Reducing Power with Activity Trigger Analysis

Jan Lanik*, Julien Legriel†, Erwan Piriou‡, Emmanuel Viaud†, Fahim Rahim†, Oded Maler*, and Solaiman Rahim †
*Verimag, †Atrenta, ‡CEA-LIST
§Email: {jan.lanik,oded.maler}@imag.fr, {julien,emmanuel,fahim,solaiman}@atrenta.com, erwan.piriou@cea.fr

***Abstract*—In this paper we propose and implement a methodology for power reduction in digital circuits, closing the gap between conceptual (by designer) and local (by EDA) clock gating. We introduce a new class of coarse grained local clock gating conditions and develop a method for detecting such conditions and formally proving their correctness. The detection of these conditions relies on architecture characterization and statistical analysis of simulation, all done at the RTL. Formal verification is performed on an abstract circuit model. We demonstrate a significant power reduction from 33 to 40% of total power on a clusterized circuit design for video processing.**

I. Introduction

Power consumption has become a very important parameter of current electronic designs, for battery-powered devices such as mobile phones as well as for non-portable applications due to the requirement for temperature control in high performance modern chips [1]. Although power consumption can be measured and estimated precisely only after physical implementation, it turned out to be important to have reliable power analysis early in the design flow, at the register transfer level (RTL). Architectural decisions made at the RTL level can severely impact power consumption and detection of bad architectural choices at later stages might require an iteration back to RTL and incur time delays with negative financial consequences.

A prominent source of power dissipation is *dynamic power*, which is consumed when the transistors in the circuit change their states [2]. A significant part of this power is dissipated in the clock tree [3]. *Clock gating* is one of the most efficient techniques for reducing power dissipation. It is based on disabling the clock of design blocks when they do not perform any useful computation. This leads to direct power savings in the clock tree and, in some cases, in the block itself [1]. The implementation of this technique depends on *clock gating conditions*, which specify when an associated block can be safely deactivated.

Clock gating conditions can be identified at two abstraction levels. At the architectural (or conceptual) level, large functional units may be clock-gated using high-level control signals. For instance, the floating point unit of a processor can be clock-gated when the currently processed instructions do not require such a computation. At the local level, registers can be clock-gated based on a local analysis of the circuit logic. This typically involves looking at the conditions under which the register data is being read (e.g., using select conditions), or propagating enables of upstream/downstream registers.

Local clock-gating has been shown to significantly reduce dynamic power consumption and is supported by several EDA tools that identify and implement clock-gating based on ODC/STC conditions, a technique which has shown some success [4]. However, a typical issue in this methodology is the complexity of the enable conditions at the local level. Sometimes these conditions are too complex and will be avoided by designers because it is unclear if the change will be safe at later design stages such as timing closure and routing. Another issue is the trade-off between the power savings achieved and the number of required changes in the RTL. Local conditions typically apply to a single bus or few flip-flops and complex (and hard to verify) conditions should be used to obtain significant savings.

Being able to provide simple and easily understandable power saving opportunities is thus an important requirement for power reduction tools. We claim that this can be achieved by detecting clock gating conditions at an *intermediate* level of abstraction, coarser than typical local clock gating methods but small enough to be difficult to spot by manual analysis. Typical targets for this type of conditions are medium-size functional units such as HDL modules with significant dynamic power. There are several advantages in using such clock gating conditions.

1. We can target simple and architecture-related conditions made of a few control signals which are cheap to implement in terms of added circuitry. They are more comprehensible to designers who can judge their correctness by themselves and make more confident decisions whether or not to use them. Ideal conditions are such that the designer himself could discover by a detailed manual analysis.

2. A single clock gating implementation can save much larger amount of power if it is used higher in the design hierarchy which furthermore enhances the complexity/saving trade-off compared to classical local methods.

Intermediate level clock gating closes a gap between conceptual and local clock gating. Clock gating at this level is very attractive, because of its potential to provide significant power savings with minimal changes to the circuit. Moreover, to the best of our knowledge, conditions for intermediate level clock gating are currently beyond the scope of what EDA tools can identify. It is an empirical question whether they are abundant, at least in some application domains, and how difficult it is to find them. The preliminary findings from our experiments are encouraging.

Main contributions

- We introduce a class of clock gating conditions called *activity triggers* that target intermediate size design blocks and which are typically related to architectural intent.

- We develop an algorithm which heuristically detects potential activity triggers based on a statistical analysis of activity files (VCD or FSDB) generated by RTL simulation of the design. These potential triggers should be verified by a designer or using formal methods.
- We formalize the concept of activity triggers and their associated clock gating conditions. We define the temporal property corresponding to the fact that the trigger is correct and the clock gating based on the trigger is safe. We use model checking to formally verify the property.
- We propose and discuss a complete methodology where these techniques, that we have implemented within a commercial EDA tool, are used in an iterative semi-automatic fashion for finding activity triggers and efficient clock gating conditions. We demonstrate the methodology on an industrial video processing design, achieving a significant reduction of power.

Organization of the paper

The rest of the paper is organized as follows. Section II presents related work on power reduction using clock gating techniques. Section III introduces the idea of *activity triggers* and illustrates them on a simple (but yet realistic) design. The concept of activity triggers is then more formally stated in Section IV. The core of our method – the statistical detection and formal verification of *activity trigger* is described in Sections IV and V. The complete proposed methodology is detailed in Section VI. In Section VII we present the experimental results on the video processing case study. Finally we conclude in Section VIII and discuss the future work.

II. Related work

There are multiple algorithms to compute the conditions for clock gating. In general, the approaches can be classified as *Observability don't care (ODC)* or *Stability condition (STC)* based. ODCs [4]–[8] were originally introduced for application in logic synthesis. The clock can be gated if outputs of a module are not observable. STCs [4], [7], [9] utilize the fact that if the next value of a register is equal to the one that is already stored in it, the register can be gated and the clock switching power saved. This is especially useful if an STC is valid for a larger block of registers, so that gating can be performed higher in the clock tree. *Activity triggers* are essentially STCs that are valid for relatively big design blocks.

The ideal clock gating conditions are often very complex and the cost of the additional clock gating circuitry would be higher than the savings. Therefore, approximate methods that compute weaker but simpler conditions were developed [6], [10], [11]. Our methodology is designed to compute simple conditions, hence approximation methods are not needed.

When designers are analyzing a circuit in order to find local clock gating conditions, some clock gating can be already present. This can be due to clock gating on the architectural level, or due to reusing parts of the design that are already clock gated. Some authors [6], [7] focused at strengthening such clock gating conditions that are already implemented in the circuit, strengthening conditions that were computed by other methods and combining them into more powerful conditions. Conditions discovered by our approach can be used as an input for these strengthening methods. Conversely, some partial clock gating in the analyzed design block could be used for simplifying the formal verification part of our flow, however this is not implemented. For detection, we ignore signals that are already used for clock gating, as our main goal is to find conditions that were not considered by a designer previously.

Hurst [9] introduces a guess-and-prove approach, which selects clock gating condition candidates from the existing nets in the design using heuristic based on timing and structural considerations. Candidates are then pruned using simulation and a formal proof is attempted for the remaining nets. Such conditions have the benefit of being already physically available in the design, therefore the added clock gating circuitry is very simple. A similar guess-and-prove approach was described in [**?**] where the candidates are not single nets but pairs of them, such that a simple invariant holds over them (a particular value of one signal implies a particular value of other signals). The invariant can be used for ODC based clock gating (the fanin of the implied signal can be disabled). The functional correctness has to be formally verified.

Another guess-and-prove method is described in [12] and [13]. This approach uses machine learning on simulation traces to infer clock gating conditions that are less complex and cheaper to implement than those coming from the traditional structural detection methods (e.g., [4]). In contrast with [9], simulation traces in [12], [13] are not just only to prune incorrect candidates, but also to collect the candidates from the positive examples in the simulation. The conditions are however very local (the analysis is done on the level of single registers).

Outside of power reduction context, the idea of mining simulation traces for useful design intent related invariants was introduced in [14] for software and later in [15] for hardware.

Our approach is based on a guess-and-prove concept, the candidates are generated based on a statistical analysis of a simulation trace, which yields a relatively small set of candidates for clock gating. Furthermore, we take the user in the loop, so that he can interactively add constraints, which may be necessary for the formal check or he can verify candidates manually if the module is too big for formal check, but the condition is simple and easy to understood. The main advantage compared to the previous work, however, is that the conditions that we collect are usually coarse grained, simple and closely related to the design intent.

III. Activity triggers

Our approach is focused on clock gating conditions that are not necessarily optimal gating conditions for a particular register but are simple to implement and shared by many registers; typically by an entire HDL module. These conditions are related to different modes in which a digital design operates. If a circuit operates in multiple different modes, there may be parts of the circuit that are used only for one of these modes. Thus, the goal is to find conditions that correspond to moving

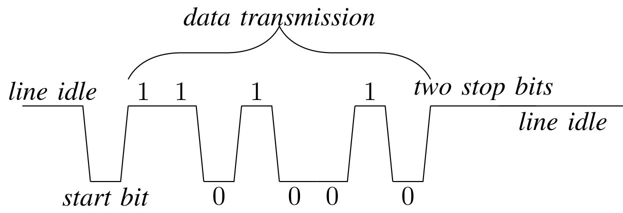

Fig. 1. Serial line transmission of the character ‘K’ in the ACSCII encoding

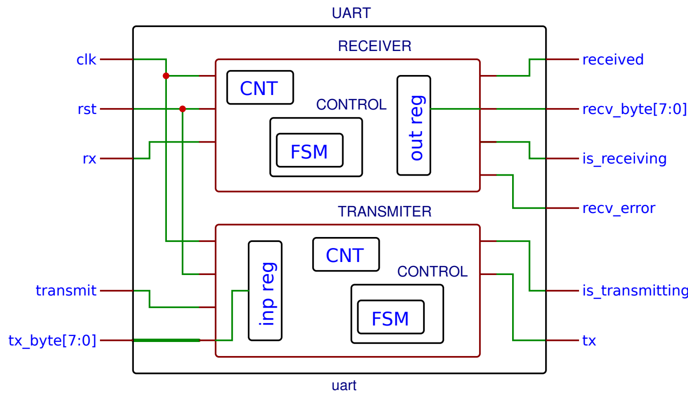

Fig. 2. Schema of a simple UART design

in and out of such a state where a part of the circuit is not used.

To illustrate the concept we use a simple UART (Universal Asynchronous Receiver/Transmitter) circuit. UART is a common digital design that can be used to interface fast electronic circuitry (from now on ‘a computer’) with slow peripheral devices. A device is connected to the UART by a serial line that allows to communicate data in a sequential fashion (one bit at a time). The communication follows a specific protocol. When there is no data transfer, the value on the serial line is set to logical 1. When the sender wants to transfer data, first it sets the serial line to 0 (start bit) for one time frame and then follows by transferring one byte of data 1 bit per frame (8 frames). Subsequently, the protocol requires the sender to insert at least two frames with logical value 1 (stop bits). Then the sender can continue sending another byte (starting with start bit) or stays idle until another transmission is needed. Fig. 1 depicts a transmission of one byte via serial line.

The implementation of UART that we use is based on an open source design available at “http://opencores.org/project,osdvu”. It is a simple implementation that supports only the core functionality. The design (Fig. 2) is divided into two main modules.

The `RECEIVER` facilitates the communications sent from a peripheral device to a computer. It listens at the serial input line `rx`. It is idle until the peripheral sets the line to 0 announcing a start of a transmission. During the next 8 frames the UART samples value from the serial line in the middle of the time frame multiple times in order to minimize possible errors. The collected bits are placed in the output register `rx_byte`. After a whole byte is received, the availability of new data is signaled to the computer by raising flag `received` (which will typically generate an interruption in the computer). If the communication protocol is not followed (e.g. missing stop bits), the UART goes to an error state (signaled by raising flag `recv_error`). The UART recovers from the error by waiting for a long time (sufficient to perform multiple transmissions) and then returning to the initial state.

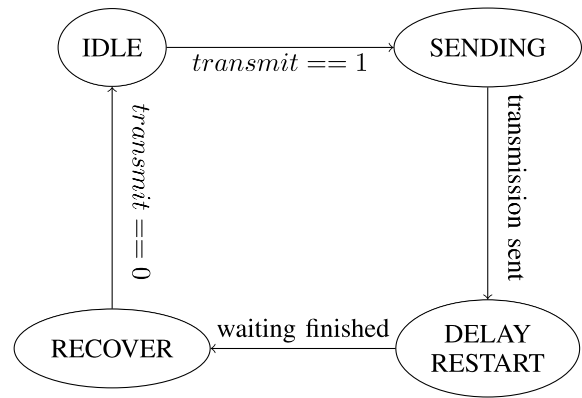

Fig. 3. UART transmitter control FSM

The `TRANSMITTER` can be used by a computer to send data to a peripheral device. The computer fills the input bus `tx_byte` with a byte of data that it wants to transmit. Then it sets `transmit` to 0. The UART transmits the byte to the serial line `tx`. The UART utilizes the flag `is_transmitting` to announce its current state: 0 signalizes to the computer that the UART is ready to transmit another byte.

Both the `TRANSMITTER` and the `RECEIVER` are divided into three main register subgroups, which can be implemented as HDL modules but do not have to be if the designer does not aim for a maximal modularization of the design.

1) `CONTROL` is a unit, which controls the operation. It contains a register bus that represents the control `FSM` plus a few additional control registers.
2) `CNT` is a counter that is used to measure the time elapsed in various `FSM` states.
3) `inp_reg/out_reg` are register buses, which are used to store the received byte in case of the `RECEIVER` and the byte that is to be transmitted in case of the `TRANSMITTER`.

Consider the control automaton of the `TRANSMITTER` module on Fig. 3. Every state corresponds to a mode of operation of the `TRANSMITTER` and not all submodules of the `TRANSMITTER` are used in every state:

IDLE: Initial state before the beginning of a transmission. The state is left when the `transmit` flag is raised. The input `tx_byte` is propagated into the input register together with changing the FSM state. All registers in the `TRANSMITTER` are stable in this state.
SENDING: The transmission is performed in this state. Some registers in `CONTROL` and `CNT` submodules are used extensively.
DELAY RESTART: The `TRANSMITTER` waits for a predefined time after the transmission is finished. `CNT` is used.

RECOVER: The design waits in case the `transmit` flag was not set to 0. No registers are used beside those in the `FSM` upon leaving the state.

The registers that are not changing value in a particular mode can be clock gated. For instance, we can clock gate the entire transmitter when it is in the mode associated with the IDLE state of the `FSM`. However the state cannot be used as a clock gating condition directly, because clock gating would prevent the state of the `FSM` to be changed and the design would remain in IDLE state indefinitely. Instead, we identify the conditions associated with entering and leaving the state. Particularly, the condition associated with entering the IDLE state and starting the clock gating would be "`FSM==RECOVER && transmit==0`". The condition for disabling the clock gating and leaving the `FSM` state would be simply "`transmit==1`".

We can find similar conditions for every state in the `FSM` and use them for clock gating of registers that are not used in that state. Furthermore, some registers are used only in rare situations. For instance the value in the input registers can be changed only when taking the transition from IDLE to SENDING. Hence the clock enabling condition for this register group would be "`FSM==IDLE && transmit==0`" and the clock disabling condition would be"`FSM==SENDING`". This way we can activate the input registers clock only when it is needed (one out of multiple thousands clock cycles).

The conditions that we identified are defined only on a few signals, so the additional clock gating logic will be simple and they are valid for relatively big design blocks (more than just one bus or a few registers), so they are more coarse than typical local clock gating conditions.

Ideally, such intermediate level coarser conditions should be identified by a designer. Then, if their validity is not obvious, our framework can be used to formally verify it. However, because of extreme complexity of current designs, reuse of old modules and purchased IPs, it is usually not possible for one person to have a detailed knowledge of all low level aspects of a design. For instance, if the UART was used in practice, it would be probably just a small part of a much bigger system and the designer might reuse the UART from some old project. In such case he would probably regard it as a black box component and he would not have time to perform a detailed manual analysis of the internal implementation details.

Hence, intermediate level clock gating conditions are often not recognized by designers and it is very useful to have an automatic tool to identify them. Furthermore, we observed that activity of some modules is not always triggered by a static condition like a state of a bus (representing an FSM state for instance) or a signal but often the activity triggering event corresponds rather to a transition. For instance, setting an instruction register to a particular value triggers the activity, which can continue for some time even after the value of the register is changed. This often occurs in cases in which some handshake mechanism is implemented.

We introduce the concept of *idle modes* and *activity triggers*. An idle mode of a sub-circuit is a mode in which it is safe to clock gate the sub-circuit. It will be associated with two events: a *stop* event, which forces the design to enter a idle mode, and a *start* event, which happens always before the exit from the idle mode. The combination of *stop* and *start* can be used for clock gating and is called an *activity trigger*.

Furthermore, it is possible that it takes some time for the module to enter the idle mode after an occurrence of a stop event. In this case, the module becomes stable only after a given number of cycles. This parameter is called *offset* and is also a part of an activity trigger.

IV. Formal Modeling and Verification

A digital circuit consists of various combinatorial logic gates and sequential memory elements. For the purpose of formal modeling we will consider the state of the circuit to be the current Boolean valuation of *all* the signals (inputs, gate outputs, memory elements). The dynamics of the circuit will be modeled by a transition relation that conveniently hides all the structural details. This definition differs from traditional circuit model in the sense that the states are not restricted to the states of the sequential elements. This assumes some discrete time domain and instantaneous signal propagation, a reasonable assumption for well-timed synchronous circuits. For simplicity, we assume that every design has one initial state. In reality, although the initial state of a circuit is typically undefined, a unique state is often reached via a reset sequence.

Definition 1: A digital circuit (design) is a tuple $(X, \mathcal{Q}, T, q_0)$, where

- X is a finite set of variables that correspond to signals in the circuit.
- $\mathcal{Q}$ is the state space of the circuit. Each state is of the form $x : X \to \mathbb{B}$ and has to satisfy combinatorial constrains imposed by the structure of the circuit.
- $T \subseteq (\mathcal{Q} \times \mathcal{Q})$ is a transition relation describing the circuit dynamics.
- $q_0 \in \mathcal{Q}$ the initial state of the design.

The semantics of the circuit, the execution traces it generates, is defined using paths in its automaton model.

Definition 2: An *execution trace* σ of a circuit $D = (X, \mathcal{Q}, T, q_0)$ is a sequence $q[0], \ldots, q[\tau]$ of states, where

- $q[0] = q_0$,
- For all $1 \geq i \geq \tau$ it holds that $(q[i-1], q[i]) \in T$.

We use notation $x[t]$ to refer to the value of a signal variable $x \in X$ at the time t when the execution trace is clear from the context.

We will use *linear time temporal logic* (LTL) with past operators (PLTL) [16], [17] to specify execution traces. This allows us to define the validity of activity triggers as properties that must hold for every execution of the circuit. Using past operators is more natural in our context than standard LTL as the clock gating conditions need to refer to the past behavior of a circuit.

Definition 3: A PLTL formula over a set of variables X is defined inductively as follows.

1) For $x \in X$, x is a PLTL formula.

2) Let Ψ and Φ be PLTL formulae. The following are also PLTL formulae:
   - $\neg\Phi$
   - $\Phi \wedge \Psi$
   - $\ominus\Phi$ (previously Φ)
   - $\Phi\mathcal{S}\Psi$ (Φ since Ψ)

The rest of the Boolean and temporal operators can be derived from $\neg, \wedge, \ominus, \mathcal{S}$. We also introduce a shortcut for a multiple application of the $\ominus$ operator, letting $\ominus^0\Phi = \Phi$ and $\ominus^i\Phi = \ominus\ominus^{i-1}\Phi$ when $i{>}0$.

Definition 4 (PLTL Semantics): Satisfaction of a PLTL formula Φ by a trace σ at a time t, denoted by $(\sigma, t) \models \Phi$, is defined as follows

$$\begin{array}{lcl} (\sigma,t) \models x & \Leftrightarrow & x[t] \quad \text{(in the context of } \sigma) \\ (\sigma,t) \models \neg\Phi & \Leftrightarrow & (\sigma,t) \not\models \Phi \\ (\sigma,t) \models \ominus\Phi & \Leftrightarrow & t{\neq}0 \text{ and } (\sigma,t{-}1) \models \Phi \\ (\sigma,t) \models \Phi\mathcal{S}\Psi & \Leftrightarrow & \exists j, 0 \leq j \leq t \text{ such that } (\sigma,j) \models \Psi \\ & & \forall i,\ j < i \leq t \text{ such that } (\sigma,i) \models \Phi. \end{array} \quad (1)$$

Satisfaction of a formula by an entire trace is defined as satisfaction (backwards) from its last state.

$$\sigma \models \Phi \Leftrightarrow (\sigma, |\sigma|) \models \Phi \quad (2)$$

We will use PLTL to define stability and other properties related to activity triggers.

Definition 5: Let $D = (X, \mathcal{Q}, T, q_0)$ be a design and let $\sigma = q[0], \ldots, q[\tau]$ be an execution trace. The stability of a signal $x \in X$ is defined as follows

$$\mathit{stable}(x) = (x \wedge \ominus(x)) \vee (\neg x \wedge \ominus(\neg x)). \quad (3)$$

Furthermore, we can naturally extend the notion to define stability of an arbitrary set of signals $M \subseteq X$.

$$\mathit{stable}(M) = \bigwedge_{x \in M} \mathit{stable}(x). \quad (4)$$

This notion is important for clock gating because if M represents a set of sequential elements of the original circuit, these elements can be gated in clock cycles in which $\mathit{stable}(M)$ is satisfied.

We can now define activity triggering events that control the transition of sub-circuits into and out of idle modes where all their registers are stable. Such events can be, in principle, sequences specified by any PLTL formulae, but for statistical detection we restrict them to be transitions on a set of signals.

Definition 6 (Signal transitions): Let σ be an execution trace and let $(x_1, \cdots, x_n)$ be an ordered set of signals. Let $b_1, \cdots, b_n, b'_1, \cdots, b'_n \in \mathbb{B}$. We say that $(x_1, \cdots, x_n)$ makes a transition $(b_1, \cdots, b_n) \to (b'_1, \cdots, b'_n)$ at time t if $(\sigma, t) \models \bigwedge_{i=0}^{i=n}((x_i \leftrightarrow b'_i) \wedge \ominus(x_i \leftrightarrow b_i))$.

An *activity trigger* for a module M consists of two events α and β such that α initiates the activity of M and β stops it. Typically, a module needs a few cycles to stabilize after the occurrence of β. The length of the stabilization period (*offset*) is denoted d. When M is active, the occurrence of β should enforce M to become idle within d time steps unless it has been aborted by an occurrence of α. This condition, whose satisfaction initiates clock gating, is expressed in PLTL as:

$$\ominus^d\beta \wedge \bigwedge_{i=0}^{d} \ominus^i\neg\alpha.$$

Clock gating can be continued as long as the start event α has not been observed.

Definition 7 (Valid activity trigger): Let α, β be signal transitions (or more generally any PLTL formulae) and d a positive. A triple (α, β, d) is a valid activity trigger for a module M if all traces of the circuit satisfy

$$\neg\alpha\mathcal{S}\left(\ominus^d\beta \wedge \bigwedge_{i=0}^{d} \ominus^i\neg\alpha\right) \Rightarrow \mathit{stable}(M), \quad (5)$$

Formal verification

A key feature of our methodology is that it can be formally verified whether a given candidate is a valid activity trigger for a module or a set of registers. To prove that an activity trigger (α, β, d) is indeed valid, we need to show that (5) holds for all possible behaviors of the system.

To implement this check using a circuit-oriented model checker we encode the condition $\mathit{stable}(M)$, which monitors the stability of a set of registers in the sense of (3) and (4), as an additional signal which is low if the value of some registers changed in the last clock cycle. Change detection for a simple memory element is realized by applying 'exclusive nor' (XNOR) to its input and output. For a more complex register, we store the previous value in an auxiliary register and apply the XNOR to the two registers. Combining the results for all registers we obtain the required signal.

Using this new signal we construct an observer automaton (Fig. 4) which corresponds to the temporal formula (5). When this automaton is composed with the circuit automaton, it will enter the property violation state and only if (5) is violated. Thus validity of candidate activity triggers amounts to non-reachability of the error state, which can be proved using any formal verification tool capable of proving safety properties. For the tools used in our implementation see Section VI.

If the activity triggers are valid, the monitor automaton can be used, in principle, to control clock gating, enabled exactly when it is MODULE IDLE state (Fig. 4). This application of the monitor is, however, possible only if the inputs α and β of the automaton are not affected by the clock gating itself.

Constraints

Often it is not possible to prove the validity of an activity trigger under every possible input, but it is still valid under all input scenarios that can occur in our system. For instance we assume by default that the reset signal is used only at the beginning of any possible execution to initialize the design. A non-default assumption can for instance fix the value of some configuration registers or force the input signal values to follow some protocol. The UART's communication protocol can be an example of a complex input assumption. Another assumption for UART is that the clock driving the input is much slower than the clock driving the UART. Most of modern

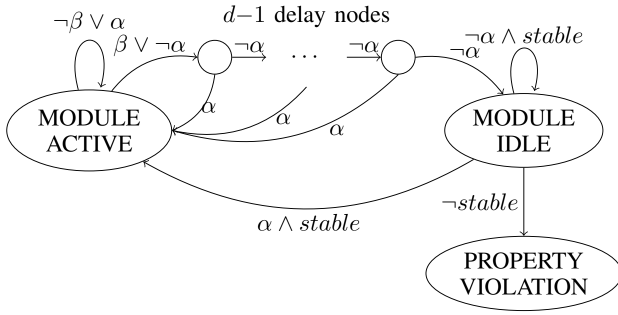

Fig. 4. An automaton for checking validity of activity triggers (5).

electronic designs are configurable and are used as a part of a bigger system that implies constraints on the input, therefore it is important to have a mechanism that allows to specify such constraints so that only constraint-satisfying behavior is considered by the reachability analysis. In our tool, we use user-defined constraints during the formal check.

It is typical that industrial designs don't contain all the constraints necessary for a successful proof of an otherwise valid trigger. For such cases we propose an *iterative constraint refinement* procedure – when the trigger is disproved, the designer can manually examine the provided counterexample. In case the counterexample does represent a behavior that cannot occur in the circuit, the designer can add a constraint and run the formal check again. This can be repeated until we find a realistic counterexample or until the trigger is proven.

V. Statistical trigger detection

Power optimization at the architectural level is sometimes performed by design teams as they analyze the simulation to find potential optimizations (e.g., idle periods where the clock is still active) or power bugs in the RTL (e.g., cases where the idle state of a block is badly implemented and the block still has activity). As the design scale increases this methodology may get difficult, prone to error, and time consuming. Also the use of third party IPs for which little knowledge on the architectural properties is known may complicate the analysis. The methodology we propose starts with automatically analyzing the simulation file in a first step, in order to identify the design blocks of interest (i.e. with significant potential power savings) as well as to point out the interesting activity events and signals that trigger them. In a second step, we provide a formal flow which can verify clock gating conditions based on activity triggers. This may be done either on the triggers found during our automatic simulation analysis or on trigger conditions directly provided by the user. When the verification is successful, the clock gating condition is proved to be safe in the sense that a whole block may be clock-gated without breaking the functional behavior of the design.

In order to detect activity triggers we use a heuristics-based statistical approach. The idea is based on a hypothesis that the activity triggering events can be found in a short time window before and after the idle periods in the simulation traces. The user performs an RTL simulation of the design, based on a set of test vectors that should represent standard behavior of the circuit. Such vectors are commonly used by designers, e.g., for functional verification or power estimation. The changes of signal values during the simulation are stored in a file (VCD or FSDB). Then our detection tool performs the following steps in order to find a set of events that have a good chance of being activity triggers.

Fig. 5. Idle periods in a simulation

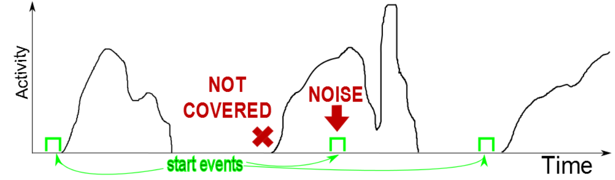

Fig. 6. Coverage and noise

1) Design decomposition: First, we need to define the set of sub-circuits for which we want to analyze the activity. In general, a sub-circuit can be any set of registers chosen by some meaningful strategy. Typically, these are RTL modules or user-defined register groups. For simplicity, we refer to chosen sub-circuits as *modules*.

2) Idle periods detection: For every module we analyze the activity. We identify all the idle periods – i.e., intervals in which no registers were switching. For such intervals, there will be a minimal length in order to remove very short periods that may be noise.

3) Finding potential triggers: For each idle period, we look for signals having a transition during a short window before/after the period. The size of this window is a parameter to the procedure. For instance, if a signal goes $0 \to 1$ before every idle period, it is a good stop event candidate. We also analyze transitions of small buses. For instance if a bus always goes $0001 \to 0010$ after an idle period, it is a start signal candidate. This is especially useful for buses that hold an FSM state.

To distinguish between potential candidates, we introduce two statistical indicators for each transition, *coverage* and *noise*. The *coverage* is the percentage of the idle periods that may be affected by a transition (i.e., the transition is observed before the period). The noise is the percentage of transition occurrences outside of the window before/after the idle period (Fig. 6). Candidates with high coverage and low noise are considered to be strong candidates, even though not all of the true triggers need to have 100 % coverage (not every idle period has to be controlled by that trigger) and 0% noise (e.g., a start event happening within an active period is possible but has no effect). However, experiments show high/low coverage/noise to be a good indicator.

4) Filtering, ranking and reporting: We employ a number of heuristics to remove weak candidates or candidates that are

strongly depending on each other.

- We filter out all the transitions that don't have a good coverage and noise rankings.
- We perform a structural check to remove signals that are not in the fanin/fanout of the module. A start event related to a signal that cannot reach the module in its fanout cone is eliminated (as it cannot influence the module). Respectively, a stop event related to a signal that doesn't have the module either in the fanin or fanout cone is eliminated (it cannot influence or be influenced by the module).
- We remove highly active signals (like clocks), if they were not already removed due to a high noise ranking.
- We compute shortest path from the signal to the module. This can be used to filter out candidates that cannot have a cau sal relation to the activity event.
- We eliminate the candidates that are already used for clock gating.

To illustrate the detection we will use the UART `RECEIVER` module. On Fig. 7 we can see the combined activity of all nets in the receiver module during a test execution. It contains blocks of activity that represent receptions of one transmission block each. After the statistical analysis we find that the transition 000→001 of the `FSM` is reported as a possible start event and the transition 0→1 of the signal `received` is reported as a possible stop event. Both events have coverage 100% as the receiver's `FSM` is moving from 000 (wait for a new transmission) to 001 (start processing a new transmission) before every block of activity (transmission processing) in the activity graph (Fig. 7) and we can see the `received` flag being raised at the end of each activity period. Both events have noise 0% as neither of them occurs in any other place in this simulation trace.

The pair ($\texttt{FSM}: 000 \rightarrow 001$, $\texttt{received}: 0 \rightarrow 1$) forms a valid activity trigger (with an offset 1), which is then formally proved by the verification tool. However, the fact that the start signal is defined on the `FSM`, which is contained in the module, means that we cannot use it directly for clock gating of the entire module (as the `FSM` would be clock gated and the module would never be activated again). We can use it to correctly clock gate all the registers other than `FSM` though, which represent most of the module's power.

Still, there exists an event which is a more desirable candidate for clock gating. The UART's communication protocol requires that the input serial line `rx` is set to 0 at the beginning of an incoming transmission. This event can be used to clock gate the whole receiver module, including the `FSM` and it is defined on one wire, hence the added clock gating circuitry will be smaller. However, `rx` is used not only to start the transition, but also to communicate the content of the transmission, hence it switches not only before the active periods but many times also within the active periods. Therefore the noise ranking of `rx` is very high and the event is filtered by the detection engine. An experienced designer can easily recognize that the reported $\texttt{FSM}: 000 \rightarrow 001$ event is triggered by $\texttt{rx}: 1 \rightarrow 0$ and use the latter for clock gating, however this phenomenon represents a room for improvement of the current detection heuristic that will be addressed in future research.

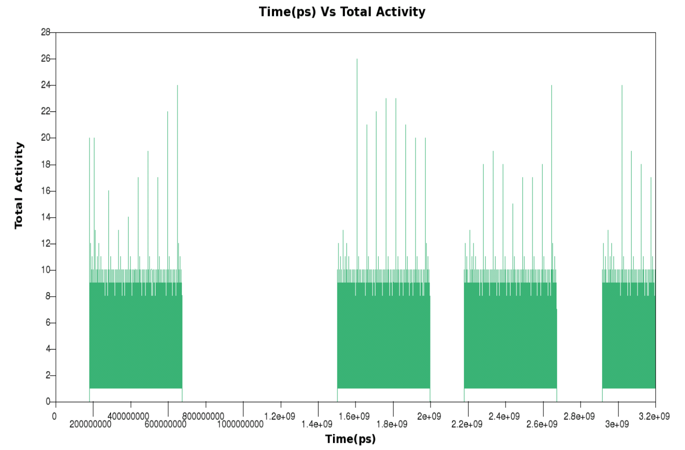

Fig. 7. Activity of UART receiver module

VI. Application flow

We provide a methodology and a tool for an extensive analysis of activity triggers in digital designs. In our flow, we expect the user to provide HDL files describing the design, the constraint definitions (clocks and resets definitions, input constraints, etc.), and simulation data (VCD, FSDB). Our tool provides two core functionalities:

Trigger detection: The input is the design HDL and simulation data. The data is analyzed by our statistical engine to infer a set of events that seem to be triggering activity. These triggers need to be verified by a designer or by the formal verification tool.

Formal verification tool: The input consists of an HDL description of a design, an activity trigger specification and a time budget for verification. The trigger specifications can be supplied manually or taken from the trigger detection engine. The tool uses ABC's [18], [19] PDR engine (property directed reachability [20], [21]) in attempt to prove the validity of the trigger, while running BMC (bounded model checking [22]) and Rarity Simulation [23] engines in parallel, which allows to disprove some incorrect triggers faster than only with PDR. The result of the formal check can be either "VALID" if the trigger was proven, "INVALID" if the verification engine found an counterexample, which is saved, or "TIMEOUT" in case that the time budget was exceeded.

We propose a semi-automatic flow for the designers to be able to exploit the clock gating opportunities maximally.

1) Run statistical trigger detection on trace files generated during the simulation. The tool will create a list of possible activity triggers for every module where some promising candidate is identified according to the heuristics described in Section V.

2) Run the formal check with a reasonable time budget for every detected candidate. The candidates that are proven at this stage can be used directly for clock gating.

3) For the candidates that are disproved or which were not proven within the allocated time, these that have a high potential power reduction (number of affected registers) should be selected for a manual examination. Sometimes it can be

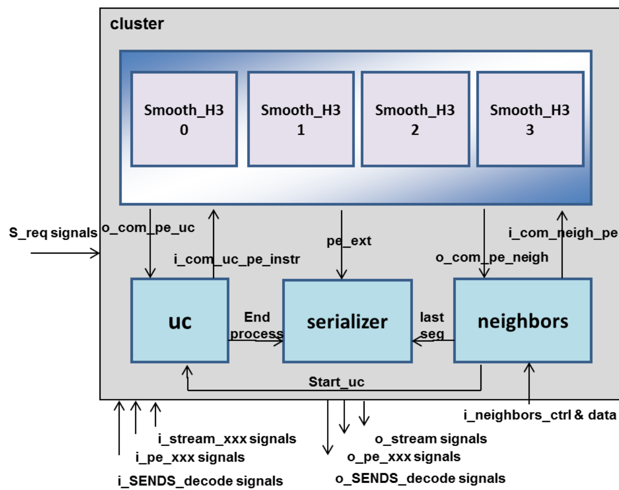

Fig. 8. A video processing architecture

obvious that a trigger candidate is correct and the designer may use it directly based on his expert assessment.

4) Otherwise, the designer should examine the counterexample trace of the failing triggers. If he realizes that the trace corresponds to a behavior that would not be possible in real execution of the circuit, he can specify input constraints as System Verilog Assertions (SVA [**?**]) and add them to the design files. This process of adding constraints can be repeated multiple times. Constraints that fix some configuration inputs or registers are usually necessary for the formal check to be successful if the design is configurable.

5) In case the designer suspects that the activity is limited to just a part of a module, it is possible to run the verification for a specified group of registers only.

For reasons explained before, the detection method performs best when the design is highly modularized. For instance, in the original version of the UART example, the `receiver` and the `transmitter` functionalities are implemented in the same Verilog module. This leads to a strong interference between different scenarios, e.g., the active periods resulting from reception in UART are marked as 'not covered' when evaluated w.r.t. a trigger that is valid for the transmitter. Furthermore, it is not possible to prove validity of any trigger, without manually specifying the group of registers for which the trigger is valid (some triggers are valid for the `receiver`, some for the `transmitter`, but not for both). Highly modularized design, where `receiver`, `transmitter` and ideally also `FSM`, `CNT` (etc.) allows the tool to detect and verify opportunities that would be otherwise missed due to activity interference.

VII. Experimental Results

In order to demonstrate the efficiency of our methodology we applied it to a real case study: a cluster for video processing called SENDS (Smooth ENgine for Data Stream, Figure 8).

SENDS is part of a larger, clusterized and configurable architecture for video processing. This architecture is coupled with a CMOS image sensor integrated on the same chip, from which it takes a pixel stream as input. The circuit can be configured to perform (concurrently or sequentially) several low-level processes such as various filtering, contrast enhancement, denoising or YUV to RGB conversion. These computations constitute a pre-processing step for more involved image processing such as complex features extraction. This kind of architecture is used, for instance, in smart cameras.

The SENDS design is organized around a FSM controlling the datapath scheduling and reception/emission of the pixel stream. The datapath can be configured. In this particular case it features smoothing engines, which operate concurrently. The IP is connected to its interface through a bus, and it has several inputs for configuration, control and synchronization, as well as data/control inputs related to the pixel stream. The data consists of columns of pixels each coming from the sensor. A pixel has 3 components: Red, Green, and Blue, 8 bits each. The processing is performed on a pixel window, where the pixel in the center is computed based on neighboring pixels. The number of concurrently processed windows corresponds to the number of processing units available in the architecture.

The pixel stream is first collected in the neighbors module. When the neighbors register file has been filled with sufficient data, it notifies the unit control (uc) by raising the signal `start_uc`. Then the unit control sends appropriate instructions (in particular pixels addresses to be processed) to the `macro_pe` module. The `macro_pe` triggers the data retrieval in the neighbors register file and launches the smoothing process. When the computation is finished it notifies the uc which in turns notifies the serializer using signal `end_process`. The filtered pixels are then sent by the `macro_pe` to the serializer which outputs the result.

There are several activity triggers which may be identified for different modules of the SENDS architecture. In particular, the smoothing engines are all controlled by the pair of signals (`start_uc`, `end_process`), and they could be clock gated whenever they are non active, waiting for sufficient pixels to come into the neighbors so that a new pixel window may be processed. Given that the smoothing engines consume the biggest part of the dynamic power, this activity trigger should be a valuable power reduction opportunity.

We applied the methodology described in this paper to the SENDS design, using different test benches corresponding to different image processing algorithms. Our statistical detection was able to correctly identify the activity trigger (`start_uc`, `end_process`) for the smoothing engines in all the test benches, as well as other activity triggers for the unit control and the serializer. We also applied the detection algorithm to the top level design composed of 4 SENDS clusters working in parallel. We were able to detect the corresponding pair for each individual cluster in the design, which shows that the statistical detection scales up to a higher design level. For each of those experiments, the detection took less than 20min of computation.

For the case of a single SENDS cluster, we clock gated the `macro_pe` using the detected activity trigger. This was straightforward given that a single signal transition controls the wake-up (`start_uc`: $0 \rightarrow 1$) and the shutdown (`end_process`: $0 \rightarrow 1$). We did this experiment for two different image processes using 3x3 and 5x5 pixel windows.

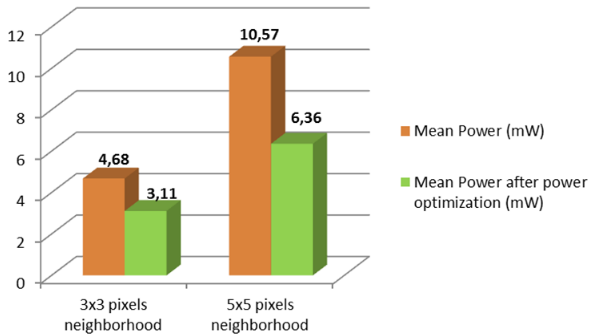

Fig. 9. Mean power consumption depending on the pixel neighborhood width before and after power optimization with the new elaborated optimization rule

design	power	power covered	# registers	reg-covered
1	4.169 mW	75.84%	26680	37.17%
2	7.163 mW	54.93%	9128	56.38%
3	0.479 mW	49.62%	1352	82.84%
4	7.145 mW	49.47%	9128	13.56%
5	5.314 μW	31.04%	326	33.74%
6	0.606 mW	16.30%	2070	6.96%
7	8.891 mW	15.58%	690	28.70%
8	92.491 mW	6.77%	30520	4.65%
9	55.851 mW	4.54%	107848	8.14%
10	92.444 mW	2.87%	114546	1.56%
11	1.430 μW	1.61%	162	14.81%
12	4.079 mW	0.70%	5292	0.15%
13	149.955 mW	0.61%	111012	1.11%
14	400.937 mW	0.31%	160530	1.58%
15	8.013 mW	0.05%	43504	0.01%
16	2.799 mW	0.02%	38158	0.04%

TABLE I. AUTOMATED RUN – FORMAL VERIFICATION RESULTS

We then estimated the power savings by running an RTL power estimation tool before and after the modification (technology is CMOS TSMC 45nm worst-case). For the test benches that we used, the clock gating brought power savings of 33% for the 3x3 case and 40% for the 5x5 case.

Of course, the activity trigger for the smoothing engines can also be identified manually with a detailed understanding of the SENDS architecture and how it is used and configured at the top level. However, such power optimization is typically finer grained that what is performed manually at the architectural level, and it could easily be missed. This optimization is somehow at an intermediate level between what is done at the architecture/system level (such as shutting down a whole cluster when it is not used) and what is done at the local register level such as ODC/STC. Furthermore we ran a commercial tool for power reduction using ODC/STC based methods on the SENDS design, and it was unable to find a simple and effective condition like the one we find with our methodology. Some clock-gating opportunities were identified at the register level, but the enable conditions were really complex, not comprehensible to designers that know the architecture. In total ODC/STC based power reduction brought only 1 % power savings for the same test benches.

Concerning the formal part, the initial result for the activity trigger (`start_uc`, `end_process`) was unproved as we obtained a counter-example. Looking at the violating trace we could quickly observe that the behavior was spurious. To eliminate the behavior we added the following two SVA constraints: 1) A constraint for the architecture configuration (size of pixel window, number of parallel smooth engines to use etc.). The constraint basically consisted in setting the value of a configuration register appropriately. 2) A constraint to limit the pixel stream throughput. This was necessary to avoid the behavior where the system gets overloaded with more data than it can handle (in practice, the pixel stream is coming from the camera which has a limited frame rate). To set this constraint we specified a minimum amount of time between changes at the neighbors module input. With these constraints our tool was able to formally prove the activity trigger property in about 5 hours of running the formal engine. Therefore, in addition to detecting the optimization automatically, we were able to provide a strong guarantee that the clock gating based on the activity trigger does not break the functional behavior of the IP.

Furthermore, to explore the applicability of our method to a wider set of designs, we tested our tool on a set of 49 industrial designs in a fully automated mode (i.e., we run the formal check on all the candidates generated by the detection tool without specifying any SVA constraints). We were able to identify a good number of triggers using the statistical detection for most of the designs. On the other hand the results of the formal check were not conclusive as we got only a few proofs but also some failures and many timeouts. Given our experience on the SENDS design and other test cases, this can be explained by the fact that SVA constraints are usually needed before we can prove the activity trigger property, at least when the activity trigger applies to a medium-size block. Additionally, in this fully automatic experiment we had to set quite a small timeout for the formal check of 15 min per property. We did a more in depth analysis for few designs where the activity triggers looked promising. In particular we could see some triggers related to FIFOs (e.g., when a FIFO is full or empty this entails that some other modules get blocked), and DMA (e.g., a module is waiting for a DMA request to be processed).

We present in table I the formal verification results for the designs whose triggers could be proved automatically. The ‘power’ column of the table indicates the dynamic power consumption of the circuit as computed by a power estimation tool and the ‘power covered’ column show what fraction of this power is consumed by the modules for which we found and formally verified some activity triggers. Note that this is an upper bound on the power reduction and the real savings depend on the relative duration of the idle periods. The ‘#registers’ column contains the total number of registers in the circuit while the column ‘reg-covered’ shows the percentage of registers that fall within the scope of the triggers.

As discussed before, the percentage of automatically proven triggers is quite small compared to all opportunities detected by the statistical engine. We learned from the SENDS experience that in most cases additional constraints are needed for a successful formal proof. However, the main observation to be taken from Table I is the relatively good scalability of the formal verification. We are able, within 15 minutes limit, to formally prove some activity triggers for design blocks that contain thousands of registers and consume enough dynamic power to be interesting for clock gating.

VIII. Conclusion and Future Work

We developed and implemented a method for semi-automatic detection of activity triggers in electronic circuits as well as formal verification of their correctness. We demonstrated that the method can achieve a significant power reduction on an image processing architecture with which we had a good knowledge of design intent. We also applied the method in a fully automatic manner to larger set of industrial designs with some more modest results. The lesson from this experience is that a proper modularization, specification of input constraints and some intervention of a human designer are in most cases crucial for a successful application of the method. The major current limitations of the approach are:

1) The sensitivity of the statistical detection procedure to interference, especially in designs that are not highly modularized;
2) The need for extensive simulation data in order to detect triggers. This prevents the use of the method in very early stages of the design flow when such simulation data may not be available;
3) The usual limitations related to the complexity of the formal check.

Addressing these issues is part of ongoing future work. We will use automatic modularization, e.g., every bus analyzed separately, to cope with the noise that comes from a small part of the analyzed module. Furthermore, we plan to improve the trigger detection engine, possibly using machine learning (as in [12], [13]) or design intent detection heuristics inspired by [15]. Some triggers may be purely functional, independent on the environment. We work on structural analysis of such triggers that would allow us to identify them without simulation, much earlier in the design flow, contributing to the general 'shift to the left' in the silicon industry.

The success of the formal part strongly depends on the designers' readiness to provide detailed input constraints that can be quite complex. We observed that some natural constraints such as alternation of start and stop events are often present, and hence can be used even if they are not specified. If the formal proof succeed under this assumption, it can be reported as a hint to the user. More constraint hints can be obtained by data mining the simulation traces.

References

[1] H. Jacobson, P. Bose, Z. Hu, A. Buyuktosunoglu, V. Zyuban, R. Eickemeyer, L. Eisen, J. Griswell, D. Logan, B. Sinharoy, and J. Tendier, "Stretching the limits of clock-gating efficiency in server-class processors," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2005, pp. 238–242.

[2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.

[3] L. Benini and G. DeMicheli, *Dynamic power management: design techniques and CAD tools*. Springer Science & Business Media, 2012.

[4] L. Benini, G. De Micheli, E. Macii, M. Poncino, and R. Scarsi, "Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers," *ACM Transactions on Design Automation of Electronic Systems*, vol. 4, no. 4, pp. 351–375, Oct. 1999. [Online]. Available: http://portal.acm.org/citation.cfm?doid=323480.323482

[5] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: Pushing power management to logic synthesis/design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 10, pp. 1051–1060, 1998.

[6] P. Babighian, L. Benini, and E. Macii, "A scalable ODC-based algorithm for RTL insertion of gated clocks," in *Proceedings - Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, 2004, pp. 500–505.

[7] R. Fraer, G. Kamhi, and M. K. Mhameed, "A new paradigm for synthesis and propagation of clock gating conditions," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 658–663. [Online]. Available: http://doi.acm.org/10.1145/1391469.1391638

[8] J. Cong, B. Liu, and Z. Zhang, "Behavior-level observability dont-cares and application to low-power behavioral synthesis," 2009.

[9] A. P. Hurst, "Automatic synthesis of clock gating logic with controlled netlist perturbation," in *Proceedings - Design Automation Conference*, 2008, pp. 654–657.

[10] E. Arbel, O. Rokhlenko, and K. Yorav, "SAT-based synthesis of clock gating functions using 3-valued abstraction," in *Formal Methods in Computer Aided Design, FMCAD 2009*, 2009, pp. 198–204.

[11] E. Arbel, C. Eisner, and O. Rokhlenko, "Resurrecting infeasible clock-gating functions," *DAC '09*, p. 160, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1629911.1629957

[12] P. Babighian, G. Kamhi, and M. Vardi, "Interactive presentation: Powerquest: trace driven data mining for power optimization," in *DATE*. EDA Consortium, 2007, pp. 1078–1083.

[13] R. Wiener, G. Kamhi, and M. Y. Vardi, "Intelligate: scalable dynamic invariant learning for power reduction," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*. Springer, 2009, pp. 52–61.

[14] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *Software Engineering, IEEE Transactions on*, vol. 27, no. 2, pp. 99–123, 2001.

[15] S. Hangal, N. Chandra, S. Narayanan, and S. Chakravorty, "Iodine: a tool to automatically infer dynamic invariants for hardware designs," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 775–778.

[16] O. Lichtenstein, A. Pnueli, and L. Zuck, "The glory of the past," in *Logics of Programs*, ser. Lecture Notes in Computer Science, R. Parikh, Ed. Springer Berlin Heidelberg, 1985, vol. 193, pp. 196–218. [Online]. Available: http://dx.doi.org/10.1007/3-540-15648-8_16

[17] Z. Manna and A. Pnueli, *Temporal verification of reactive systems: safety*. Springer Science & Business Media, 2012.

[18] B. L. Synthesis and V. Group, "ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/ alanmi/abc/."

[19] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, 2010, pp. 24–40. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_5

[20] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Formal Methods in Computer-Aided Design (FMCAD), 2011*. IEEE, 2011, pp. 125–134.

[21] A. R. Bradley, "Sat-based model checking without unrolling," in *Verification, Model Checking, and Abstract Interpretation*. Springer, 2011, pp. 70–87.

[22] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS*, 1999, pp. 193–207.

[23] S. Chatterjee, A. Mishchenko, R. K. Brayton, and A. Kuehlmann, "On resolution proofs for combinational equivalence," in *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*, 2007, pp. 600–605. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278631