PowerQuest: Trace Driven Data Mining for Power Optimization

Pietro Babighian, Intel Corp., Leixlip – Ireland
Gila Kamhi, Intel Corp., Haifa – Israel
Moshe Vardi, Rice University, Houston - Texas

Abstract

We introduce a general framework, called PowerQuest, with the primary goal of extracting "interesting" dynamic invariants from a given simulation-trace database, and applying it to the power-reduction problem through detection of gating conditions. PowerQuest adopts machine-learning techniques for data mining. The advantages of PowerQuest in comparison with other state-of-the-art Dynamic Power Management (DPM) techniques are: 1) Quality of ODC conditions for gating 2) Minimization of extra logic added for gating. We demonstrate the validity of our approach in reducing power through experimental results using ITC99 benchmarks and real-life microprocessor test cases. We present up to 22.7 % power reduction in comparison with other DPM techniques.

1. Introduction

The key idea in power management is that unused parts of a complex design can be shut down during system operation. Power-management techniques based on this principle belong to the broad class of dynamic power management (DPM) methods [1]. Recent work [2] in automatic DPM detects conditions where the clock can be stopped via clock gating without compromising functional correctness. The detection of such conditions in real-life RTL designs with minimum impact on area and performance requires extensive logic analysis and is widely acknowledged as a difficult task [2]. The latest techniques in [2] achieve scalability by limiting the usage of *Observable Don't Care* (ODC) conditions to the steering modules of the design. While ODC detection is less expressive for steering modules, it does not capture the most general idleness conditions risking the omission of significant gating opportunities.

We introduce a general methodology and framework, called *PowerQuest,* with the primary goal of extracting "interesting" dynamic invariants from a simulation trace database. *PowerQuest* adopts machine-learning techniques, for data representation, manipulation and invariant extraction. We demonstrate the benefits of dynamic invariant extraction especially to the domain of dynamic power management through the efficient detection of interesting invariants (characteristics) with respect to power consumption (e.g., unobservability).

Why is trace data mining especially applicable to power optimization? Both at micro-architectural level and RTL, power benchmarks (in form of simulation traces) represent the execution scenarios (e.g., common-case, peak activity) that are of interest with respect to power consumption. Optimizing and estimating power based on common-case computation have become an established practice. Thus, dynamic invariants extracted from power benchmarks include common-case legal computation data that is impossible to extract solely from static topological logic analysis.

In this paper, we especially focus on the efficiency of *PowerQuest* in extraction of Observable Don't Care Conditions that are used later for gating unused parts of the design. The advantages of *PowerQuest* in comparison with other state-of-the-art DPM techniques are: 1) Quality of ODC conditions for gating 2) Minimization of extra logic added for gating. Utilizing assumptions embedded in traces that represent the common case computation, ODC conditions detected by *PowerQuest* are potentially simpler. Moreover, *PowerQuest* utilizes the global functional view provided in trace data that is not obtainable from structural topological analysis; i.e., gating (ODC) condition extraction is not limited to latch boundaries or only steering logic modules as in [2] and thus has higher potential of not missing significant gating opportunities. The user can request the formulation of these invariants based on a small set of significant signals (not necessarily in the vicinity of the sequentials to be gated). Furthermore, *PowerQuest* does not add extra logic for gating, but rather *learns* a signal (if any) that can be used as substitute for all or part of the logic needed to control the activation. Computation of such a signal would have been practically impossible using only structural/functional analysis without the trace data.

In the general usage mode, one can extract invariants that can characterize various scenarios of interest with respect to power (either idleness or excessive activity). The end result will be acquisition of new and useful information that can help the designer make better decisions in order to manage power dissipation. Alternatively, this information can be utilized by downstream synthesis tools to automatically perform the gating. As usage scenarios, we can consider both power-saving opportunity detection and root-cause analysis of excessive power consumption.

This paper is organized as follows. In Section 2, we present an overview of *PowerQuest* system and usage scenarios. In Section 3 we describe in detail the data-mining techniques used for power-aware dynamic invariant extraction. Section 4 explains how RTL power-saving opportunities detected by known state-of-the-art dynamic power management (DPM) techniques can be identified by *PowerQuest*. In Section 5, we present experimental results using ITC99 [3] benchmarks and real-life microprocessor test cases that show up to 22.7 % power reduction over classic dynamic power management techniques. Section 6 compares *PowerQuest* with related work. Finally, in Section 7 we summarize and conclude.

2. System Overview

PowerQuest makes use of *Learning from Examples* [4] techniques (more specifically Extended Matrix Approach [5,6]) to identify idleness situations that are likely to occur and extracts simple Boolean conditions that can be used for the gating of the idle blocks.

PowerQuest gets as input simulation traces based on validation or power benchmarks. We will refer to each trace as a set of simulation data points over time $P=\{p_1,...,p_n\}$ where a point p_i

978-3-9810801-2-4/DATE07 © 2007 EDAA

represents the values of all the signals at a given clock phase *i*. An additional input to *PowerQuest* consists of the design features $\{F_1,...,F_k\}$ needed to classify the positive and negative data points (clock phases) of the traces with respect to a splitting criteria identified by label L. Each label partitions the set of points into two sets: positive examples (PE), which represent data points with label 1, and negative examples (NE), which represent data points with label 0.

PowerQuest has a few usage modes which can affect the way positive and negative examples (trace data points) are classified. The user can control: 1) Time interval of interest 2) Signals of interest 3) Assumptions (constraints) under which the invariants are mined. The trace data is split based on this input and restricted to the data of interest. Obviously, control parameters can significantly restrict the size of the data to be mined.

The signals of interest dictate the variables of the invariant relation. The filtering of the invariants based on the comparison between positive and negative invariants ensures that every candidate learned invariant p_i satisfies all the positive points PE and does not satisfy any of the negative points NE.

PowerQuest extracts dynamic invariants from simulation data, which by definition is not exhaustive. Thus the validity of each invariant detected by *PowerQuest* has to be formally verified (otherwise conditioned to the specific cases where it holds).

3. Learning From Traces

Generally speaking, *Learning from Examples* [4] addresses the problem of extracting, for a given group of objects, a set of predefined *features* differentiating the objects, and a classification of the objects as positive and negative examples, a rule that generates the classification in terms of feature values. Since this is equivalent to classification of the given objects in accordance to the set of *features*, we call this process *class label* extraction. More in detail, the above learning process can be viewed as the process of going from specific observational knowledge about some objects to a general rule that implies or accounts for the observations. In this section we will make the analogy of Learning from Examples technique to learning from simulation traces.

3.1 Class Representation

We start with a set $P=\{p_1,...,p_n\}$ of data points. We then define a set $\{F_1,...,F_k\}$ of features. With each feature F_i we associate a data domain D_i. Each feature F_i is a function from P to D_i. Thus, with each point p_j we associate a feature tuple $F(p_j)=<F_1(p_j),...F_k(p_j)>$. The set of these tuples constitutes the feature space. A label L is a function from P to the domain {0,1}. Thus, each label partitions the set of points into two sets: positive examples (PE), which represent data points with label 1, and negative examples (NE), which represent data points with label 0.

For example, let us consider the circuit in Figure 1 where $S=(S_1,S_2,\ldots S_6)$ is a subset of the signals driving combinational logic (in grey) as well as sequential elements (in white). Let us also assume that the above logic network has been *levelized* in decreasing topological order leading to 6 logic levels where sequential elements are at level 0 while combinational logic is distributed over levels 1 to 5. By simulating the circuit, we obtain a set of points, each corresponding to the state of the circuit at a given point in time. This provides the set P of data points. What are the features by which the points are described? Obvious features are signal values. Such features have logic values. By default, *PowerQuest* extracts built-in features as well as user defined features. Some examples of power relevant features for a point p_i are Sw_F and *Loc,* representing, respectively, the number of switching signals at a given clock cycle, and the locality of the switching signals (i.e., the average topological distance of the switching signals in S from the input of registers). Values for feature Sw_F can be *quantitative* (extracted by measurements from data traces) or *qualitative* based on a *subjective* division into categories provided by the user.

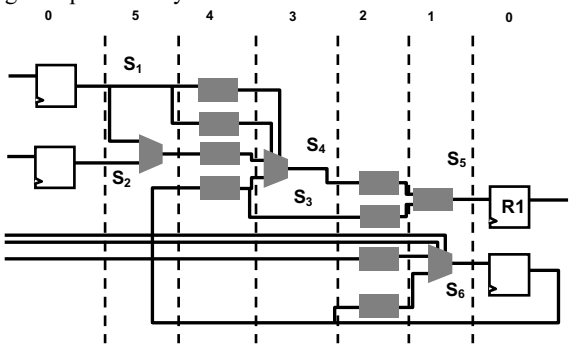


Figure 1. A generic sequential circuit.

For example, if we find out that at a given clock cycle the number of switching signals is just 5% over the total amount of signals in S, then we can associate the numerical value 5% to the attribute Sw_F. On the other hand, if we argue that a percentage greater than 5% represents a quantity distinguishing between a high-activity scenario from a lower-activity one, then we can associate the symbol *High* (first category) to every value greater than 5%, the symbol *Low* (second category) to the values lower or equal to 5% and *None* (third category) to the value 0%. Similarly to the feature Sw_F, *Loc* can exhibit numeric values (measurement of level proximity) as well as *subjective* category-based values. For example, the user might specify as input to *PowerQuest* the following categories: *Far≥4*, *4<Close<1* and *None=1*.

A *label* L is a function from P to the domain {0,1}. A *label* can be viewed as a Boolean feature, but we give it a special role in classifying the data points as positive or negative examples. As an example, let us assume that there exists a Boolean function say $L=f(S_1,S_2,\ldots S_6)$ that is 1 if and only if both the input and the output of the flip-flop R1 in Figure 2 have the same logic value at a given clock cycle. Thus we can classify each cycle in accordance to the idleness due to the concurrent data retention at both input and output of R1 (*hold condition*). In other words, we can define two *labels* L=1 and L=0 associated respectively to the *hold* and *not-hold* conditions.

As previously mentioned, the goal of *Learning from Examples* is all about learning a relation between *features* and *labels*, which can be used as a classification model as well as an explanatory tool to distinguish between objects of different *classes*.

An atom is a formula of the form $x_i \in A_i$ or $\sim(x_i \in A_i,)$ where A_i is a subset of D_I, the domain of the feature F_i. A feature tuple $<v_1,...,v_k>$ satisfies the formula $x_i \in A_i$ (resp., $\sim(x_i \in A_i)$), if $v_i \in A_i$ (resp., $\sim(v_i \in A_i)$). We are interested in formulas in disjunctive normal form, that is, disjuncts of conjunctions of atoms. A feature tuple $t=<v_1,...,v_k>$ satisfies a formula *f* if there is a disjunct *d* of *f* such that *t* satisfies all conjuncts of *d*. Our goal is to learn a formula that describes the sets of positive and negative examples; i.e., it is satisfied by all positive examples and is falsified by all negative examples. That is, for a point p, L(p)=1 implies that F(p) satisfies *f* and L(p)=0 implies that F(p) falsifies *f*.

Clearly, a tuple $t=<v_1,...,v_k>$ can be described uniquely by the conjunction $d_t = x_1 \in \{v_1\}$ & ...& $x_k \in \{v_k\}$. Thus a formula *f* that describes PE and NE always exists; simply take *f* to be the disjunction of all conjunctions d_t, for *t* in PE. Our goal is to minimize the size of the formula *f*. For example, the formula *f* might express clock-gating logic and we want to minimize its power requirements. In general, minimizing the formula *f* is a generalized covering problem and is NP-hard [14], requiring us to resort to a heuristic approach.

3.2 PowerQuest Implementation

PowerQuest extracts classification rules from positive and negative examples. Each example characterizes a data point (in our case a specific clock cycle), and its *label* specifies the correct decision associated with that point. The generated decision rules are expressed as symbolic descriptions involving relations between feature values of the point.

As an example, if we analyze each signal in $S=(S_1,S_2,\ldots S_6)$ in Figure 1 over a period of five clock cycles corresponding to five data points $(p_1,p_2,\ldots p_5)$, we can classify the behavior of the signals themselves over the aforementioned set of *features* and with respect to the *labels* previously extracted (*hold* and *not-hold* conditions) obtaining a table similar to the following one that we can call *features-label* table.

Cycle	*Data Point*	*Sw*	*Loc*	*Label*
C_1	P_1	None	None	1
C_2	P_2	Low	Close	0
C_3	P_3	High	Far	1
C_4	P_4	None	None	1
C_5	P_5	High	Close	0

The rows associated with cycles C_1, C_3 and C_4 (i.e., data points p_1, p_3, p_4) represent *positive examples,* while rows associated with cycles C_2 and C_5 (i.e., data points p_2, p_5) represent *negative examples*. By analyzing the above table, it is easy to extract the following relation: "*The hold condition for R1 is satisfied if and only if there is either no activity within the support of L (i.e., the set S) or activity only in signals that are topologically far (i.e., N logic levels away when N defines the threshold of locality feature) from R1 itself*". Nevertheless, not all relations that can be extracted from a features-labels table are interesting. More importantly, the number of *features* and/or examples may be too high. For this purpose, we focus our attention on automated techniques that aim at extracting rules from unordered set of positive and negative examples; specifically, we use the *Extension Matrix Approach* (EMA) developed by Hong [5] and subsequently improved by Wu [6]. In more detail, the EMA is a covering technique, which represents knowledge in the form of disjunctive Boolean formula by performing a heuristic search through a space of logical expressions, until it finds a decision rule that covers the positive examples, while leaving out the negative examples.

In summary, *PowerQuest* detects invariants in two steps. In the first phase, power saving opportunities are extracted by splitting trace data (i.e., the set of examples) into two groups of examples (i.e. positive and negative) and by *learning* a Boolean function that exhibits a False value for all negative examples and a True value for the positive ones.

In the second step, the *learning space* of the mining process is reduced to the previously extracted set of positive examples in order to quantify user-defined metrics (e.g., switching activity of signals and signals activity peaks) that are of interest. In other words, *PowerQuest* starts learning values of features provided by the user within the space of positive examples and the fact that a given feature exhibits a specific value (or within a specific range of values) represents a property that is *invariant* within the set of positive examples.

4. Power-Savings Opportunities

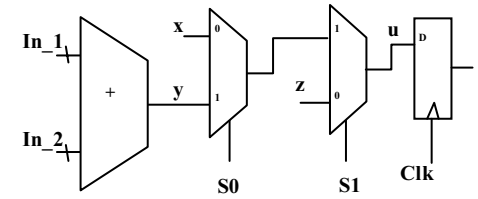
We now show how *PowerQuest* automatically detects common DPM opportunities as *Operand Isolation* [7] through efficient approximation of observability don't care function from trace data. Then, we demonstrate through Pre-computation [8, 9] technique how *PowerQuest* minimizes the gating logic by learning a signal that can substitute the gating logic.

4.1 Operand Isolation

Operand isolation [7] is a technique to minimize the power overhead incurred by redundant operations by selectively blocking the propagation of switching activity through the circuit. In Figure 2-a, we observe that there is an active path from the adder to the flip-flop only when the control signals Sel_0 and Sel_1 are both high. In all other cases, the output of the adder is not visible by the flip-flop, leading to useless computation when a change of value occurs at the input of the adder. As shown in Figure 2-b, operand isolation stops any activity at the input of the adder by insertion of a block of gating logic that is activated by a signal *AF* (Activation Function) with the ability to detect the useless cycles of the adder. *AF* is high in the "active" clock cycles when the adder output is being used and low otherwise. In other words, *~(AF)* represents the *Observability Don't Care* (ODC) [10] condition of the output *y* of the adder block and it can be extracted by computing the condition $ODC^y = (F_{y=1}\ XOR\ F_{y=0})$, where F_y represents a Boolean function of the variable *u* with variable *y* in its support, while $F_{y=1}\ XOR\ F_{y=0}$ is the ordinary *Boolean difference* $\partial F/\partial y$ of F_y with respect to *y*. The value of ODC^y is thus True in those cases where *y is not* observable at *u*, and False otherwise. In our approach we generate ODC^y of a signal *y* with respect to a signal *u* by initially obtaining the function F_y based on simulation traces and subsequently we simplify F_y properly in order to find two simple Boolean functions representing $F_{y=0}$ and $F_{y=1}$. This can be accomplished by generating sets of positive and negative examples based on the *splitting condition u=1* and then learning the function F_y by establishing a relation between *u* and *y*. EMA then yields a DNF function F_y that approximates the real Boolean function (let us name as g) driving the signal *u* for the given circuit. We do know, however, that F_y agrees with g on all points in the trace. Now we can take the ODC condition to be ($F_{y=0}$ XOR $F_{y=1}$). Clearly, we need to use formal techniques to ensure that our ODC is correct (i.e., the gating based on the computed ODC will be preserving the functionality of the original circuit). Although in this simplistic example both trace based data mining and structural logic analysis techniques would compute the same unobservability condition $ODC^y = (S_0 \cdot S_1)$, trace based data mining has the potential of detection of simpler and more global ODC conditions utilizing the common-case legal computation data embedded in the traces.

4.2 Precomputation

The procedure adopted in previous section to extract ODC conditions for *Operand Isolation* can be extended also to the case of *Precomputation* [8]. *Pre-computation* relies on duplication of part of the logic with the purpose of pre-computation of an ODC condition one clock cycle earlier. In this section, we will demonstrate how *PowerQuest* does not add extra logic for gating, but rather *learns* a signal (if any) that can be used as substitute for the computed ODC condition.



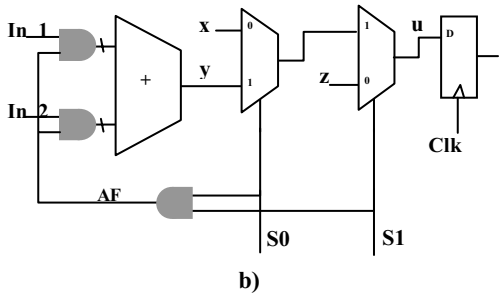


Figure 2. Exploitation of Operand Isolation.

Referring, for example, to the circuit in Figure 3, we may want to use the previously extracted activation function *AF* in order to selectively stop the clock signal and thus gate the activation of sequential elements that eventually drive the adder input. In this manner, we decrease power consumption not only in the fan-out cone of logic of the input flip-flops but also within the flip-flops themselves. The main difficulty, however, is due to the fact that ODC conditions masking flip-flops in clock cycle "T" should be used to gate their clock in cycle "T-1". In other words, the clock-gating logic should be active in the clock cycle immediately before the flip-flop becomes unobservable [2]. Unfortunately, the control signals at the inputs of the ODC functions are generated one clock cycle too late. If the control signals S_0 and S_1 are available directly as outputs of flip-flops, the instantiation of the clock-gating logic is relatively straight-forward. Logic gates implementing the ODC conditions are inserted and their inputs are connected to the inputs of the flip-flops driving S_0 and S_1. In real-life designs, however, the control inputs of the steering modules are seldom coming directly from the flip-flops; instead, they are often generated by additional logic, as shown in Figure 3. In this case, the entire cone of logic between flip-flops and control signals S_0 and S_1 needs to be duplicated and connected to the inputs of the flip-flops, and the ODC computation gates need then to be connected to the outputs of the duplicated cones [2]. Clearly, the addition of this extra logic may represent a non-negligible overhead. Referring to Figure 3, our approach tries to learn about the existence, within the fan-in cone CL, of a signal, say *t,* whose logic behavior is similar to the ODC^y condition.

In such a case, *t* can be used as an active-low signal to condition the activation of registers driven by the block of logic CL. The extraction procedure of the signal *t* is straight-forward and consists of four major steps:

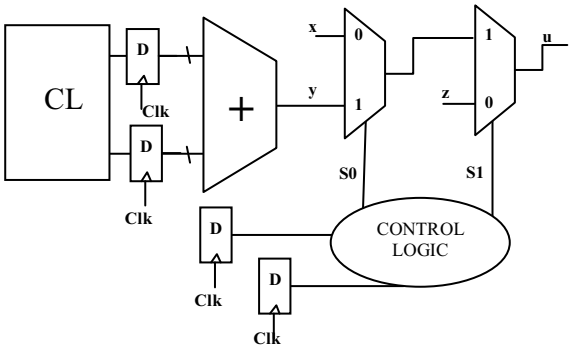


Figure 3. A generic sequential circuit

1. A Boolean function representing the ODC^y is computed by the method discussed earlier. A matrix of ODC^y *positive and negative examples* is built. We call such a matrix OEM^y (i.e., ODCs Examples Matrix) and it contains all the configurations for which ODC^y equals 1 or 0. In this learning process, the features are values of signals of S_0 and S_1. We split the set of trace points according to ODC^y == T and ODC^y == F and label them in the matrix, respectively.
2. By considering the simulation trace of the signals within the block *CL*, a matrix of *unclassified* examples is generated. We call such a matrix *UEM* (i.e., Unclassified Examples Matrix) and it contains only features' (signals') values since no *class label extraction* has been performed.
3. We compare each example (i.e., signals' values at a particular trace point/cycle) in the matrix *UEM* with the labels in OEM^y. More specifically, we identify all the features (signals) that exhibit value 1 in an example (trace point/cycle) p_i in matrix *UEM* and has ODC^y = T in the next example p_{i+1} in matrix OEM^y as potential candidates to be clock-gating signals.
4. Finally, the set of found features (if any) is pruned in order to get the set which exhibits the largest number of 1's. The signals associated to the features in this set will have functionality close to ODC^y and thus will be good candidates for clock gating (i.e., good candidates for *t*).

In Figure 4, the OEM^y matrix for the *Observability Don't Care condition* $ODC^y = (S_0 \cdot S_1)'$ extracted previously is represented. The columns in grey represent the clock cycles during which signal *y* is *not* observable (ODC^y=1). Let us suppose that block *CL* in Figure 3 contains exactly four signals A_0, A_1, A_2, A_3 represented by the *UEM* matrix shown in Figure 5. We want now to find a signal among A_0, A_1, A_2, A_3 that could be used effectively as clock-gating signal for the sequential element driven by the block *CL*. This can be accomplished by considering the examples (i.e. columns) in UEM and by checking for each of them if the value of ODC^y to the associated *next* example in OEM^y is T or F revealing the presence or absence of observability at the next cycle for the signal *y*. In particular, if *y* is observable at the next clock cycle then the considered example (column) in UEM is removed since it does not have a candidate feature (signal) that

can be used as activation signal. By applying the above procedure we are, hence, pruning the set of examples in *UEM* obtaining a smaller *UEM* matrix as shown in Figure 6. In particular, the higher the number of 1's, the better it is since the signal will cover more cases of un-observability. For the case above, the number of 1's of each row is

ODC^y	T	T	T	F	T	T	T	F	T	T	T	F	T	T	T	F
S_0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
S_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 4. The *OEM*^{y} Matrix

A_1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
A_2	0	0	1	0	1	1	1	1	0	0	1	0	1	1	1	1
A_3	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
A_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 5. The *UEM* Matrix of traces for the block of Logic *CL*

A_1	1	1	1	1	1	1	0	0	0	0	0	0	
A_2	0	0	0	1	1	1	0	0	0	1	1	1	
A_3	1	0	1	0	0	1	0	0	1	0	0	1	
A_4	0	1	1	0	1	1	0	1	1	0	1	1	

Figure 6. The pruned *UEM* Matrix

respectively, $A_1 = 6$, $A_2 = 6$, $A_3 = 5$, $A_4 = 8$ leading to the conclusion that signal A_4 is the best choice.

5. Experimental Results

We have implemented trace based data mining using the EMA approach and supporting the proposed flow and methodology for power optimization in *PowerQuest*. In order to evaluate the scalability and performance trends, we ran *PowerQuest* on an industrial micro-processor design as shown in Figure 7. The efficiency of the tool in extracting interesting invariants that aid getting significant power reduction was tested, as shown in Table 1 and 2 using ITC'99 suite. The industrial design is a data-path block with very regular structure, extracted from a floating-point multiplier of a state-of-the-art micro-processor design. It consists of 935 sequential elements, 599 primary inputs and 309 primary outputs. Experiments have been performed on a Dual-Core Workstation with 4 Gb of main memory. Performance of *PowerQuest* has been evaluated in two different scenarios over trace data of 10000 cycles. In the first case, we constrained the mining process by specifying a property under which invariants (i.e., unobservability conditions) should be extracted. In practice, we directed the extraction to gating conditions for sequentials by constraining the splitting condition with SA (Switching Activity) > 0.5 for each sequential thus reducing the search space. In the second case, we did not specify any initial constraint, letting *PowerQuest* make its choice. Clearly in such a case performance decreases since the tool has to process the entire data set in order to gather *features* as well as splitting conditions. As shown in Figure 8, the analysis of over 500 signals takes less than 60 minutes proving the scalability of the method to real-life test case. Tool efficiency in mining interesting invariants has been evaluated on 10 benchmarks included in the ITC'99 suite [3]. We have chosen the benchmarks since the size of the benchmarks facilitates in-depth analysis of results with respect to their gating quality with respect to power and their functional validity. Table 1 reports, for each benchmark, number of cells (Cells), number of flip-flops (FF), number of primary input (PI) and output (PO) as well as number of interesting invariants associated with power saving opportunities such as Operand isolation (OI-Inv) and Pre-computation (PComp-Inv). Also, Table 1 reports number of invariants found by *PowerQuest* and related to the existence of regions in the design with distinguishably different Switching Activity (SA-Inv) and Activity Peaks (APk-Inv). We computed the gating manually utilizing the gating

Bench	*Cells*	*FF*	*PI*	*PO*	*OI-Inv*	*PC-Inv*	*SA-Inv*	*APk-Inv*
Bch_01	45	5	4	2	2	0	0	0
Bch_02	25	4	3	1	0	0	0	0
Bch_03	150	30	6	4	7	8	2	5
Bch_04	480	66	13	8	12	3	1	6
Bch_05	608	34	3	36	17	2	1	9
Bch_06	66	9	4	6	1	0	0	0
Bch_07	382	51	3	8	8	5	4	9
Bch_08	168	21	11	4	3	1	1	5
Bch_09	131	28	3	1	3	2	4	2
Bch_10	1000	121	7	6	21	11	7	12

Table 1. Invariant Extraction

condition automatically detected by *PowerQuest* and compared the results against classic ODC-based clock gating introduced in [2] by running Power Compiler on the output design. The validity of dynamic invariants found by *PowerQuest* was checked by formally verifying the functional equivalence of the gated and original designs. For the results reported in Table 2, we have assured functional correctness of the gated design utilizing the gating function computed by *PowerQuest*. Gating results for *Bch_01*, *Bch_02* and *Bch_06* have been left out in Table 2, since no invariants associated to pre-computation opportunities are found, eliminating one of the prerequisite needed for exploitation of ODC-based clock gating. As shown in Table 2, *PowerQuest* extracted ODC based clock gating gets superior results (up to 22% power reduction with no negative impact to timing and area). Interestingly, delay on critical path improves up to 31.7%. This is due to the fact that implementation of ODC-based clock gating is not performed by involving logic duplication or retiming transformation (as in [2]); rather, *PowerQuest* extracts signals (if any) already available within the design that can be used as substitute for all or part of the logic needed to control the activation of FFs based on *Observability Don't Care* conditions. Moreover, since *PowerQuest* tries to approximate the general ODC condition rather than the approximation of the restricted ODC condition in [2], it can come up with higher quality gating condition.

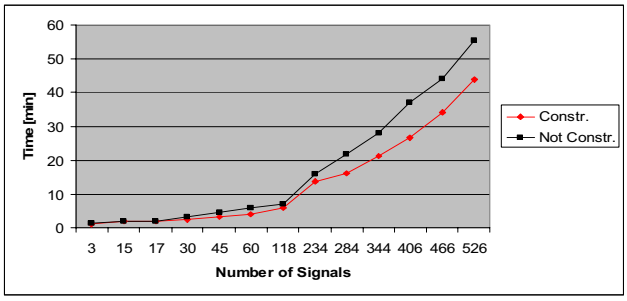


Figure 7. Performance evaluation on different scenarios.

Bench	ODC-based Clock Gating				PowerQuest ODC-based Clock Gating							
	Cells	Area	Delay [ns]	Power [uW]	Cells	Cells [%]	Area	Area [%]	Delay [ns]	Delay [%]	Power [uW]	Power [%]
Bch_03	94	1640	1.84	513.4	77	18.1	1555	5.2	1.57	14.7	437.4	14.8
Bch_04	278	4726	2.08	911,9	193	30.6	4186	11.4	1.62	22.1	874.5	4.1
Bch_05	280	4073	2.66	299.3	263	6.1	3837	5.8	2.37	10.9	275.2	8.0
Bch_07	184	3173	1.89	476.7	160	13	2931	7.6	1.29	31.7	435.8	8.6
Bch_08	102	1450	1.96	241.8	93	8.8	1364	5.9	1.75	10.7	226.3	6.4
Bch_09	94	1561	1.65	450.8	92	2.1	1525	2.3	1.25	24.2	444.6	1.3
Bch_10	601	8855	2.77	1242	578	3.8	8628	2.56	2.43	12.3	958.9	22.7

Table 2. *PowerQuest* ODC-based Clock Gating vs. ODC-based Clock Gating in [2]

6. Related Work

Other approaches that have contributed to dynamic invariant detection are [12, 13]. Invariant extraction from software traces has been suggested in [12], and then adapted in [13] to hardware traces. The invariants are selected from a set of candidate invariants; the tool just has to check which invariants hold in the input traces. The construction of the set of candidate invariants is hardwired into the tool. In our approach, the selection of features and labels is left to the user, but the invariants are extracted using machine-learning techniques. We pay a higher computational price, but we are able to extract much more complex invariants. Moreover, our approach is unique in the application of dynamic invariant extraction to the domain of dynamic power-reduction opportunities detection (See [15] for applications of machine-learning techniques to the design of shut-down policies for putting modules into sleep mode). We utilized here Extension Matrix Approach (EMA) as the machine learning technique based on the experience reported in [5, 6]. Whether other machine-learning approach, cf. [4, 11], can perform better in the context of trace driven data mining for dynamic invariant extraction for power reduction is an interesting question and has not been the focus of our current research. In summary our contribution is in the demonstration the benefits oftrace data-mining for extraction of dynamic invariants for power optimization rather than application of the specific machine learning technique (i.e., EMA). Our machine-learning techniques overcome the difficulties of the standard approach of extracting clock-gating opportunities by structural analysis [2]. The advantages of our approach have been three folded: 1) Gating (ODC) condition approximation is not limited to latch boundaries or only steering logic modules as in [2]. 2) Utilizing assumptions embedded in traces that represent the common case computation of the design, the ODC conditions that we compute are potentially simpler. 3) We do not add extra logic for gating but rather *learn* a signal (if any) that can be used as substitute for all or part of the logic needed to control the activation. Computation of such a signal would have been practically impossible using only structural/functional analysis without the trace data. The disadvantage of our approach is the need of formal verification of the gating conditions since we extract dynamic invariants from simulation data, which by definition is not exhaustive.

7. Conclusions

We introduced here *PowerQuest*, a novel framework, with the primary goal of extracting of "interesting" invariants for power optimization given a simulation trace database, based on machine-learning techniques. We demonstrated using ITC99 benchmarks how *PowerQuest* can automatically extract classic dynamic power-management (e.g., pre-computation, operand isolation) opportunities and get up to 22.7 % reduction on power without negative impact on delay and area over selected classic power reduction methods. Additionally, using real-life microprocessor benchmark data we demonstrated the method's robustness and scalability on real-life design environment. To the best of our knowledge, *PowerQuest* pioneers the usage of machine-learning techniques in mining dynamic (i.e., simulation based) power data to facilitate power reduction.

References

[1] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer, 1997

[2] P. Babighian, L. Benini, E. Macii,"A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation," *IEEE Transactions on CAD of Integrated Circuits and Systems, Volume: 24, Issue: 1 , Jan. 2005.*

[3] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html

[4] J. G. Carbonell, (Ed.). "Machine Learning: Paradigms and Methods", MIT Press., 1990.

[5] J.R. Hong,"AE1: An Extension Matrix Approximate Method for the General Covering Problem", Int. J. Comput. Inf. Sci., Vol. 14, 1985.

[6] X.D. Wu,"Rule Induction with Extension Matrix", Journal of the American Society for Information Science, Vol. 49, No. 5, 1992.

[7] M. Munch, B. Wurth, R. Mehra, J. Sproch, N. Wehn, "Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths," DATE-00: IEEE Design Automation and Test in Europe.

[8] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *IEEE Transactions on VLSI Systems*, December 1994

[9] J. Monteiro, S. Devadas, A. Ghosh, "Sequential Logic Optimization for Low Power Using Input-Disabling Precomputation Architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Mar 1998.

[10] M. Damiani, G. De Micheli, "Don't Care Set Specifications in Combinational and Synchronous Logic Circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, March 1993.

[11] T. Mitchell, "Machine Learning", McGraw Hill, 1997.

[12] M.D. Ernst, J. Cockrell, W.G. Grisswold, and D. Notkin, "Dynamic Discovering Likely Program Invariants to Support Program Evolution", *IEEE Transactions on Software Engineering*, Feb 2005

[13] S. Hangal, N. Chandra, S. Narayan, S. Chakravorty, "IODINE: a tool to automatically infer dynamic invariants for hardware designs". Proc. *Design Automation Conference*, 2005, pp. 775-778.

[14] R.S. Michalski and R.L. Chilausky, "Learning by Being Told and From Examples," *International Journal of Policy Analysis and Information Systems*, 1980

[15] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy effcient programmable computation," *IEEE Trans. on VLSI Systems,*Vol.4, No. 1 (1996), 42-55.