modified norm. The implementation of the power macromodelling process is described in Section IV. The iterative optimization process is proposed in Section V. We present our results in Section VI and draw conclusions and future work in Section VII.

II. Related Work

A. Power Macro-Modeling

There has been a fair amount of work on generating models for power dissipation at higher levels of abstraction. Top-down approaches have been proposed in [9] and [10]. They are both based on the concept of entropy and their focus is to derive implementation-independent measures of the signal activity in the circuit. A number of assumptions are made in both [9] and [10] on how to propagate the entropy of the inputs through the circuits. These methods can be very efficient, though given all the required approximations and the fact that they ignore issues such as glitching implies that these techniques are not very accurate.

Our method follows a bottom-up approach (for a survey, see [8]), where the model is obtained from an actual circuit implementation. This offers the best level of accuracy. These methods build their models from data points that consist of 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 [3]. N-dimensional tables have been 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], [2], [11], [17]. A combination of both of these methods has also been proposed [3]. Models for specialized functions, such as arithmetic units, use different expressions for different inputs, namely depending on whether they correspond to the most or the least significant bits [6], [7]. Using models tailored to specialized functions, however, requires very specific knowledge of the problem which restricts the applicability of the technique. Table interpolation or regression implicitly assumes a polynomial or spline representation and may require a large number of coefficients if the underlying surface is somewhat nonlinear. SVMs are quite useful in such a context as they can readily adapt to the data at hand, even if it exhibits strong nonlinearities. Therefore, much sparser approximants can be used while retaining or even improving on the accuracy.

We believe the model we propose, based on LS-SVMs, is more robust than previously proposed approaches: it is generic, systematic, 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.

B. 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 $y(\mathbf{x})$, an 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.

A version of a SVM for regression was proposed by Vapnik el al [16]. 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 were introduced [15]. In both methods the model is:

$$y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + b \tag{1}$$

The $\varphi(\mathbf{x})$ mapping is usually a non-linear function that transforms the data into a higher dimensional feature space, and is weighted by $\mathbf{w}$. Constant b is the bias term.

LS-SVM correspond to solving the following constrained optimization problem:

TABLE I
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:	$\Psi(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\phi \mathbf{x}_i^T \mathbf{x}_j + \theta)$

$$\min_{\mathbf{w}} J = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{k=1}^{N} e_k^2 \quad \text{s.t.} \quad z_k = \mathbf{w}^T\varphi(\mathbf{x}_k) + b + e_k \tag{2}$$

The $\mathbf{w}^T\mathbf{w}$ term stands for minimizing the length of the weight vector, while the C constant is the trade-off parameter between the complexity of the representation and the minimization of training data errors. The number of training samples, known as support vectors, is given by N.

Using Lagrange multipliers, in order to transform the problem into an unconstrained optimization problem, gives:

$$L = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{k=1}^{N} e_k^2 - \sum_{k=1}^{N} \alpha_k[\mathbf{w}^T\varphi(\mathbf{x}_k) + b + e_k - z_k] \tag{3}$$

Which is guaranteed to have a global minimum when:

$$\frac{\partial L}{\partial \mathbf{w}} = 0; \frac{\partial L}{\partial b} = 0; \frac{\partial L}{\partial e_k} = 0; \frac{\partial L}{\partial \alpha_k} = 0,$$

resulting in the following linear system of equations:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \boldsymbol{\Omega} + C^{-1}\boldsymbol{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix} \tag{4}$$

where $\boldsymbol{\Omega}$ is the kernel matrix, $\Omega_{kl} = \Psi(x_k, x_l) = \varphi(x_k).\varphi(x_l)$, $k, l = 1, \ldots, N$, and Ψ is the kernel function. The resulting LS-SVM is given by

$$y(\mathbf{x}) = \sum_{k=1}^{N} \boldsymbol{\alpha}_k \Psi(\mathbf{x}, \mathbf{x}_k) + b \tag{5}$$

The simplicity of (5) is not without a cost. 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 I. In this work, we will focus exclusively on the RBF kernel since it has good empirical results and has a nice smooth behavior.

III. 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 then we present a modification to the norm used by the RBF kernel that allows different weights for each input of functional units.

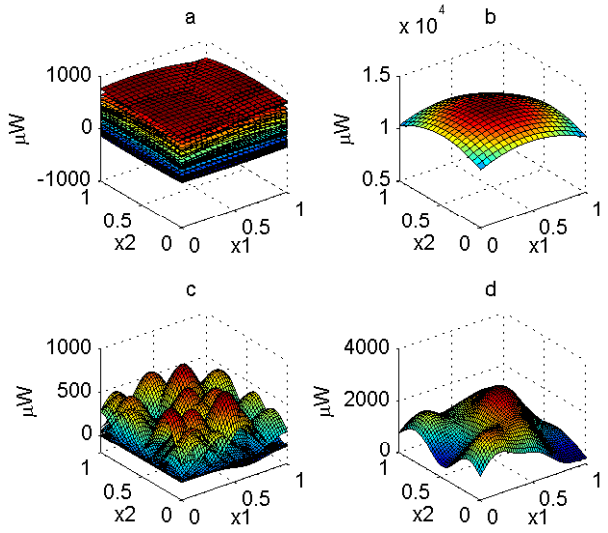

Fig. 1. Power surfaces for: a) $\sigma = 1$, components; b) $\sigma = 1$, sum; c) $\sigma = 0.02$, components; d) $\sigma = 0.02$, sum.

A. Kernel Behavior

There is some work on how to choose the 'optimal' parameters, C and σ, for the RBF kernel [12]. 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 I), 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 a single σ for all SVs, the only degree of freedom LS-SVMs have is the selection of the $\boldsymbol{\alpha}$ weights of each SV, which serves as a scaling factor to the respective component curve (the bias term, b, is one more 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

$$||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\frac{\sum_{l=1}^{p}(x_{i_l} - x_{j_l})^2}{p}} \tag{6}$$

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

B. 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 introduce our proposed 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

$$||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\frac{\sum_{l=1}^{p}\beta_l(x_{i_l} - x_{j_l})^2}{\sum_{l=1}^{p}\beta_l}} \tag{7}$$

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 [13]. In Section IV-C, we present a more expedite method.

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

IV. 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.

A. 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:

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

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

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

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

Ultimately we aim to achieve an E_3 of 100% and an E_1 smaller than 1%. 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. To effectively cover the input space we randomly generate a set of points where half follow an uniform distribution and half follow a normal distribution with variance $\gamma = 0.3$ (both with average 0.5) [4].

B. LS-SVM Parameters

The parameters that we have to tune are C, σ and the number of support vectors.

As referred in Section II-B, 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

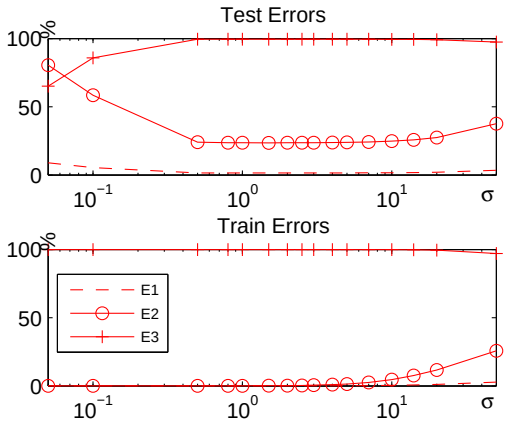

Fig. 2. Errors for different values of σ.

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 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.

C. 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. For this reason, 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.

D. Model Size

From Equation 5, 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 7) which is a sum of p elements. Hence, the model requires $O(Np)$ operations to compute $y(\mathbf{x})$. Computing the model's output is much faster than simulating the circuit. On a 3GHz AMD64 it took no

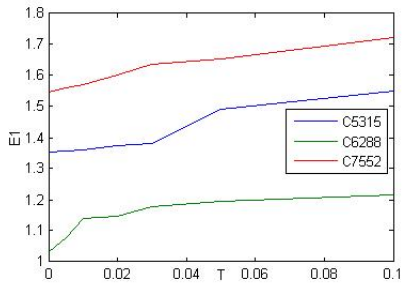

Fig. 3. Error increase with the increase of T.

more than 10 seconds to estimate 10,000 outputs for any of the tested circuits.

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.

E. Model Pruning

There are two parameters that define the model size, N and p. There are methods that reduce N by searching for "redundant" SVs, *i.e.*, SVs whose removal has minimal impact on the error [5].

In our work, we apply one of the methods prescribed in [5]. In each step the algorithm solves the linear system (eq. 4) and removes the least important SV which is the $\mathbf{x}_k$ with the smallest d($\mathbf{x}_k$):

$$d(\mathbf{x}_k) = [A.\Delta\mathbf{x}]_{k+1} = \frac{\alpha_k}{[A^{-1}]_{k+1}} \tag{8}$$

where A is the square matrix from Equation 4. This is an expensive method but has a good model-reduction to model-error ratio. For a more detailed description we refer the reader to the original paper.

Additionally, we investigate the reduction of p, which amounts to a form of feature selection. The process is to simply 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. Starting from lower β, we remove dimensions until the sum of removed betas (normalized) is below a user specified threshold $T \in [0, 1]$. Figure 3 shows the increase in error with the increase of T.

V. Model Optimization

The analysis in the previous section indicates that the best number of support vectors to use and the value of the σ parameter are interdependent. We propose an iterative method to automatically determine these parameters so that we maximize the generalization of the LS-SVM-based power macromodel.

The standard method of computing the kernel parameters is simply to solve the linear system given in Equation 4, using a set of data points $\{\mathbf{x}_k, z_k\}_{k=1}^N$, the training set. The model obtained can then be evaluated against a different and disjoint set of datapoints, the test set. From the obtained errors in each of these sets, we can conclude:

1) large errors on the test set might be due to two reasons: the 'area' of the input space where samples have large test errors is badly covered by the training set; or the kernel function is excessively local, which means that the influence of each SV is limited to a very small 'area' around it.
2) large errors on the training set indicate that the kernel function is smoother than it should be, not having enough flexibility to approximate steep surfaces.

To address these issues an extension to the training procedure was devised consisting of an iterative addition of SVs.

The iterative method developed starts from a given LS-SVM model and uses the following three sets:

- the final test set, with data points which will only be used to evaluate the final model (hence, after the iterative process has finished)
- an initial training set of size M_T of the given model.
- a validation set of size M_V, which can be much larger than M_T

The following steps are repeated while user specifications have not been met, namely that E_1 and E_2 on the validation set are smaller than some thresholds T_{E_1} and T_{E_2}:

1) to increase the generalization, we move k data points with the largest errors from the validation set into the training set.
2) if the error on the training set exceeded specifications, we reduce parameter σ by s. As discussed in the previous section, the σ parameter defines how local or global the impact of each support vector is. Hence, by reducing this value by a factor of s, $s < 1$, the error is reduced on the training set. This may potentially lead to worse generalization, which is compensated by the extra data points that are moved to the training set. Nevertheless, σ should not be greatly reduced between iterations.

The initial σ value should be relatively large to start from a very smooth solution and steeping it only as much as necessary by reducing σ. Parameters k and s define the granularity of the process. Larger values will reduce the number of iterations, but may lead to a larger model. The size of the initial model and respective training set should be relatively small to give room for the addition of the more problematic data points in the validation set, but, at the same time, large enough to give reasonably good predictions. In our experiments, we start from a model of size 500 obtained from applying (8) to a training set of size 2,000.

VI. Results

In Table II we present results for circuits of the ISCAS benchmark set, obtained under different methods. Here we compare several modifications of the basic LS-SVM method

TABLE II
LS-SVM TEST RESULTS.

Circuit	Circuit Info		Usual Norm			Weighted Norm			Iter. Weighted Norm				4D Tables		Pruned			
	Ins	Nodes	E_1	E_2	E_3	E_1	E_2	E_3	N	E_1	E_2	E_3	E_1	E_2	Ins	E_1	E_2	E_3
C499	41	202	3.5	21.3	98.0	0.4	4.0	100	500	0.5	3.7	100	3.9	16.3	33	1.0	6.0	100
C880	60	383	6.9	69.2	75.8	1.6	16.0	99.8	1000	1.4	17.6	100	3.6	14.0	39	1.7	16.6	99.8
C1355	41	546	3.1	17.2	98.9	0.5	5.3	100	500	0.5	3.7	100	4.0	15.0	33	0.9	5.9	100
C1908	33	880	4.9	41.7	91.6	1.0	10.8	99.9	695	1.1	9.8	100	3.7	15.7	28	1.3	11.1	100
C2670	233	1193	5.3	36.2	87.4	1.3	9.2	100	1000	1.2	7.2	100	2.2	10.2	107	1.3	8.7	100
C3540	50	1669	5.9	50.4	85.4	1.1	14.0	99.9	740	1.2	11.6	99.9	3.2	15.6	29	1.2	15.1	99.9
C5315	178	2307	4.0	21.5	96.5	1.0	5.9	100	600	1.1	5.9	100	2.1	12.2	94	1.1	7.0	100
C6288	32	2406	3.9	44.7	95.7	0.5	9.1	100	500	0.6	5.4	100	2.2	17.4	29	1.0	10.7	100
C7552	207	3512	4.4	27.1	96.2	1.3	13.4	99.9	1000	1.3	11.2	99.9	2.7	14.3	124	1.3	13.1	100
average:			4.7	36.6	91.7	0.95	9.73	99.9	726	1.0	8.46	99.98	3.1	14.5		1.2	10.5	99.97

with state of the art table lookup methods. In the table, E_1 represents average error, E_2 maximum error and E_3 estimates with error below 10%. Under "Usual Norm" we give the performance of the base LS-SVM models, computed using $C = 10^4$, $\sigma = 1.1$ (Section IV-B) and from training sets of $N = 2,000$ samples, generated as described in Section IV-A. Weighted norm tests are shown in columns "Weighted Norm", under the same conditions. Under "Iter. Weighted Norm" are the results after the iterative process, where N indicates how many support vectors were used. The results reported by [3] are given under "4D Tables". Finally, under "Pruned" are the results after applying the pruning technique described in Section IV-E, using a threshold value of $T = 0.08$.

Several interesting observations can be made from the results presented in Table II. First, we note that the results obtained with the original kernel were already very good, with average error E_1 below 5%. However, maximum error E_2 can be large, with circuit C880 presenting an error close to 70%. The use of the proposed weighted norm is by itself very effective in reducing error. Its impact is especially visible on the maximum error, which on average reduces to just under 10%, with circuit C880 still presenting the largest maximum error, but reducing from 70% to 16%. On average, the E_1 comes to under 1% and only 0.06% of the samples have error above 10% (E_3)!

The next set of results demonstrates that the iterative process allows for significantly smaller models sizes, while practically maintaining the error levels. Whereas for the previous results a constant number of 2,000 support vectors is used in the model, the iterative process finds equally good solutions with only 726 support vectors on average, providing models almost $3\times$ more compact.

The model size can still be effectively reduced using the proposed pruning method, without significant effect on the error level. For larger examples, the number of inputs effectively used in the model are close to half of the original circuit, leading a model half the size and half the evaluation time.

VII. 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 [14].

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] Zhanping Chen and K. Roy. A power macromodeling technique based on power sensitivity. *Design Automation Conference, 1998. Proceedings*, pages 678–683, Jun 1998.

[3] 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.

[4] A. Gusmao, L.M. Silveira, and J. Monteiro. Parameter tuning in svm-based power macro-modeling. In *Quality of Electronic Design (ISQED09)*, pages 135–140, March 2009.

[5] L. Hoegaerts, J.A.K. Suykens, J. Vandewalle, and B. De Moor. Primal space sparse kernel partial least squares regression for large scale problems. *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, 1:–563, July 2004.

[6] G. Jochens, L. Kruse, E. Schmidt, and W. Nebel. A new parameterizable power macro-model for datapath components. *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, pages 29–36, 1999.

[7] P.E. Landman and J.M. Rabaey. Architectural power analysis: The dual bit type method. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 3(2):173–187, Jun 1995.

[8] 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.

[9] D. Marculescu, R. Marculescu, and M. Pedram. Information Theoretic Measures of Energy Consumption at Register Transfer Level. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 81–86, April 1995.

[10] F. Najm. Towards a High-Level Power Estimation Capability. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 87–92, April 1995.

[11] 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.

[12] 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.

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

[14] 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.

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

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

[17] Qing Wu, Qinru Qiu, M. Pedram, and Chih-Shun Ding. Cycle-accurate macro-models for rt-level power analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(4):520–528, Dec 1998.