Automatic generation of power state machines through dynamic mining of temporal assertions

Alessandro Danese
Department of Computer Science
University of Verona, Italy
Email: alessandro.danese@univr.it

Graziano Pravadelli
Department of Computer Science
University of Verona, Italy
Email: graziano.pravadelli@univr.it

Ivan Zandonà
Department of Computer Science
University of Verona, Italy
Email: ivan.zandona@studenti.univr.it

***Abstract*—Several papers propose approaches based on power state machines (PSMs) for modelling and simulating the power consumption of system-on-chips (SoCs). However, while they focus on the use of PSMs as the underlying formalism for implementing dynamic power management techniques, they generally do not deal with the basic problem of generating PSMs. In most of these papers, PSMs just exist, in some cases they are manually defined, and only a few approaches give a hint of semi-automatic generation, but no fully-automatic approach exists in the literature. Indeed, without an automatic procedure, an accurate power characterization of complex SoCs by using PSMs is almost impossible. Thus, in this paper, first a methodology for the automatic generation of PSMs is proposed, and then, a statistical approach based on a Hidden Markov Model is presented for their simulation. The core of the approach is based on a mining procedure whose role consists of extracting temporal assertions describing the functional behaviours of the IP, which are then automatically mapped on states of the PSMs and characterized from the energy consumption point of view.**

I. Introduction

Power state machines (PSMs) are a well-known formalism to model and simulate the time-based energy consumption of IP cores for early virtual prototyping of system-on-chips (SoCs) [1], [2], [3], [4], [5], [6]. In this context, the PSMs of IPs included in the model of the target SoC are controlled by a power manager to allow the exploration of different dynamic power management solutions [7].

In a PSM, the energy behaviours of the IP are associated to a set of states. In its simplest form, the power consumption of each PSM state is modelled as a constant value derived by a designer estimate or from the IP's data sheet [1], [2]. When a higher level of accuracy is desired and more precise information about the IP's energy behaviours are available, the power consumption of a PSM state is computed by a more complex function. For example, in [3], [5], such a function is derived by means of a calibration process based on linear regression, which exploits, as golden reference, power traces generated at gate level, where the IP's power consumption can be more precisely estimated. However, despite of the wide adoption of PSMs, in the most of the works either the presence of PSMs is assumed [1], [2], [7] or they are manually defined starting from a more or less precise knowledge of the functional blocks composing the target IP [4], [6]. Only in a few cases, automatic approaches are proposed to create the association between PSM states and their power consumptions, but the identification of such states remains manual [3], [5]. Unfortunately, such a manual definition reveals to be inappropriate for the power characterization of complex designs leading to the generation of a less-accurate simplified model.

To allow a more precise definition of PSMs, this paper presents a fully-automatic methodology for their generation and an efficient statistical approach for their simulation. The proposed methodology does not require to instrument the functional model of the target component and it can be applied even in case of black-box IPs. It only assumes the availability of two kinds of training traces: a set of functional traces exposing the IP's behaviours, and a corresponding set of power traces over time that characterize the dynamic part of the IP's energy consumption depending on its switching activity. The way training traces are generated is independent from the proposed methodology, however their quality highly impacts on the accuracy of the final PSMs. In particular, if the functional traces were unable to cover all the functional behaviours of the IP, the PSMs would be incomplete, thus leading to a wrong estimation of the power consumption, each time the PSMs encounter an unknown behaviour during their simulation. On the other side, the use of not-accurate power traces for characterizing the energy consumption associated to the states of the PSMs will negatively reflect on their precision. To rely on a high-quality set of training traces, functional traces are generally obtained by simulating the IPs with the testbenches used for their system-level functional verification, while accurate power traces can be generated by running a gate-level power simulator, like Synopsys PrimeTime PX [8], on the same functional traces.

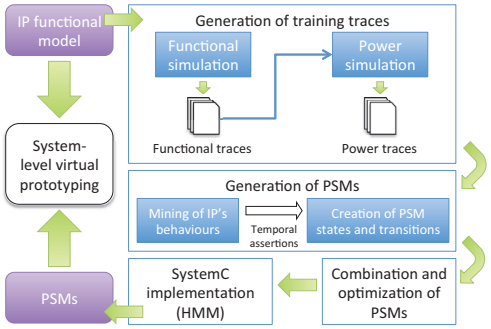

Fig. 1. Methodology overview.

Fig. 1 shows an overview of the methodology. It starts by dynamically mining temporal assertions from the functional traces. Such assertions are logic formulas that capture the functional behaviours of the IP over time [9]. From them, the states and the transitions of a corresponding set of PSMs are generated, under the hypothesis that different functional behaviours may expose different power consumptions. Then, each state is associated to a power consumption by exploiting the reference power traces. The obtained PSMs are then merged and optimized to generate a compact set of more accurate PSMs. These PSMs are finally implemented into a SystemC module, in the form of a Hidden Markov Model (HMM) [10], to allow their efficient and effective simulation concurrently with the simulation of the IP functional model.

The rest of the paper is organized as follows. Section II introduces preliminary definitions. Section III presents the mining approach for the generation of the initial set of PSMs. Section IV describes how PSMs can be combined and optimized to obtain a more compact and accurate set of

This work has been partially supported by the EU project CONTREX (FP7-2013-ICT-10-611146).

PSMs. Section V deals with the HMM-based simulation of the combined PSMs. Finally, Section VI and Section VII conclude the paper with experimental results and final remarks.

II. Preliminaries

The following definitions are necessary to formalize the methodology proposed in the rest of the paper.

Definition 1. *An **atomic proposition** is a logic formula that does not contain logic connectives. A **proposition** is a composition of atomic propositions through logic connectives. A **temporal assertion** is a composition of propositions through temporal operators.*

The proposed methodology mines assertions based on the temporal operators $next$ and $until$ belonging to the linear time logic (LTL). According to the semantic of LTL, if p and q are LTL assertions, then $next\ p$ is true if p holds at the next instant, and $p\ until\ q$ is true if p remains true until q becomes true.

Definition 2. *Given a finite sequence of simulation instants $\langle t_1, \ldots, t_n \rangle$, the set of variables V representing primary inputs (PIs) and primary outputs (POs) of a model M, and a set of propositions $Prop$ predicating over V, a **functional trace** of M is a finite sequence $\Phi = \langle \phi_1, \ldots, \phi_n \rangle$, where $\phi_i = eval(V, t_i)$ is the evaluation of variables in V at simulation instant t_i; a **proposition trace** is a finite sequence $\Gamma = \langle \gamma_1, \ldots, \gamma_n \rangle$, where $\gamma_i \in Prop$ is the proposition that holds at simulation instant t_i; and a **power trace** is a finite sequence $\Delta = \langle \delta_1, \ldots \delta_n \rangle$, where δ_i is the dynamic energy consumption of M at simulation instant t_i according to the formula $\delta_i = \frac{1}{2} V_{dd}^2 f C \cdot \alpha(t_i)$, being C the total switched capacitance, V_{dd} the supply voltage, f the clock frequency, and $\alpha(t_i)$ the switching activities of M at time t_i.*

Definition 3. *A **power state machine** is defined as a 7-tuple PSM=$\langle I, O, S, S_0, E, \lambda, \omega \rangle$, where I is the input alphabet, O is the output alphabet, S is a set of states, $S_0 \subseteq S$ is the set of initial states, E is a set of enabling functions $e : I \to \{0, 1\}$, $\lambda : S \times E \to S$ is the transition function, $\omega : S \to O$ is the output function that produces the power consumption.*

Figure 2 shows an example of a PSM composed of three power states that characterize the power consumption of the IP when it is turned off, idle and operating with three different constant values (0mW, 15mW and 100mW). Input symbols are associated with the values that can be assumed by the primary inputs of the IP, (i.e, *on*, *ready* and *start*). Enabling functions are represented as guards associated to edges.

III. PSM generation

A. Mining of the IP's functional behaviours

Given a set of functional traces, the behaviours of the corresponding IP is captured through a set of proposition traces that are automatically generated by a mining procedure. Among the several approaches existing in the literature, we adopt the mining technique described in [9]. It works in two phases. In the first phase, for each functional trace Φ, the procedure extracts a set of atomic propositions, which hold frequently on Φ, predicating over the PIs and POs of the IP. The atomic propositions represent relations between PIs and POs of the IP that hold in a set of subtraces of Φ. The output of this phase is represented by a matrix m, where the generic element

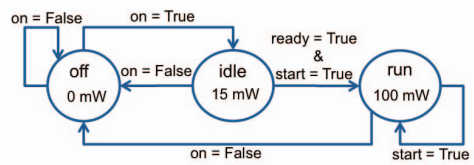

Fig. 2. An example of a power state machine.

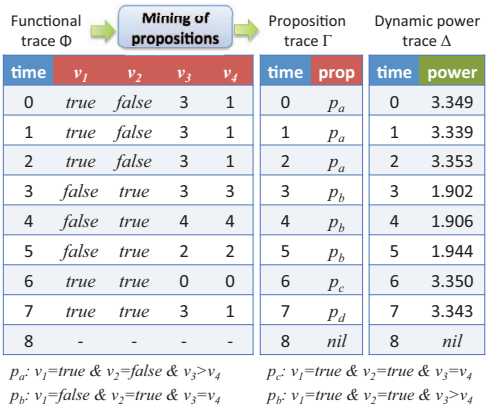

Fig. 3. A functional trace and its proposition and power traces.

in position $[i, j]$ reports the truth value of the j-th atomic proposition at the i-th time instant of the functional trace. In the second phase, the atomic propositions are combined into a set $Prop$ of propositions, such that in each simulation instant of Φ one and only one of propositions belonging to $Prop$ holds. In particular, a composition procedure generates one proposition from each row of the matrix m by composing in an AND formula all atomic propositions that are marked as true. Finally, the proposition trace is obtained by identifying which proposition is true in each simulation instant of the functional trace. The extracted propositions are used to generate temporal assertions that capture the functional behaviours of the IP exposed by the functional trace. Such behaviours are then mapped on states of the PSM as described in Section III-B.

An example of proposition trace generation is reported in the left side of Fig. 3, Given, the functional trace Φ, atomic propositions that frequently hold on it are, for instance, $v_1 = true$, $v_2 = false$, $v_3 > v_4$, etc. After their extraction, the mining procedure generates the proposition trace composed of propositions p_a, p_b, p_c and p_d that hold, respectively, in the intervals $[0, 2]$, $[3, 5]$, $[5, 6]$ and $[6, 7]$.

B. Generation of states and transitions

The assumption under the use of a PSM to model the dynamic power of an IP is that there is a correspondence between a specific functional behaviour (characterized by a switching activity) and its energy consumption. Thus, to generate a PSM, first we mine the IP functional behaviours and then we associate a power consumption to each of them.

Before presenting technical details, let us describe the intuitive idea underlying the proposed approach. By observing a time window between two simulation instants we can observe that the functional behaviours of an IP follow two temporal patterns, namely, *next* and *until*. These two patterns generally alternate when the IP is operating, such that we can observe several consecutive instants where the IP remains in a (sequence of) stable condition(s) from the functional point of view (*until* pattern), followed by an arbitrarily-long sequence of jumps among different states (*next* pattern), before reaching a new stable condition. Moving from one behavioural pattern to another, the energy consumption varies as well. Thus, the basic idea for the automatic generation of a PSM consists of capturing the sequence of *until* and *next* patterns exposed by the IP during its activity and associating to them the corresponding energy consumption.

More formally, given s_i and s_j characterizing two functional states of an IP, the meaning of the *next* and *until* patterns can be described as follows:

- the *next* pattern $s_i X s_j$ corresponds to the temporal assertion $(state = s_i) \to next(state = s_j)$ specifying that after s_i, at the next instant, the IP moves to s_j;

• the *until* pattern $s_i U s_j$ corresponds to the temporal assertion $(state = s_i)\ until\ (state = s_j)$ specifying that s_j is preceded by a sequence of time instants where the IP stays in s_i.

According to the mining procedure described in Section III-A we can associate a functional trace with a proposition trace that formalizes the functional behaviours of the IP as a sequence of propositions holding on the different time instants. Thus, to automatically extract the functional behaviours of the IP it is sufficient to search for the *until* and *next* patterns inside the proposition trace. For example, let us consider the proposition trace Γ reported in Fig. 3. Starting with the time instant 0, Γ exposes the following behaviours: $p_a U p_b$, $p_b U p_c$, and $p_c X p_d$, respectively, in the intervals $[0, 2]$, $[3, 5]$, and $[6, 7]$. Thus, we derive that there are three functional behaviours that must be associated to three power states of the PSM.

Entering in the technical details, to automatically extract the *next* and *until*-based behaviours, and thus defining the corresponding PSM states, we defined the *PSMGenerator* procedure described in Fig. 4, and the *XU* automaton shown in left side of Fig. 5. The inputs of the procedure are a proposition trace Γ, a dynamic power trace Δ, and a reference to the *PSM* that will be created. The core of *PSMGenerator* is represented by the *XU* automaton. At the beginning, the *XU* automaton is initialized by the function *XU_initialize* (line 2) whose role consists of filling in a *FIFO* data structure f with the two propositions corresponding to instants 0 and 1 of Γ, and setting the current state of the automaton to *X*. Then, the function *XU_getAssertion* (line 5) iteratively traverses the *XU* automaton till an assertion corresponding to one of the temporal patterns *X*,*U* is identified in Γ. During the traversal, each time a transition is taken on the *XU* automaton the *FIFO* scrolls forward by one position on Γ. As soon as a temporal assertion is identified in Γ, the function *XU_getAssertion* returns the triplet $\langle p, start, stop \rangle$. The *start* and *stop* indexes capture the time interval where p holds in Γ.

For each mined temporal assertion, the following steps are then performed (lines 7-13):
1) *getPowerAttributes* is called (line 7) to collect the triplet $\langle \mu, \sigma, n \rangle$, where, $n = stop - start + 1$ is the number of time instants where the assertion holds, μ is the mean of the energy consumption values reported in the dynamic power trace Δ in the time interval $[start, stop]$, and σ is their standard deviation. In the following we will refer to the triplet $\langle \mu, \sigma, n \rangle$ with the term *power attributes*.
2) *createPowerState* is called (line 8) to create a new state of the PSM characterized by the temporal assertion *p* and by the power attributes $\langle \mu, \sigma, n \rangle$. The output function of the state is represented by the constant value μ. The new state is then added to the PSM by the function *addState* (line 9).
3) for all the new states except the first, *createTransition* is called (line 11) to create a transition t between the new state new_s and the previously extracted state $prev\_s$. The enabling function e labelling t is represented by the proposition included in element $f[1]$ of the *FIFO* when *XU_getAssertion* stops and recognizes a pattern for $prev\_s$. Finally, *addTransition* adds

```
1: procedure PSMGenerator(Γ, Δ, PSM)
2:     XU_initialize(Γ)
3:     prev_s = nil
4:     while true do
5:         ⟨p, start, stop⟩ = XU_getAssertion(Γ)
6:         if p == nil then break end if
7:         ⟨μ, σ, n⟩ = getPowerAttributes(Δ, start, stop)
8:         new_s = createPowerState(p, ⟨μ, σ, n⟩)
9:         addState(new_s, PSM)
10:        if prev_s ≠ nil then
11:            ⟨t, e⟩ = createTransition(prev_s, new_s)
12:            addTransition(⟨t, e⟩, PSM)
13:        end if
14:        prev_s = new_s
15:    end while
16: end procedure
```

Fig. 4. PSM generation.

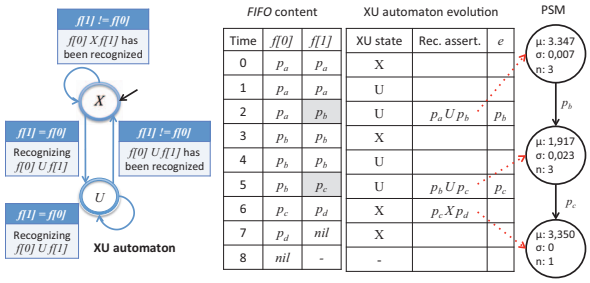

Fig. 5. The XU automaton and the exemplification of the procedure of Fig. 4.

the transition to the PSM (line 12).

To clarify how the *PSMGenerator* procedure works, the right side of Fig. 5 exemplifies its application to the proposition trace Γ and the power trace Δ reported in Fig. 3. The XU automaton initially moves from X to U because at time 0 the condition $f[1] = f[0]$ is satisfied. This means we are going to see at least two consecutive instances of the proposition p_a in Γ (at times 0 and 1), and thus we are going to recognize an *until* pattern. Then, at time 1 the *FIFO* is scrolled forward, and the automaton remains in U because $f[1] = f[0]$ is still true. Finally, at instant 2 the automaton exits U and comes back to X because $f[1] \neq f[0]$, and consequently *XU_getAssertion* recognizes the assertion $p_a U p_b$ and returns the triplet $\langle p_a U p_b, 0, 2 \rangle$ which corresponds to the first state of the PSM. The state is then populated with the mean and the standard deviation corresponding to the values of the power trace Δ in the interval [0,2]. At instant 3, the automaton moves again to U starting the capture of a new *until* pattern. The exit condition from U is reached at the instant 5 when $f[1] \neq f[0]$ with the recognition of $p_b U p_c$ in the interval $[3, 5]$. Thus a new state is created and connected to the first state with a transition whose enabling function is p_b, namely the value of $f[1]$ at time 2 when the first state based on the *until* pattern was created. In a similar way, the *PSMGenerator* procedure mines the final state corresponding to $p_c X p_d$ and it completes when nil is encountered. At the end, the PSM reported on the right of Fig. 5 is obtained.

C. Simulation of a single PSM

The PSM generated by the previous methodology is in the form of a chain of states, where each state has a unique successor and a unique predecessor. Its simulation is synchronized with the simulation of the corresponding IP by connecting primary inputs and outputs of the IP to the PSM. In this way, at each simulation instant, the values assumed by PIs and POs of the IP are passed as inputs to the PSM, which decides how to move according to the temporal assertion characterizing its current state. When the PSM is in a state s it checks its associated temporal assertion p, whose satisfiability depends on the functional behaviour captured through PIs and POs of the IP. If p follows the *until* pattern $p_a U p_b$, the PSM stays in s till p_a is true and it moves to the next state as soon as p_b becomes true. On the contrary, if p follows the *next* pattern $p_a X p_b$, the PSM moves to the next state at the next simulation instant. The enabling function of the transition outgoing from the current state s is satisfied by construction, because, in both cases, it corresponds to the exit condition represented by the activation of proposition p_b.

It is worth noting that if the PSM extracted by the procedure of Fig. 4 is stimulated by adopting a functional trace different from the one used for its generation, the power consumption estimated by the PSM may be wrong when it reaches a state characterized by an unexpected temporal assertion. This is due to the fact that the PSM exactly resembles the temporal assertions mined in the proposition trace extracted from the reference functional trace. For example, let us consider a PSM reaching a state s whose temporal assertion is $p_a U p_b$. Then,

suppose that when the PSM enter s, p_a is true for a while till p_a becomes finally false, but at that instant p_b continues to remain false as well. In this case, the PSM cannot traverse the outgoing transition of s because p_b is expected. Thus, it remains in s losing the correct synchronization with the functional trace and generating a wrong power estimation. This limitation is overcome by generating and combining together several PSMs extracted by a set of different functional traces. More the functional traces are representative of all combinations of IP behaviours, lower is the probability of loosing the synchronization between the functional trace and the power simulation. The combination and simulation of different PSMs corresponding to the same IP and their optimization is described in the next sections.

IV. Combination and Optimization of PSMs

Given a set of PSMs $\mathcal{P}$, generated for the same IP according to the procedure proposed in the previous section, we propose an automatic methodology to create a reduced and optimized set of PSMs $\mathcal{P}^{opt}$.

The first step of the methodology consists of calling the *simplify* procedure for each PSM in $\mathcal{P}$. The PSMs extracted by the *PSMGenerator* of Section III-B are in the form of a chain of states. The effect of *simplify* is to shorten such chains, if possible. Thus, for each PSM in $\mathcal{P}$, *simplify* iteratively merges into a single state, a sequence of adjacent states which are "mergeable" from the power point of view. Intuitively, two adjacent states s_i and s_{i+1} are mergeable when the means μ_i and μ_{i+1} of energy consumptions associated respectively to s_i and s_{i+1} are "similar", and their standard deviations σ_i and σ_{i+1} are "low". For now, the meaning of terms "similar" and "low" is intuitively understandable to capture the notion of mergeable states, while technical details are reported in Section IV-A.

According to Section III-B, a state s of a PSM is characterized by the two triplets $\langle p, start, stop\rangle$ and $\langle \mu, \sigma, n\rangle$. When a sequence of adjacent mergeable states $\langle s_i, \ldots, s_{i+j}\rangle$ is found, they are substituted by a new state s_{new} whose triplets $\langle p_{new}, start_{new}, stop_{new}\rangle$ and $\langle \mu_{new}, \sigma_{new}, n_{new}\rangle$ are computed as follows:

- $start_{new} = start_i$; $stop_{new} = stop_{i+j}$; and $n_{new} = n_i + n_{i+1} + \cdots + n_{i+j}$;
- $p_{new} = \{p_i; p_{i+1}; \ldots; p_{i+j}\}$, which represents that first p_i holds in the interval $[start_{new}, stop_i]$, then p_{i+1} immediately follows in the interval $[start_{i+1}, stop_{i+1}]$, and so on till p_{i+j} finally holds in the interval $[start_{i+j}, stop_{new}]$.
- μ_{new} and σ_{new} are, respectively, the mean and the standard deviation of the energy consumption values reported in the interval $[start_{new}, stop_{new}]$ of the reference power trace.

Finally, s_{new} is connected with the predecessor s_{i-1} of s_i and the successor s_{i+j+1} of s_{i+j} through the transition outgoing from s_{i-1} and ingoing to s_{i+j+1}. The procedure iteratively executes till no new mergeable state is found. Figure 6(a) graphically exemplifies the effect of $simplify$ on a sequence composed of two states.

After the application of the *simplify*, the resulting PSMs are merged into a reduced set $\mathcal{P}'$ by means of the *join* procedure. It works similarly to *simplify* by collapsing mergeable states, but in this case they are not required to be adjacent and they can belong to different PSMs. As a consequence, the triplets characterizing the new state are computed in a different way with respect to *simplify*. In particular:

- $start_{new}$ and $stop_{new}$ become two arrays whose generic element i contains, respectively, the start and stop value of the merged state s_i, while n_{new} becomes the sum of values n of the merged states;
- $p_{new} = \{p_i||p_j||\ldots||p_k\}$, which represents that each time s_{new} is entered one of the assertions characterizing the set of merged states $\{s_i, s_j, \ldots, s_k\}$ is satisfied and then, when it becomes false, s_{new} is left.
- μ_{new} and σ_{new} are, respectively, the mean and the standard deviation of the energy consumption values reported in the interval $[start_i, stop_i]$ of the reference power trace for each merged state.

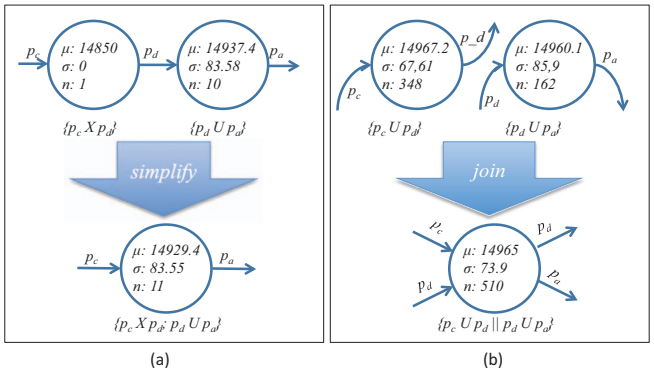

Fig. 6. Exemplification of procedures *simplify* and *join*.

Finally, s_{new} is connected with the predecessors and the successors of all merged states through the transitions, respectively outgoing from the predecessors and ingoing to the successors. The procedure iteratively executes till no new mergeable state is found. Figure 6(b) graphically exemplifies the effect of $join$ on two not-adjacent states.

At the end of the *join*, we obtain a new set $\mathcal{P}'$ of more compact PSMs, whose cardinality can be lower than the cardinality of the original set $\mathcal{P}$, if the *join* merged at least two states belonging to different PSMs of $\mathcal{P}$. It is worth noting that, in a particular case, the *join* generates a not-deterministic PSM. This happens when the *join* merges states that are characterized by the same temporal assertions and have the same enabling functions in the respective ingoing transitions and the same enabling functions in the respective outgoing transitions. In this case, when we enter such a collapsed state during simulation, we cannot deterministically decide the transition to be traversed when the state is left. The simulation of such a not-deterministic PSMs is guaranteed by exploiting an Hidden Markov Model (HMM), as described in Section V.

The final step of the methodology transforms the set of PSMs $\mathcal{P}'$ in a more accurate final set $\mathcal{P}^{opt}$ by acting on power states with a "too high" standard deviation σ. These states have a high probability of being data-dependent from the energy consumption point of view, i.e., when the IP is in one of such states the energy consumption strictly depends on the sequence of data provided to IP's primary inputs. Thus, the use of a constant, represented by the mean μ of energy consumption values, to characterize the power of such data-dependent states is inaccurate. To improve the precision of the power estimation for such a kind of states, we substitute μ in a state s with a function extracted by applying a linear regression between the Hamming distances of consecutive input values exposed in the functional trace and the corresponding values in the power trace. This substitution is applied only for states with an strong linear correlation between Hamming distances of inputs and corresponding values in the power trace, which is a necessary condition for achieving an accurate result from the linear regression [11].

A. Quantifying the mergeability of power states

The mergeability of power states is evaluated by comparing the power attributes of the target states according to three different cases.

Case 1: comparison between s_i and s_j, where $n_i = n_j = 1$. This happens when both the states are characterized by a temporal assertion respecting the *next* pattern. In this case we affirm that s_i and s_j are mergeable when $|\mu_i - \mu_j| < \varepsilon$, where ε is the tolerance fixed by the designer.
Case 2: comparison between two power states s_i and s_j, where $n_i > 1$ and $n_j > 1$. In this case the states are both characterized by the *until* pattern. To identify if it is worth merging the two states, we perform the Welch's t-test [12] on the power attributes $\langle \mu, \sigma, n \rangle$. Such a test is generally used to determine if two sets of data are significantly different from each other with an arbitrarily percentage of error. For lack of space we omit its mathematical formulation, which can be retrieved in [12].
Case 3: comparison between two power states s_i and s_j, where $n_i > 1$ and $n_j = 1$. This happens in the tentative of merging an *until*-based state with a *next*-based state. Similarly to Case 2, we use a different formulation of the t-test to see if the single sample represented by state s_j can be merged with the larger set of values represented by s_i.

V. Simulation of multiple PSMs

While the basic simulation for one single PSM has been already presented in Section III-C, the next paragraphs deal with the concurrent simulation of the complete set of PSMs associated to an IP. A method for resynchronizing the PSMs when unknown behaviours are encountered is presented too.

With respect to the case presented in Section III-C, the simulation of multiple PSMs, obtained after the application of the *simplify* and *join* procedures, has two main differences: (i) the power states can be characterized by more than one assertion, and (ii) some PSMs may be non-deterministic. Concerning the first aspect, when the PSM enters a state s characterized by a sequence of assertions $\{p_i; p_{i+1}; \ldots; p_{i+j}\}$ (as a result of *simplify*), it expects they are satisfied in a cascade fashion one after the other. Thus, first the PSM checks if p_i is satisfied (for an unbounded period of time in case of *until* pattern, or just for one time instant for a *next* pattern). Then, when p_i becomes false (in case of *until*) or simply at the next instant (in case of *next*), the PSM repeats the same analysis for p_{i+1}, and so on till p_{i+j}, when it leaves s according to the enabling function of the outgoing transition. On the contrary, if one of the assertions fails the analysis, i.e., it is not satisfied when expected, it means the PSM has reached an unknown functional behaviour. In this case, the simulation continues but the PSM state is not changed till a *resynchronization procedure* allows it jumping to a different state from which a known behaviour can be recognized. During the resynchronization period the power estimation provided by the PSM is not reliable.

When a *join* merges a set of states, the resulting state s is characterized by a set of concurrent assertions of the form $\{p_i || p_j || \ldots || p_k\}$. In this case, at least one of these assertions must be satisfied when entering s, otherwise the resynchronization procedure is called. When exactly one assertion is satisfied the simulation proceeds as in the basic case described in Section III-C and s is left by traversing the outgoing transition corresponding to the satisfied assertion. For example, in Fig. 6(b) the merged state is left by traversing the transition labelled with p_d (respectively p_a) when $p_c U p_d$ (respectively $p_d U p_a$) is satisfied. In some cases, as reported in Section IV, the *join* procedure can generate a state where the set of characterizing assertions includes two or more instances of the same assertion. When such identical assertions are satisfied in a state s, a *not-deterministic choice* must be taken to exit s. It is worth noting that a not-deterministic choice could be necessary also at the very beginning of the simulation, when we need to choose the starting state among all the initial states that can be activated. Remember that an IP is associated to a set of PSMs derived from different functional/power traces, thus we have a set of initial states.

To efficiently manage the not-deterministic choices and the resynchronization procedure, we adopted a statistical approach based on a Hidden Markov Model. HMMs are frequently used in temporal pattern recognition, thus they are well suited in our context to predict the state with the highest probability of being the correct choice in case of non-determinism or when a resynchronization is necessary. A HMM is defined as a quintuple $\langle Q, E, A, B, \pi \rangle$, where $Q = \{Q_1, \ldots, Q_m\}$ is set of hidden states; $E = \{E_1, \ldots, E_n\}$ is a set of observable events; $A = \{a_{ij}\}$ and $B = \{b_{jk}\}$ are two matrices such that their elements $a_{ij} = P[x_{t+1} = Q_i | x_t = Q_j]$ (with $1 \leq i, j \leq m$) and $b_{jk} = P[E_k | Q_j]$ (with $1 \leq j \leq m, 1 \leq k \leq n$) represent, respectively, the probability of reaching the state Q_i at time x_{t+1} starting from the state Q_j at time x_t, and the probability of observing the event E_k at state Q_j; π=$\{p_i\}$ is a vector such that its element $p_i = P[x_0 = Q_i]$ (with $1 \leq i \leq m$) represents the probability of being in state Q_i at time 0. By contextualizing the definition of a HMM to the problem of predicting the next state of a PSM, we implemented a model where Q contains the states of all the generated PSMs and E contains their characterizing assertions. Elements $\{a_{ij}\}$ and $\{b_{ij}\}$ of matrices A and B are then defined according to, respectively, the number of transitions exiting from state i to reach state j, and the number of times the same assertion j has been included (by *join* operations) into the set of assertions characterizing the state i. Finally, the value of the i-th element of the vector π is computed by counting the number of functional traces that have originated a PSM with i as its initial state. Given this formalization, during the simulation of the HMM the next state is chosen by applying the *filtering* approach, i.e., a state-of-the-art procedure to predict the distribution of the next (hidden) states according to a sequence of observations, which in our case are the functional behaviours captured by the temporal assertions mined on the proposition traces. When a wrong state s is predicted (i.e., the simulation cannot exits s because none of its characterizing assertions is satisfied when expected), the HMM reverts to the last valid state and it follows a different path by fixing to 0 the probability of reaching again the same wrong state in the matrix A. In case all transitions exiting from the current state bring to a wrong state, an unexpected behaviour is encountered. This highlights that the functional traces used for generating the PSMs where incomplete with respect the the ones used for the simulation. In this case, the simulation of the model proceeds by remaining in the last valid state till a known behaviour is finally recognized in the future instants.

VI. Experimental results

The proposed methodology has been implemented in an automatic tool that generates a SystemC model of the extracted PSMs. Its effectiveness and efficiency have been evaluated by generating the PSMs for the RTL Verilog descriptions of the IPs reported in Table I: a multiplier-accumulator from the Synopsys DesignWare Library, and the implementations of a 1KB RAM memory and the AES and Camellia encryption/decryption algorithms from the Open Core Library. SystemC models of the considered IPs have been obtained from the original Verilog descriptions by using HIFSuite [13]. Table I reports the number of code lines, the size in bits of PIs and POs, the time required for the gate-level synthesis by using Synopsys DesignCompiler, and finally the number of memory elements in the gate-level netlist.

The results of a first experiment are reported in Table II. Above the dashed line, the results are referred to the simulation of the IPs by using the set of test sequences defined for their functional verification, thus they are assumed to cover the most

of IP behaviours. We will refer to such a testset with the name *short-TS*. Below the line, a longer set of test sequences has been applied to stimulate the IP's functionality several times with different set of data. We will refer to such a testset with the name *long-TS*. The precise number of test sequences, which correspond to the total length of the functional traces used to extract the IP's behaviours, is shown in Column *TS*. Column *PX* refers to the time required for generating a corresponding set of reference power traces by using Synopsys PrimeTime PX. The time required by our tool for the generation of the PSMs is then shown in Column *PSMs gen.*, while the number of PSMs' states and transitions are reported, respectively, in Columns *States* and *Trans.* Finally, Column *MRE* reports, as a measure of PSMs' accuracy, the mean relative error (MRE) obtained by comparing the power values estimated through the simulation of the PSMs with respect to the reference values provided by PrimeTime PX. Analysing the results for each benchmark, we first observed that RAM presents a high correlation between the Hamming distance of two consecutive input data and its energy behaviour. Thus, the linear regression-based approach adopted in our tool works satisfactory when it relates the RAM's internal switching activity with the power consumption by observing the behaviours of PIs and POs. For this reason, the MRE is very low, even if RAM behaves as a data-dependent IP from the energy consumption point of view (when it operates in the writing modality). MultSum is a data-dependent IP too. Its MRE is a bit higher than RAM, since to effectively capture its functionality by observing PIs and POs it requires to correlate PIs and POs values on an time window wider than the one currently considered by the linear regression mechanism implemented in the tool. On the contrary, AES and Camellia are not data-dependent. However, differently from RAM and MultSim which have no subcomponent, AES and Camellia are composed by a set of subcomponents. In this case, it could be more difficult extracting, in an automatic way, the correlation between the IP's behaviours and the switching activity by observing only the changes in IP's primary inputs and outputs, without a visibility on internal signals connecting the subcomponents. This is due to the fact that the switching activity is distributed among subcomponents that could present power behaviours poorly correlated to each other. This is exactly the case of Camellia. On the other hand, the subcomponents of AES present a stronger correlation, and thus its MRE is sensibly lower than Camellia. As a final consideration on Table II, we observe that, with the exception of Camellia, the MREs below the dashed line are not sensibly improved with respect to their counterparts above the line. This confirms the fact that high-quality PSMs can be generated from functional traces obtained by simulating the IP with the same testbenches adopted for their functional verification.

In conclusion, Table III reports a performance evaluation and a further accuracy analysis on the PSMs generated by using the *short-TS* set. Columns 2 and 3 show the time required to simulate with the *long-TS* set, for 500,000 instants, respectively, the SystemC model of the IPs (*IP sim.*) and the same IP model connected to the PSMs (*IP+PSMs*). Then, Column 4 shows the simulation overhead due to the presence of the PSMs with respect to simulating the IPs without PSMs. As shown, the overhead ranges between 3% and 26% and it is inversely proportional to the complexity of the IP. An even more significant fact is observed by comparing Column *IP sim.* of Table III with the values reported in Column *PX* under the dashed line of Table II. This shows that estimating the power values by simulating the PSMs is up to two orders of magnitude faster than using PrimeTime PX. This speed-up is not paid in terms of accuracy, as shown by the last two columns of Table III, which report the MRE and the percentage of wrong-state predictions obtained by simulating the PSMs, generated from the *short-TS* testset, with the *long-TS* testset.

IP	*Lines*	*PIs*	*POs*	*Syn. time (s.)*	*Memory elements*
RAM	101	44	32	140.2	8192
MultSum	45	49	32	18.8	225
AES	1089	260	129	42.6	670
Camellia	777	262	131	75.2	397

TABLE I. CHARACTERISTICS OF BENCHMARKS.

IP	*TS*	*PX (s.)*	*PSMs gen. (s.)*	*States*	*Trans.*	*MRE*
RAM	34130	169.0	1.2	9	18	0.30 %
MultSum	12002	19.5	0.6	2	2	4.03 %
AES	16504	144.8	0.7	5	7	3.45 %
Camellia	78004	74.5	5.7	5	10	32.66 %
RAM	500000	3316.7	20.1	9	18	0.29 %
MultSum	500000	750.1	22.6	3	4	3.27 %
AES	500000	3626.0	115.6	13	29	3.09 %
Camellia	500000	2699.0	221.2	5	11	32.64 %

TABLE II. CHARACTERISTICS OF THE GENERATED PSMS.

IP	*IP sim. (s.)*	*IP+PSMs (s.)*	*Overhead*	*MRE*	*WSP*
RAM	13.8	17.5	26.4%	0.29%	0%
MultSum	20.4	24.2	18.4%	3.97%	0%
AES	93.4	98.7	5.6%	3.11%	0%
Camellia	277.1	286.9	3.5%	32.64%	20%

TABLE III. SIMULATION TIMES AND ACCURACY EVALUATION.

VII. CONCLUDING REMARKS

The paper presented a methodology for the automatic generation of PSMs by adopting an approach based on (i) dynamic mining of temporal assertions to extract the IP's functional behaviours from a set of functional traces, and (ii) a calibration process to extract the associated power behaviours from a corresponding set of references power traces. Finally, a Markov model was defined to implement a SystemC simulatable model of the PSMs. The power estimation obtained by a system-level simulation of the automatically generated PSMs is up to two orders of magnitude faster than running a state-of-the-art gate-level power simulator like PrimeTime PX without a significant loss of accuracy for all IPs, but Camellia, which is composed by a set of subcomponents whose power behaviours are low correlated to each other. To mitigate the limitation highlighted by Camellia, we foresee, as future works, the automatic generation of a power model based on hierarchical PSMs that distinguishes among IP subcomponents.

REFERENCES

[1] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proc. of IEEE ISLPED*, 1998, pp. 173–178.

[2] R. Bergamaschi and Y. Jiang, "State-based power analysis for systems-on-chip," in *Proc. of ACM/IEEE DAC*, 2003, pp. 638–641.

[3] S. Schurmans, D. Zhang, D. Auras, R. Leupers, G. Ascheid, X. Chen, and L. Wang, "Creation of esl power models for communication architectures using automatic calibration," in *Proc. of ACM/IEEE DAC*, 2013.

[4] D. Lorenz, P. A. Hartmann, K. Grüttner, and W. Nebel, "Non-invasive power simulation at system-level with SystemC," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, ser. LNCS. Springer, 2013, vol. 7606, pp. 21–31.

[5] D. Lorenz, K. Gruettner, and W. Nebel, "Data-and state-dependent power characterisation and simulation of black-box RTL IP components at system level," in *Proc. of Euromicro DSD*, 2014, pp. 129–136.

[6] H. Lebreton and P. Vivet, "Power modeling in SystemC at transaction level, application to a DVFS architecture," in *Proc. of IEEE ISVLSI*, 2008, pp. 463–466.

[7] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. on VLSI*, vol. 8, no. 3, pp. 299–316, 2000.

[8] http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.

[9] A. Danese, T. Ghasempouri, and G. Pravadelli, "Automatic extraction of assertions from execution traces of behavioural models," in *Proc. of ACM/IEEE DATE*, 2015, pp. 67–72.

[10] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *The annals of mathematical statistics*, pp. 1554–1563, 1966.

[11] T. Swinscow and C. M.J., *Statistics at square one*. BMJ Publishing Group, 2009.

[12] B. L. Welch, "The generalization of 'student's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. pp. 28–35, 1947.

[13] http://www.hifsuite.com.