*Journal of
Low Power Electronics*
Vol. 9, 1–10, 2013

Fast Power Evaluation for Effective Generation of Test Programs Maximizing Peak Power Consumption

P. Bernardi[1], M. De Carvalho[1,*], E. Sanchez[1], M. Sonza Reorda[1], A. Bosio[2], L. Dilillo[2], M. Valka[2], and P. Girard[2]

[1] *DAUIN, Politecnico di Torino,Turin, 10129, Italy*
[2] *LIRMM, University of Montpellier II/CNRS, Montpellier, France*

(Received: 22 February 2013; Accepted: 17 June 2013)

High power consumption during test may lead to yield loss and premature aging. In particular, excessive peak power consumption during at-speed delay fault testing represents an important issue. In the literature, several techniques have been proposed to reduce peak power consumption during at-speed LOC or LOS delay testing. On the other side, limiting too much the power consumption during test may reduce the defect coverage. Hence, techniques for identifying upper and lower functional power limits are crucial for delay fault testing. Yet, the task of computing the maximum functional peak power achievable by CPU cores is challenging, since the functional patterns with maximum peak power depend on specific instruction execution order and operands. In this paper, we present a methodology combining neural networks and evolutionary computing for quickly estimating peak power consumption. The method is used within an algorithm for automatic functional program generation used to identify test programs with maximal functional peak power consumption, which are suitable for defining peak power limits under test. The proposed approach was applied on the Intel 8051 CPU core synthesized with a 65 nm industrial technology reducing significant time with respect to old methods.

Keywords: Test Power, Functional Peak Power, Automatic Functional Program Generation, Neural Networks, Evolutionary Computing.

1. INTRODUCTION

Nowadays, semiconductor product design and manufacturing are being affected by the continuous CMOS technology scaling. On the other side, high operation speed and high frequency are mandatory requests, while power consumption is one of the most significant constraints, not only due to the large diffusion of portable devices. This influences not only the design of devices, but also the choice of appropriate test schemes that have to deal with production yield, test quality and test cost.

Testing for performance, required to catch timing or delay faults, is mandatory, and it is often implemented through at-speed testing.[1] Usually, performance testing involves high frequencies and switching activity (SA), thus triggers significant power consumption. As a consequence, performance test may produce yield loss (due to over-stress), e.g., when a good chip is damaged while testing. In Ref. [2], the authors analyzed these phenomena and demonstrated that in some cases, yield loss may also occur when a good chip is declared as faulty during test (again due to over-stress). Hence, reduction of test power is mandatory to minimize the risk of yield loss; however, some experimental results have also proved that too much test power reduction might lead to test escape and create reliability problems because of the under-stress of the circuit under test (CUT).[1]

Hence, it is crucial to know which is the maximum peak power consumption achievable by the CUT under normal operational conditions. In this way the designer can check whether it can be tolerated by the technology and the test engineer can develop proper stimuli to force the CUT to work in these conditions.

In our previous work,[3] we presented a flow able to precisely measure power consumption during test and functional modes of a processor. We proposed a methodology to generate functional stimuli able to maximize peak power consumption during the operational mode. The approach proposed in Ref. [3] uses an evolutionary algorithm able to

[*] Author to whom correspondence should be addressed.
Email: mauricio.decarvalho@polito.it

create functional patterns (i.e., assembly programs) maximizing the functional peak power consumption. This novel strategy is effective but time consuming, since it uses slow simulations of the CUT in SPICE-like transistor-level description, where the final result may be obtained after weeks or even months depending on the desired level of details.

In another work,[4] we presented a detailed analysis describing the correlation among switching activity and power consumption obtained by feeding the CUT with the functional patterns. The results demonstrate that the overall switching activity strongly correlates with the overall power consumption, but one-to-one correlation among the identified peaks is weak. So, we proposed a method able to increase the maximum number of identified peaks of power by analyzing the peak switching activity occurring at different instants.

In this paper we propose to speed-up the automatic functional pattern generation approach proposed in Ref. [5] by exploiting an intelligent and fast power estimator based on mimetic learning for increasing functional peak power of CPU cores. In Ref. [5] the approach employs an evolutionary algorithm (EA) that generates functional patterns and improves them based on their power consumption. However, power consumption evaluation is very slow, especially if we need to evaluate several hundred thousands of patterns. To speed-up the framework we propose a fast power estimator to be used in conjunction with the EA, thus reducing the evaluation time from weeks to days. In particular, we propose a method for fast power estimation based on neural networks whose inputs are the new switching activity per gate type and the output is the estimated power. In order to build the fast power estimator we need to perform the following tasks:

(1) Analysis of the total switching activity within each clock cycle

(2) Computation of a new SA (NSA) metric, by properly weighting transitions and glitches

(3) Training a feed-forward neural network (FFNN) by using a 2-phase strategy which assigns proper weights to different gate types according to their NSA

(a) Generation phase: it increases the search space by generating proper patterns

(b) Training phase: it adapts the neural network for the newly generated patterns

(4) Increasing the functional peak power by using the FFNN as a fast power estimator within the automatic functional pattern generation engine. The NSA per gate type is inserted in the FFNN and the power estimation is used as the feedback value for the EA.

In the proposed approach we are not trying to develop a new method for computing the exact power consumption, because power evaluators already exist by several vendors. Instead, we are proposing a method to quickly estimate peak power consumption; therefore, the method may sometimes miscompute power, but is generally able to correctly drive the EA to generate peak power effective patterns. The approach can reduce time from weeks to days providing a functional peak power consumption measure effective for mapping test power. The main novelty of the proposed strategy lies in its ability to improve the automatic functional pattern generation framework by inserting a FFNN-based external power evaluator which is faster than commercial ones.

We validated the proposed methodology using the Intel 8051, synthesized using a 65 nm industrial technology. The training process of the FFNN required two days. Also, the final generation using the trained FFNN reduced a single evaluation time by more than 60% with respect to commercial tools, while always individuating the test program points where the peak power was maximized.

The rest of the paper is structured as follows: in Section 2 we discuss background concepts about power evaluation metrics and functional pattern generation for a CPU core. Section 3 describes the proposed strategy to identify the subset of time points within a functional test pattern set that are the best candidates for achieving the highest power consumption. Section 4 describes the case studies and Section 5 shows the results obtained on a sample processor core. Section 6 draws some conclusions.

2. BACKGROUND

Performance testing of low-power devices is crucial and challenging. In extremely small technologies, this type of test may reduce production yield due to high test power produced by the excess of toggle activity that may reach twice as much than functional mode.[1, 2] In scan-based tests, internal flip-flops are linked to form a shift register. Input test patterns are shifted in as quickly as possible in test mode, then several clock cycles are executed under functional mode, and finally output test values are shifted out. In some cases test data simultaneously stimulate parts of the circuit which would not have been stimulated under functional mode. If frequency and switching activity are increased during test so will the power consumption, as forecasted by formula (1). On the other side, we know that in low-power devices synthesized in small technologies the power consumption is an issue since it may very easily damage the component.

The average dynamic power consumption of a CMOS circuit can be expressed as:

$$\text{Power} = \frac{1}{2} \times f \times vdd^2 \sum_{i=1}^{n} C_i \times \alpha_i \tag{1}$$

where: f is the clock frequency, Vdd is the supply voltage, C_i and α_i are capacitance load and switching activity of node i, respectively. Most of evaluators use formula (1) for estimating power consumption in a CMOS circuit. For a well designed circuit, the power can be approximated by the switched-capacitance.[6, 12]

The use of scan-chains for testing the circuit in a non functional mode may significantly increase the switched-capacitances, leading to high power consumption.

Several works investigate power modeling by approximating CMOS capacitances.[7, 8, 12] In Ref. [7], the authors model capacitances for several logic gates which allow estimating power at SPICE level. Results report that gates with larger capacitances have higher power consumption. In Ref. [8], the authors propose modeling capacitances per blocks according to the type of logic used to estimate power. These works mainly allow the correct estimation of the average power consumption.

On the other side, in Ref. [4] the authors investigate peak power estimation of CPU cores using different tools. The functional peak activities and power consumption of the following levels were compared:

- register-transfer-level switching activity (RTL SA)
- gate-level switching activity (GL SA)
- weighed switching activity (WSA)
- transient analysis power evaluation (TA)
- post-layout analysis power evaluation (PLA).

From this comparison, it was verified that normal activities were very strongly correlated but peak activities did not correlate at all.

Table I qualitatively summarizes the characteristics of each method in terms of accuracy and computational requirements. As we can see from the table, RTL SA is the fastest evaluation we could perform, however it does not have a good level of detail and power cannot be evaluated with reasonable accuracy. As we go further down on the table we have more accurate results but the time for obtaining them increases significantly.

In Ref. [3] the authors consider structural tests reducing power consumption belonging to both launch-on-shift (LOS) and launch-on-capture (LOC) types and compare the power consumption during performance testing with the functional power during normal operations. They propose a power evaluation flow for CPU cores at SPICE level, which allows comparing LOS/LOC test power with functional power. Results show that test power is 14% more than functional power in 90 nm technology and almost 47% more under 65 nm. In addition, if technology scales and frequency increases during test the power can be even more harmful for particularly low-power devices.

Table I. Power evaluation methods.

Method	Accuracy	Computational cost
RTL SA	Very low	1X
GL SA	Low	2X
WSA	Medium/High	10X
TA	High	300X
PLA	Very high	1,000X

In Ref. [4], maximum peak power estimation at TA by analyzing gate-level SA was proposed. Overall correlation indexes among the abovementioned evaluation methods are strong, ranging from 0.8 to 0.9, although it was verified that one-to-one relationship at maximum peaks do not correlate. This is due to the fact that the impact on power consumption of glitches and valid transitions were considered equally.

In Ref. [9] the effects of glitches on the power consumption have been studied. They are shown to be responsible for 10% up to 60% of the overall power consumption, while the rest (from 40% to 90%) is due to valid transitions. This wide range depends on the circuit description, the used library, and the adopted synthesis parameters. If all these features have been carefully selected glitches minimally impact power consumption.

The peak power of functional patterns obtained in Refs. [10] and [11] was studied in Ref. [3]. The former work exploits an automatic functional pattern generation approach for maximizing stress by trying to activate every gate in the CPU core. The latter provides code snippets containing instructions able to increase power consumption. In both works a methodology for automatic functional pattern generation proposed in Ref. [5] was exploited. It employs an evolutionary algorithm able to generate functional patterns effective in maximizing a certain metric arising from specific circuit stimuli.

The maximum functional peak power value can be obtained by combining the automatic functional pattern generation with power evaluators (e.g., WSA, TA and PLA) for extracting peak values. However, the approach generates and evaluates hundreds of thousands of programs before converging to a desired maximum peak power value. Therefore, the evaluation phase must be very quick and able to generate effective programs in a reasonable time. Ideally, we would like to maximize metrics at RTL (i.e., peak SA in a clock cycle) and observe an increasing trend in PLA power consumption. Yet, this specific metric is still missed and alternative solutions shall be proposed.

3. PROPOSED METHODOLOGY

In this paper we propose to speed-up the automatic functional pattern generation approach by developing an intelligent fast power estimator based on mimetic learning. The whole environment we refer to is shown in Figure 1. Since our target is CPU cores, the input functional patterns are assembly programs. In Figure 1, a power estimator tool is introduced in the evaluation phase to determine the maximum peak power and thus driving the EA for correctly generating functional patterns with maximum peak power consumption (*survival of the fittest*). For our objective it is an efficient tool because it randomly generates functional patterns for CPU cores, and evaluates them resorting to

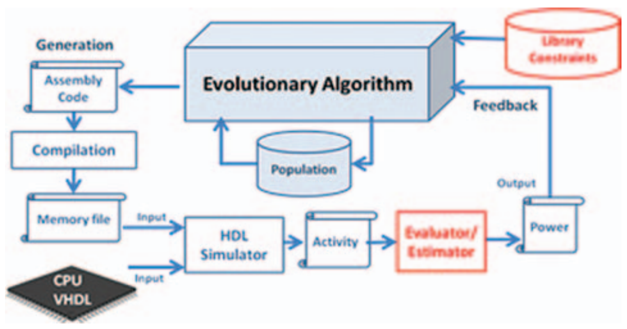

Fig. 1. Automatic functional pattern generation framework.

an external evaluator. It is then able to discard assembly programs or increase their strength according to the power result received from the evaluator. We resort to EA because it is able to maximize the power metric from an initial random population, discarding the weak ones and improving the stronger patterns. Random approaches are less effective than EA because generating functional patterns is an NP-hard problem where testing all combinations is computationally impossible. Therefore, EA is preferred since it can twitch good random individuals by applying genetic operators to quickly improve their characteristics.

Since the automatic generation approach typically requires the evaluation of hundreds of thousands of functional patterns, the speed of the evaluator (as well as its accuracy) is crucial for the effectiveness of the whole approach.

The proposed idea for power estimation is summarized in Figure 2. We consider a bijective function where each element of the subset SA corresponds to an element in WSA, TA and PLA subsets when applying the functions f(SA), f'(SA) and f"(SA) respectively. However, it is difficult to discover these functions manually due the large quantity of data that must be analyzed. So, in order reduce complexity to develop these functions, we exploit a learning approach which matches SA to power consumption: initially the SA and TA subsets are elaborated from several functional patterns. The first is elaborated just by counting the SA using a simple ad-hoc program while in the second the power consumption is evaluated using a commercial tool. By integrating suitable learning mechanisms, that is a feed-forward neural network, the tool is able to understand the power evaluator behavior and provide an f'(SA) which matches SA to TA subsets and thus correctly estimating power consumption. We resort to mimetic learning algorithms because it is able to estimate power also for unknown patterns, whereas curve fitting algorithms are able to fit known data from one subset to another.

In order to construct the fast power estimator, we propose to perform some preprocessing of the CPU activity data by performing the following tasks:

(1) The SA evaluation takes into account both effective transitions and glitches occurring within each clock-cycle

(2) The NSA is computed by weighting glitches and effective transitions. We define k_1 the weight for glitches

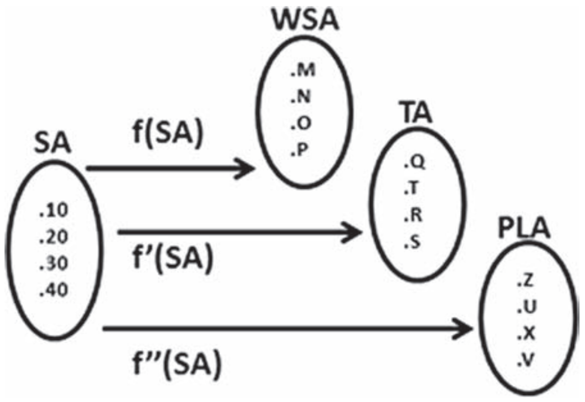

Fig. 2. Matching SA pattern to WSA, TA and PLA power consumption.

and k_2 the weight for valid transitions. By choosing the correct weights the correlation among NSA and power consumption improves, especially at the peaks.

Moreover, we propose the adoption of a fast power estimator, corresponding to a feed-forward neural network (FFNN). As discussed in Section 2, each gate type has its own properties[7] and usages[8] that impact accordingly on power consumption.[7] The FFNN is able to mimic the power evaluation. Clearly, the quality of the estimation by the FFNN depends on its appropriate training that is better detail in Subsection 3.1.

To summarize, the proposed approach to increase functional peak power is composed of two parts:

(1) Training strategy: correctly trains the FFNN for mimicking a commercial power evaluator and provide correct power estimation

(2) Increasing functional peak power: exploits the EA and the trained FFNN to automatically generate functional programs containing the maximum peaks of power.

3.1. FFNN Training Strategy

In this subsection we initially discuss analysis of SA and NSA within the clock cycle, explaining that the latter is better to be used as input for the FFNN-based fast power evaluator. Then we describe the FFNN and how it can be used for estimating power consumption mimicking a commercial power evaluator. Finally, we propose the training methodology which is able to correctly train the FFNN to effectively estimate power consumption. The proposed training methodology is composed of two phases that are iterated over time to achieve better results:

(1) Generation phase: it creates functional patterns effective for training the FFNN

(2) Training phase: it adapts the FFNN to correctly match NSA to its real power consumption and thus perform suitable power estimation.

We will now better detail the key points in the above procedure.

3.1.1. Sum SA Within the Clock Cycle

In Ref. [4], the proposed approach is to match peak activities at gate-level SA to TA power consumption of CPU cores. The activities were analyzed at every 2 hundredth of the clock period, that is, per event. However, gate-level simulation assumes zero delay, meaning that gate and interconnect delays are not considered, like it is in TA and PLA power evaluations. In this way, it is extremely difficult to match peak activities. On the other hand, the activities within the clock cycle should be rather similar since the working models must respect the clock period. Therefore, we propose to sum gate-level SA, thus getting a metric that generally matches TA peak power within the clock cycle.

3.1.2. The NSA Value

The power consumption associated to a glitch is much smaller than the one of a valid transition, and sometimes considered negligible.[9] However, glitches may impact on power consumption as they are more frequent than valid transitions. Such assumption is particularly true when analyzing a CPU containing thousands of gates where the occurrence of glitches is much more frequent than the number of valid transitions. Though, a single glitch may not have a huge impact on power, but many of them may impact more than the sum of the valid transitions. Therefore, we propose to weight glitches and valid transitions in such a way to improve the correlation index among estimations based on SA at logic level and real power behavior. We propose the adoption of a new switching activity (NSA) value where glitches are multiplied by a constant k_1 and valid transitions by another constant k_2. Formula 2 shows how to compute the NSA value at clock-cycle i.

$$NSA_i = \sum_{g=1}^{G}(k_1 GL_g + k_2 VT_g) \tag{2}$$

where GL_g and VT_g are the number of glitches and valid transitions of gate g, respectively, while k_1 and k_2 are the multiplication constants of glitches and valid transitions, respectively. G is the total number of gate outputs where glitches and valid transitions are observed. Moreover, since a single glitch is less powerful than a single valid transition (as discussed in Ref. [9]), the following relationship must hold:

$$k_2 > k_1 \tag{3}$$

The k_1 and k_2 weighting coefficients are constant for the whole circuit and we do not estimate them for every single gate.

3.1.3. FFNN for Power Estimation

The main goal of artificial intelligence (AI) is to get knowledge from known input and output patterns. The ability of a computer system to match input to output patterns in an intelligent manner is called *machine learning*. We propose a fast power evaluator based on a feed-forward neural network (FFNN). A FFNN is an AI-based tool corresponding to a directed graph with three layers: input, hidden and output.

Each layer has its own artificial neurons that sum the products of its inputs by weights and feeds the final result to the successive layer. In Figure 3, a FFNN is drawn and formula 4 describes the computation of the output value y of a single neuron k. Originally, FFNNs were introduced for pattern recognition. An input pattern is fed at the input layer and the FFNN computes an estimated value characterizing the recognition or not for that input pattern.

We propose to use the FFNN as power estimator by injecting at the input the NSA values for each gate type to compute an estimation of power consumption. The output y_k of the k-th neuron is given by the formula

$$Y_k = \varphi(\sum_{j=0}^{m} W_{kj} x_j \tag{4}$$

where φ is the transfer function of the neuron; considering a neuron with m inputs, x_o through x_m are the input signals and w_o through w_m are the weights. This neuron model is not the most used but it is the best suited for our approach since NSA inputs must be correctly weighted resulting in an approximate but not strictly accurate value of power consumption for the specific gate type. It is also a more generic neuron model, and using it in an FFNN allows quickly computing the correct weights, thus reducing the training time.

The idea of using a FFNN to estimate power values comes from the concept of power estimation methods provided by Refs. [7] and [8]. In the former work, the authors propose to estimate capacitance and power models for each type of CMOS gate with an error of maximum 10%. Also, they verify that gates with higher capacitances present also higher power consumption. The latter work reinforces the idea of Ref. [7] and also proposes that power estimation must be performed on a block basis. The authors estimate the circuit capacitance by adding the combinational

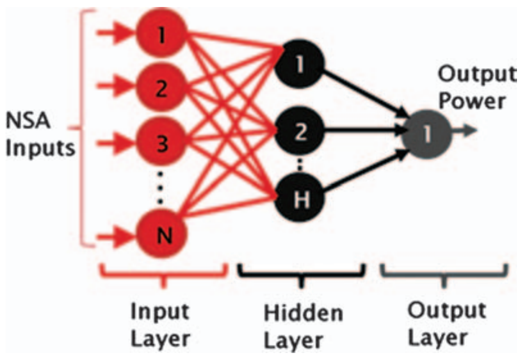

Fig. 3. Artificial intelligence feed-forward neural network.

and sequential capacitances. Moreover, the average fan in, fan out and capacitance values are used to compute the power consumption. Assuming the above mentioned concepts for power estimation, we propose to estimate power by assigning weights for each type of gate. The FFNN neuron weights are multiplied by the sum of the NSA for each gate type and the output result is the power estimation, almost accurate to the power estimation presented in formula 5 in respect to the example in Figure 4.

In Figure 4 a circuit containing combinational and sequential elements is shown. There are 18 gates and 6 gate types: flip-flop, NAND, AND, OR, NOR, and inverter. The power estimation at every clock cycle k for this circuit is shown in Formula 5.

$$P_k = \sum N_{kf} w_f + \sum N_{ki} W_i + \sum N_{kn} W_n + \sum N_{ko} W_o + \sum N_{kno} W_{no} + \sum N_{ka} W_a \quad (5)$$

where N_{kz} is the NSA for the gate type z at the instant k, and w is the weight for each gate type (f corresponds to flip-flop, i to inverters, n to NANDs, o to ORs, no to NORs, and a to ANDS).

The trained FFNN uses the correct weights for each gate type, such that they assume the average characteristics of capacitance, fan-out, wiring resistance and other characteristics impacting on the power consumption. So, the power estimation provided by the FFNN and corresponding to an approximation of formula 5 must be very similar to the power evaluation provided by the commercial evaluation tool.

In order to assign the weights, it is necessary to train the FFNN; this means that the weights are adjusted to provide the correct power consumption estimation. This is performed by using a back-propagation training algorithm. This is an iterative training algorithm which initially assigns random weights to the FFNN and computes the error values (i.e., the difference between the computed and the desired output value) given a known set of input values. The back-propagation algorithm propagates back the error to the input nodes adjusting the FFNN weights, such that providing known input values to the FFNN will make it converge to the desired output ones. The iteration stops when a sufficient Mean Squared Error value is satisfied, that is when the Mean Squared Error between the difference of the computed output values and the desired output values are under a given threshold defining the FFNN's precision.

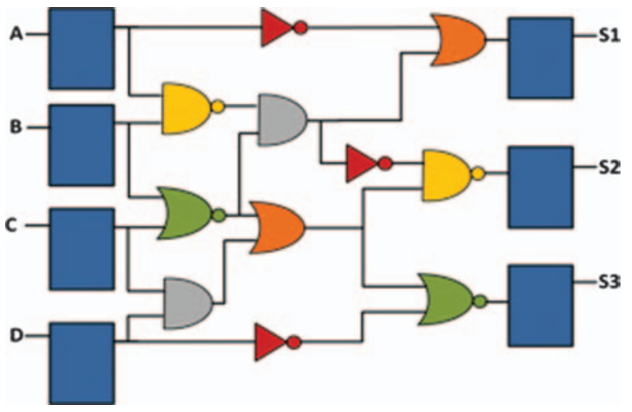

Fig. 4. A sample circuit with different gate types.

3.1.4. *FFNN Training Methodology*

Using the FFNN for estimating power does not require too much effort. However, the difficulty in its development is to train the FFNN, i.e., assigning proper weights for the neurons of every layer. In addition, ideal functional patterns must be provided in such a way to cover most of the functional pattern space set. Hence, we propose a 2-phase training methodology: generation and training. Figure 5 depicts both phases in a cooperation work to generate the correct functional patterns and to adapt the weights of the FFNN.

The first phase (generation phase) employs an evolutionary algorithm to generate functional patterns for the target circuit. The automatic functional pattern generation follows the strategy proposed by Ref. [5] and uses a framework similar to that of Figure 1. This phase is iterated for several generations until enough useful individuals have been produced. Useful individuals are corner case patterns not covered by the FFNN. Hence, these patterns are used to adapt the FFNN in the training phase and consequently, improving the power estimation.

In Figure 6 an evolution curve related to the output of the EA is presented. The EA evaluates a population of individuals and ranks the best and worst ones along the

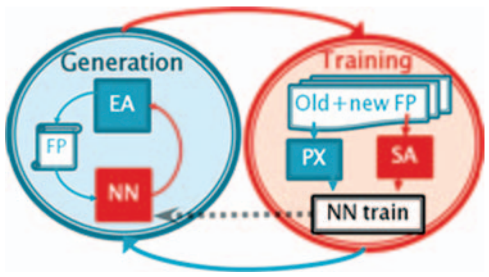

Fig. 5. 2-phase strategy to train the FFNN.

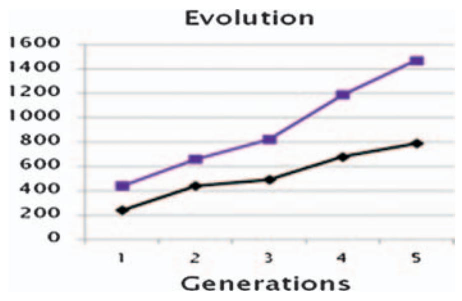

Fig. 6. Evolution curve to extract individuals.

generations, whose typical behavior is shown in the figure. Ideally, it is necessary to generate many individuals to discover corner cases not covered by the FFNN as explained in the previous paragraph.

In the second phase, power evaluation is performed on the functional patterns extracted from the evolution curve. Also, the NSA values per gate type of these individuals are extracted and matched to the evaluated peak power. Then, a back-propagation algorithm for training the FFNN is employed to improve weights and consequently the power estimation.

Also, during the training phase we remove NSA which has already been logged to speed-up the training process. Usually, repeated sequences of assembly code throughout the individuals produce the same NSA, and therefore they should be excluded from the training data set.

3.2. Increasing Functional Peak Power Using the Trained FFNN as the Fast Power Estimator

We increase the functional peak power by performing an evolution exploiting the automatic functional generation approach using the FFNN as the fast power estimator and thus speeding up the overall procedure. The final procedure is iterated for several generations until the ending point is reached i.e., when evolution reaches a stable point. Then, some individuals of this curve are sampled and power evaluated in order to verify that there is an increasing trend in the real power consumption evaluation.

In Figure 7, the final framework for increasing peak power consumption is presented. The EA generates a functional pattern that is compiled, simulated, and the NSA value per gate type is extracted to be inserted at the FFNN inputs where power estimation is performed. Then the estimated value is fed back to the EA so that it can generate better functional patterns.

4. CASE STUDY

In order to validate the proposed methodology we use an Intel 8051 CPU core synthesized in a 65 nm technology. The tool implementing the EA is the open source ugp3

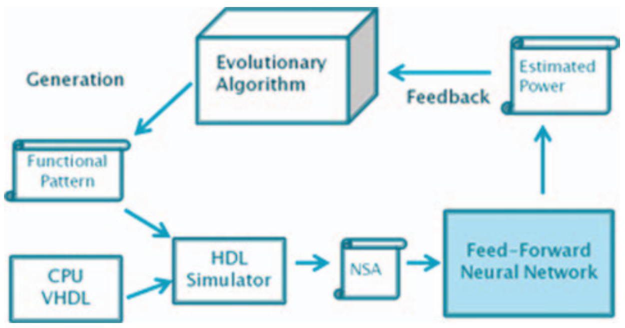

Fig. 7. Final framework including the FFNN as the fast power estimator.

software [ugp3sourceforge], which was designed to automatically generate assembly programs for a desired CPU core under a set of constraints. The tool for simulating the hardware to extract the circuit activity is Modelsim 6.3b. The tool used to evaluate power consumption during the training phases is Synopsys Primepower. Lastly, the tool evaluating electrical signals is the Nanosim Synopsys tool suite, which allows precise measurement of the power consumption of functional patterns under a specific technology library. The neural network was trained using the Matlab NN toolbox.

4.1. Intel 8051

The selected case study is the Intel microcontroller MCS-51(MC8051) synthesized in 65 nm industrial technology. The 8051 core represents a classical Harvard non-pipelined CISC architecture, with 8-bit ALU and 8-bit registers.

Table II gives the details of the microcontroller synthesis in terms of number of primary inputs/outputs, flip flops, logical gates, and the number of gate types used in the circuit.

4.2. Glitch and Valid Transition Constant Assignment for NSA Computation

Several functional patterns with many different characteristics were analyzed. Glitches and valid transitions were multiplied by several k_1 and k_2 values. Then correlation indexes were computed among the NSA and power consumption. The best value found for k_1 is 0.1 and for k_2 is 0.9. In order to discover these values, a divide and conquer algorithm was employed. Initially, k_1 and k_2 were assigned the value of 0.5. Then, the value of $k_1/2$ was added to k_2 and subtracted from k_1, in fact increasing the correlation index. If the correlation index increases, then the values of k_1 and k_2 are updated, otherwise they assume their previous values. The correlation index considered was the average value of 20 programs, and consumed less than 1 hour of computer effort to compute the values of k_1 and k_2.

4.3. FFNN and Parameters

The FFNNs used in the proposed approach were implemented using the Matlab 2008b NN toolbox. Some training parameters were defined that allowed developing the correct FFNN:

(a) Number of input nodes
(b) Number of output nodes

Table II. 8051 characteristics.

Primary inputs	65
Primary outputs	94
FFs	578
Gates	4, 246
Gate types	97

(c) Minimum and maximum values for each input
(d) Number of neurons and layers implementing the network
(e) Neuron transfer function (φ) for each layer
(f) Training type.

The FFNN for the 8051 CPU core for power estimation contains three layers: input layer, hidden layer and output layer. The input layer is composed of 97 input neurons with a hyperbolic tangent sigmoid transfer function. The NSA values per gate type are inserted in these neurons. The hidden layer is composed of 3 hidden neurons with a linear transfer function. The output layer has just one neuron with a linear transfer function; it sums altogether the intermediate values provided by the hidden layer and outputs the estimated power value. The interval of allowed NSA values in the input neurons is from 0 to 200 and the training type is back propagation. Also, the Mean Squared Error (MSE) training precision was stipulated as 0.005 Watts. This value could have been lower but training time would also increase. The result of the training is a FFNN which is able to estimate the power consumption of the case study circuit following the mechanism of formula 5.

5. RESULTS

This section draws all the results we gathered to quantitatively characterize the proposed approach. The experiments were carried out on a 12 core Intel Xeon @ 2.67 GHz and 24 GB of RAM. We initially show the correlation index evolution and trends obtained by applying the 2-phase strategy for training the gate type weights in the FFNN. Then, we compare peak power with estimated peak power values to verify that the approach is feasible and that it is able to provide a certain gain in time. Finally, we show the results in terms of performance and comparison measures using the proposed methodology containing the FFNN and the old methodology containing a commercial tool for evaluating power consumption described in Ref. [11].

5.1. Training Trend

In Table III, correlation index results for the Intel 8051 CPU core are presented. We have analyzed 11 relevant programs and evaluated the SA, NSA, FFNN estimated power and TA power consumption. These programs were obtained by using the automatic generation approach in Ref. [11] that uses a commercial power evaluator. At the end of each generation the best pattern is extracted, and when peak power improvement over a given number of generations becomes lower than a given threshold, the evolution is stopped. In this way we obtained 11 programs coming from an evolution using a commercial tool. They have a wide range of peak power, ranging from a small peak to a high one, and each of the selected individuals, best describes the 1,000 programs of each generation. Lastly, we computed the correlation indexes among TA power consumption and the abovementioned metrics. In the table, we can verify that the correlation values increases from left to right, and the use of the FFNN improves the correlation measures at the peaks, thus justifying its use for estimating the power consumption.

Table III. Correlation indexes comparison.

Case	SA versus Power	NSA versus Power	NN versus Power
Best	0.8533	0.9812	0.9943
Worst	0.8316	0.9333	0.9752
Average	0.8439	0.9582	0.9858

Figure 8 shows the correlation index evolution for the Intel 8051 CPU core by applying the 2-phase training procedure. We have iterated 8 times from generation phase to training phase and vice-versa. As we can see from the graph, the initial trained neural network has a correlation index ranging from 0.8 to over 0.9. In the literature, it is said that these values are strongly correlated, but in fact they are insufficient when we try to match peak values, and therefore still not useful for the final generation phase.

Also, it is possible to validate the correctness of the FFNN by computing the correlation index among the estimated value and the power consumption evaluated on a commercial tool of several functional patterns, as shown in Figure 8. As we increase the number of training steps, the correlation indexes and their ranges become even more strongly correlated. The final correlation indexes obtained are presented in Table III. We have noticed that not only the overall correlation index has increased but also that the test program generation procedure was able to find higher power consumption peaks.

In addition, we have spotted some misleading results, because the trained neural network provides an approximation of the desired value which can be over or under estimated defined by an error threshold computed by the training methodology. However, we have also verified that the estimated peak power in about 95% of the cases matches the actual peak power.

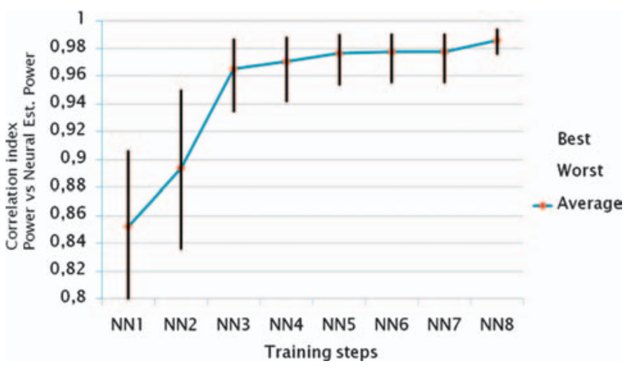

Fig. 8. 8051 Training trend.

Figure 9 shows the values of SA, NSA, FFNN estimation and power evaluations for 30 clock cycles for a sample functional pattern for the 8051. This functional pattern was not used for training the FFNN. The correlation indexes are reported in Table III for the best case. In Figure 9, the graphs have the resemblance with power consumption in an increasing order from SA to NSA and FFNN.

5.2. The Computed Gate Type Weights

In Table IV the weights for each gate type computed by the back-propagation training method of the FFNN are reported. In the circuit there are 4,246 gates and the most used ones are reported. In the case study circuit there are 9 gate types responsible for 2,340 gates, whose weights are reported in the table. Also, their maximum capacitance values extracted from the spice library, average fan out extracted from the circuit and their number are reported.

5.3. Increasing Functional Peak Power Evolution

In Figure 10, the evolution of functional peak power consumption applied to the Intel 8051 CPU core is reported. The constant ascending curve (red) is the estimated peak power value computed by the FFNN-based fast power estimator. In the figure, we have extracted 10 functional patterns characterizing the evolution curve.

In blue, the irregular ascending curve is the functional patterns' actual power value obtained using a TA power evaluator. There are two functional patterns in which disparity is high: 3 and 7. In the first case the FFNN has under estimated the power value while on the latter it over estimated it. The reason why FFNN misestimated these 2 values is because they provide corner cases which have not been used during training. This could have been fixed if the FFNN had been trained with better precision and used more sample patterns, although training time would increase significantly. Nonetheless, the evolution was not compromised by these misestimated values. As we can observe, the TA power evaluation continues to increase irregularly but satisfactory to achieve our goal.

In Figure 10, only the first best individuals of 10 generations are shown, this is because the other successive 10 generations did not present improvements to this value, and therefore the approach was stopped because it arrived on a steady state.

5.4. Performance

This subsection describes the CPU requirements of the overall proposed approach. In Table V, quantitative values

Table IV. Gate type characteristics and computed weights.

Gate type	Number	Max capacitance (pF)	Average fan out	Computed weight
D Type FF	429	0.16	6.8	0.0127
Inverter	404	0.035	2.41	0.0011
2-Input NOR	358	0.035	3.86	0.0053
2-Input NAND	349	0.053	3.72	0.0034
OR-AND inverted × 3	284	0.035	1.15	0.0043
OR-AND inverted × 2	185	0.035	1.01	0.0029
AND-OR inverted × 2	130	0.035	1.13	0.0009
AND-OR inverted	105	0.035	1.38	0.0005
4 Input MUX	99	0.07	1.22	0.0095

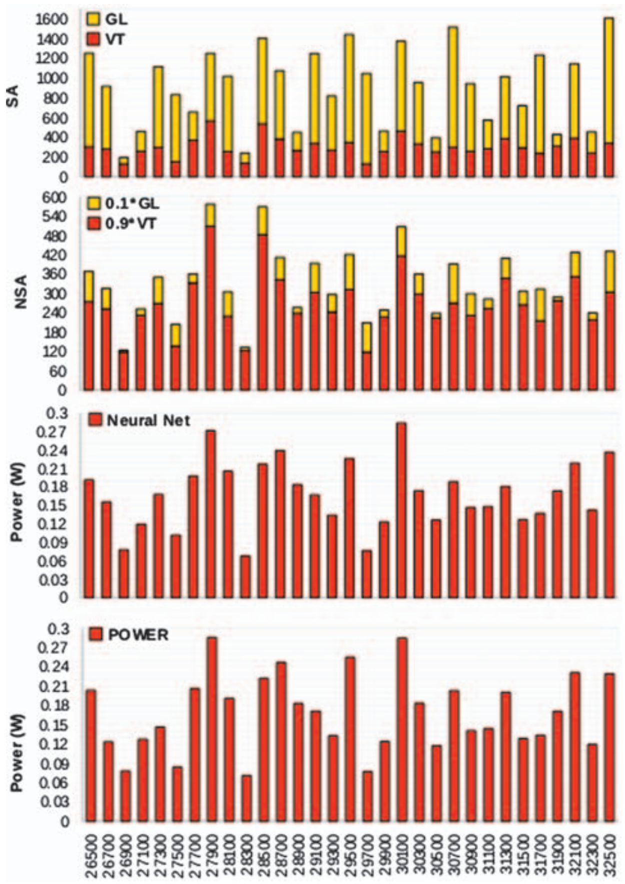

Fig. 9. SA, NSA, FFNN power estimation and power evaluation of a sample program.

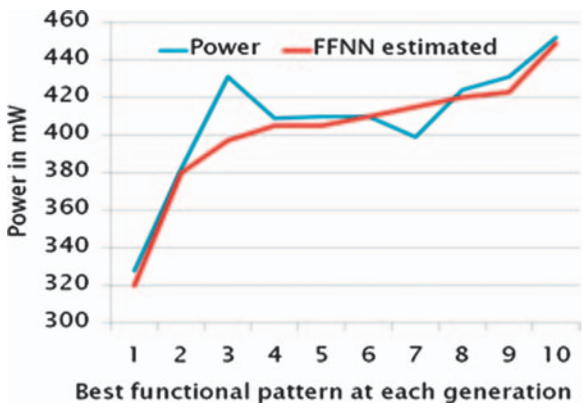

Fig. 10. Peak power evolution.

Table V. Time consumption of each tool.

Tool	Time
Gate level simulation	90 s
Ad-hoc C NSA extraction	120 s
Power evaluation	400 s
FFNN back-propagation training	1, 800 s
Feed-forward neural network	20 s

Table VI. CPU time requirements.

CPU	Old method	Proposed approach
8051	430,000 s (∼5 days)	190,000 s (∼2 days)

for each tool used to generate, train and evaluate functional patterns in terms of time are reported.

For the 2-phase training strategy, we have to separate the generation and training phase time consumption. In the first phase it is necessary to generate around 2,000 patterns, use the gate-level simulation, an ad-hoc C program for NSA extraction and for the usage of the FFNN. The training phase uses the same tools used in the generation phase; additionally, the power evaluation and back-propagation training for the FFNN have to be executed. The number of 2,000 patterns was obtained after achieving a 0.98 average correlation index between the FFNN estimated power and the commercial power evaluator.

In Table VI, the final CPU time consumption of the proposed approach and the old approach are reported. The latter uses the automatic generation approach containing the EA as the program generation engine and the WSA power consumption evaluator that is provided in Ref. [11]. The time required to train the FFNN was around 2 days and was not included in the generation time shown in Table VI.

The power evaluation using the trained FFNN requires 140 seconds which is around 60% faster than the commercial power evaluator used in the old approach.[11] The methodology applied to PLA power estimation could be much more time efficient, being twice as slow compared to RTL SA analysis but having an accurate measure (higher than 0.95 correlation index) compared to PLA power evaluation commercial tool.

6. CONCLUSIONS

In this paper we have presented a methodology that automatically identifies functional test programs with maximal CPU core's peak power consumption; these programs could be useful to correctly define test power limits. We have performed an analysis of the switching activity and power consumption correlation measures. Then, we used a mix of neural networks and evolutionary computation to generate functional patterns dataset to compute the correct weights for each gate type. After applying the training phase and iterating it over several training steps, we use the trained FFNN as a fast power estimator to be used in the automatic functional generation approach. The methodology has then been used for identifying functional test programs with maximal functional peak power consumption. We have validated the proposed strategy on an Intel 8051 CPU core synthesized in a 65 nm industrial technology.

The value of the correlation index computed during the 2-phase training strategy increases with the number of training steps. However, increasing too much the number of steps also makes the strategy and training slower. The results in fact show an evolution trend on the power consumption, although some irregularities have been observed. Nevertheless, the evolution trend is also observed in the power consumption evaluation. Lastly, the proposed approach was able to reduce the time for power hungry test program generation from 5 to 2 days.

The proposed methodology is currently being experimented on the OR1200 CPU core, which is an open source architecture with a 4-stage pipeline: it has a more complex architecture (including caches) and for such reason the proposed approach needs some small adjustments to be applied: however, preliminary results seem to confirm the validity of the proposed approach.

References

1. P. Girard, N. Nicolici, and X. Wen (eds.), Power-Aware Testing and Test Strategies for Low Power Devices, Springer, ISBN 978-1-4419-0927-5 (**2009**).
2. J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger, A case study of IR-Drop in structured at-speed testing, *IEEE Int. Test Conf.* (**2003**), pp. 1098–1104.
3. M. Valka, et al., A functional power evaluation flow for defining test power limits during at-speed delay testing, *IEEE European Test Symposium* (**2011**), pp. 153–158.
4. P. Bernardi, et al., Peak power estimation: A case study on CPU cores. *IEEE Asian Test Symposium* (**2012**), pp. 167–172.
5. F. Corno, et al., Automatic test generation for verifying microprocessors. *IEEE Potentials* 34 (**2005**).
6. L. Glasser and D. Dobberpuhl, 'The design and analysis of VLSI circuits. Reading, Addison-Wesley, MA (**1985**).
7. S. R. Vemuru, et al., Short-circuit power dissipation estimation for CMOS logic gates. *IEEE Transactions on Circuits and Systems I* 762 (**1994**).
8. D. Liu and C. Svensson, Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid-State Circuits* 29, 663 (**1994**).
9. D. Rabe and W. Nebel, Short-circuit power consumption of glitches, *International Symposium on Low Power Electronics and Design* (**1996**), pp. 125–128.
10. M. de Carvalho, et al., An enhanced strategy for functional stress pattern generation for system-on-chip reliability characterization, *IEEE International Workshop on Microprocessor Test and Verification* (**2010**), pp. 29–34.
11. A. Calimera, et al., Generating power-hungry test programs for power-aware validation of pipelined processors, *23rd Symposium on Integrated Circuits and System Design* (**2010**), pp. 61–66.
12. F. Najm, A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 446 (**1994**).