Reviving Erroneous Stability-based Clock-Gating using Partial Max-SAT

Bao Le[1], Dipanjan Sengupta[1], Andreas Veneris[1,2]

***Abstract*—The conflicting yet increasing demand for high performance and low power in multi-functional chips has pushed techniques for power reduction to the forefront of VLSI design. Although recent developments have automated most of the low power implementations, designers often manually modify the circuit in order to achieve further power savings. This human intervention is often paved with many errors that are bound to typical logic functional failures. Debugging these errors can be a resource intensive process that requires considerable manual effort. This discourages engineers and achieving power savings at the micro level of the design sometimes remains unrealized. This paper proposes a novel debugging methodology to rectify erroneous clock-gating implementations. With the use of Partial Max-SAT, the method localizes and rectifies the design error introduced in the circuit during a clock-gating implementation. The net effect of the proposed methodology leads to shorter debug time ensuring additional power savings. Extensive experiments on benchmark circuits confirm the effectiveness of the approach.**

***Index Terms*—Debugging, Design Errors, Low Power design, Clock Gating, Stability Condition, Max-SAT**

I. Introduction

As low-power implementation is ubiquitous in modern VLSI chip design, designers are under immense pressure to adopt aggressive power-saving techniques. Although progress has been made in automating such implementations, significant manual effort is still required to realize savings. Due to inherent design complexity and the human factor, power-saving methods can introduce design errors. When such errors occur, debugging the design to identify the root cause of failure is an overwhelming and time-consuming process. Consequently, designers often get discouraged and fail to maximize power-savings.

Clock gating [1] is a circuit transformation where the clock input of a register no longer latches in new value but instead holds its current value. This enables power savings as unnecessary switching of gates can be avoided. The enable/control signal to stop the clock is computed in two phases. First, coarse-grained clock-gating is implemented by identifying the idle condition for major functional unit(s) of the design. Next, fine-grained clock-gating is inserted in register(s) by identifying idle conditions that are not visible at the architectural level.

For any register in the circuit, fine-grained clock-gating takes place under the following two situations: 1) the output of the register is not observed at the primary outputs, also known as an *Observability Don't Care (ODC)* [2] condition, or 2) the output of the register retains the same logic value for two or more consecutive clock cycles, a situation widely known as a *Stability condition (STC)* [3]. ODC is a well-studied combinational technique, studied extensively and automatically extracted from steering logic such as multiplexers, tri-state buffers and enable states [2]. In contrast, STC is a sequential technique that must be extracted from the finite state machine of the circuit and thus is much more challenging. Moreover, computing the exact STC is a resource intensive process [3]. Although [4]–[7] attempts to find these conditions automatically in any design, it is much easier for designers, having a detailed understanding of the circuit, to extract the STC and implement it manually for a particular register. Evidently, this process is susceptible to design errors. For this reason, in the remainder of this paper, we focus on debugging fine-grained clock-gating under STC.

STC is implemented by replacing the clock input of the gated register(s) by a new enable signal. The logic used to realize this enable is termed as *clock-gating cone* (CGC). During this process, it is often the case that existing combinational logic must also be modified to achieve the power saving transformation. Functional correctness of the clock-gated circuit is ensured by using Sequential Equivalence Checking (SEC) and/or simulation-based verification approaches. If the verification step detects the presence of design error(s), it returns a counter-example such that at least one output of the clock-gated circuit differs from the expected value for the same input vector. In such a situation today, the root cause of the failure is identified and rectified manually. With increasing design complexity and stringent time-to-market windows, finding such design errors becomes an arduous task. As such, it is common practice for some of the clock-gating implementations to be discarded leading to reduced power savings to what ideally one may be able to achieve.

This work proposes a novel debug strategy for erroneous clock-gating implementations under fine-grained STC. The first step is a light weight checking technique for identifying the relative location of the design error. In this step, the erroneous clock-gated circuit is transformed to a modified circuit that is used to quickly provide an insight to the location of the error - namely the CGC and/or the modified combinational logic. Next, a Partial Max-SAT solver utilizes this localization by the pre-processing step to iteratively identify the set of erroneous gates that may be responsible for the logic failure. Once the set of possible erroneous gates have been identified, a gate-level rectification methodology is devised to correct the error. As a result of the above debug and rectification procedures, more power savings can be revived in the circuit leading to increased power savings.

Previous work [8], [9] focuses on reducing power in a clock-gated circuit by allowing problematic/erroneous clock-gating conditions. In [9] the error effects are canceled out before being observed at the primary output by injecting new clock-gating conditions. In [8], problematic clock-gating conditions are merged by approximation and clustering techniques. As discussed earlier, finding additional clock-gating is an extremely challenging task and merging clock-gating is a complex process, requiring deep understanding of the design. In contrast, our rectification technique refrains from introducing additional clock-gating conditions as well as merging different clock-gating logic. It rather fixes the root cause of the design error. The motivation behind the proposed approach is to retain most or all of the manual modifications in the design while being able to rectify the circuit. Experiments on benchmark circuits show that our proposed method is more robust and the pre-processing step leads to significant reduction in debug runtime. Moreover, 98% power savings can be regained by rectifying the erroneous CGC.

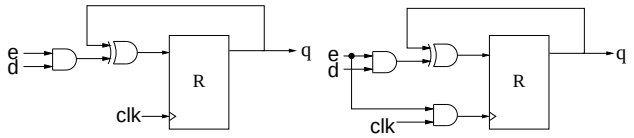
The remaining paper is organized as follows. Section II discusses related work in clock-gating as well as Max-SAT approach for design debugging. Section III proposes the pre-processing step for error localization. Section IV presents the Max-SAT formulation for clock-gating debug followed by the proposed rectification methodology. Experimental results are presented in Section V followed by conclusion in Section VI.

II. Preliminaries

Given a sequential circuit C (C_a), the symbols $X(X_a), Y(Y_a)$, and $S(S_a)$ denote the set of primary inputs (PI), primary outputs (PO), and state elements (flip flops) respectively. The circuit is

[1]University of Toronto, ECE Department, Toronto, ON M5S 3G4 ({lebao, dipanjan, veneris}@eecg.toronto.edu)

[2]University of Toronto, CS Department, Toronto, ON M5S 3G4



(a) Stability Condition Example (b) Clock Gating Example

Fig. 1. A Clock Gating Example

modeled as a directed graph $G(V, E)$, where vertices in V represent the gates and edges in E represent the connections between the gates. The source(s) and the sink(s) of the graph are the PI and PO of the circuit. We say that a path, denoted as $P(u \rightsquigarrow v)$, exists from node u to v, $u, v \in V$, when there is a way to propagate a value at u to v. Let clk denote the primary clock input for circuit C with single clock domain. For each register, the term *clock signal* is used to indicate its dedicated clock input.

Time-frame expansion is a modeling technique for sequential circuits which replicates (i.e unrolls) the circuit for k time-frames, such that the current state of each time-frame is connected to the next state of the previous time-frame. The variable k represents the number of clock cycles that the circuit has been unrolled. For any node z (set of nodes Z), the term $z^t(Z^t)$ represents the corresponding node (set of nodes) in time-frame t during time-frame expansion. The behavior of C during the t^{th} clock-cycle is dictated by the transition relation predicate $T(s^t, s^{t+1}, x^t, y^t)$, which can be extracted from C and encoded in Conjunctive Normal Form (CNF).

A. Clock gating

This subsection describes STC and its application in clock-gating design. As the focus of this work is fine-grained clock-gating under STC, for the rest of this paper, the term clock-gating is used to represent fine-grained clock-gating under STC.

Definition 1 *A register is said to be stable when its output does not change for two or more consecutive clock-cycles. i.e* $q^t = q^{t-1}$.

Figure 1(a) shows a simple example of a register and its stability condition. In this example, register R receives a new data whenever $e = 1$. Otherwise, when $e = 0$, the output q of R remains unchanged. Because q retains its value from the previous time-frame, register R does not perform any switching. As a result of the stability of q, the designer can safely turn off the clock signal of R to reduce dynamic power dissipation.

The clock-gating register using the stability condition from Figure 1(a) is shown in Figure 1(b). In order to turn off the clock signal of R, the designer can connect e and clk with an AND gate. As such, when $e = 0$, the output of the AND gate stays 0 regardless of the clk value; hence, no switching activity happens at the clock signal of R. Note that the output q of R of Figure 1(b) is equal with its counterpart in Figure 1(a) even when its clock signal is turned off. In practice, the clock-gating circuitry uses latches to prevent timing errors such as setup and hold time violations. However, as this work focuses on functional and not timing errors, we simply assume that the data signals do not violate setup and hold time requirements. As a result, these latches play no role in functional debugging and are simply removed during our analysis.

Figure 2 shows an example of a clock-gated circuit. Conceptually, there are *three* components in such a design – namely the existing state elements, combinational circuitry from the original implementation before clock-gating is applied and the additional clock-gating circuitry that computes the new clock signals.

For example in Figure 2, state elements include registers R_1 to R_z where $z = |S|$. The combinational circuitry is shown at the top and it computes next states $\{d_1, ..., d_z\}$ and PO $\{y_1, ..., y_m\}$ from PI $\{x_1, ..., x_n\}$ and current states $\{q_1, ..., q_z\}$. The clock-gating circuitry is shown underneath and it computes clock signals of all registers from existing nodes of the combinational circuitry. For a clock-gating register, a new enable signal is computed and connected with the clock input clk to create the new clock signal for the register.

B. Partial Max-SAT and Application to Design Debugging

This subsection briefly describes Max-SAT and Partial Max-SAT as well as their application in design debugging. It also introduces notation used in the paper.

Given an unsatisfiable CNF instance Φ, the goal of Max-SAT solver is to return an assignment that maximizes the number of satisfiable clauses in Φ [10]. This is a well-known NP-Hard problem. In Partial Max-SAT, the CNF formula Φ is organized as a set of hard clauses which must be satisfied and a set of soft clauses which may or may not be satisfied. The set of hard clauses and soft clauses are denoted as Φ_H and Φ_S, respectively, i.e $\Phi = \Phi_H \cup \Phi_S$. The goal of Partial Max-SAT is to find an assignment that satisfies all hard clauses while maximizing the number of satisfied soft clauses. Recent advancements in Maximum Satisfiability solvers have demonstrated promising results in many industrial applications [11]. Next, we give an overview of design debugging and its Partial Max-SAT formulation.

Design debugging is a process of localizing the error(s) based on the failing counter-example trace. Given an erroneous design C, a counter-example of length k, and an error cardinality N, a debugger returns all sets of N gates that can be potentially responsible for the erroneous behavior exhibited by the counter-example. To simplify our presentation, and without loss of generality, we assume that $N = 1$. However, the proposed techniques are still applicable when $N > 1$ [12]. In [12], a Partial Max-SAT formulation for design debugging is proposed. First, the circuit is unrolled k times using time-frame expansion. The unrolled circuit is then encoded in CNF using Tseitin transformation [13]. Overall, the debugging problem can be formulated as a Partial Max-SAT instance as follows:

$$Debug = (\bigwedge_{t=1}^{k}, T_{en}(s^t, s^{t+1}, x^t, y^t)) \wedge \Phi_X(x^1, ..., x^k) \wedge \Phi_Y(y^1, ..., y^k) \wedge \Phi_{IS}(s^1) \quad (1)$$

where $\Phi_X(x^1, ..., x^k)$ denotes the input constraints provided by the counter-example, $\Phi_Y(y^1, ..., y^k)$ represents the expected outputs and $\Phi_{IS}(s^1)$ is the initial state. Note, Φ_X, Φ_Y and Φ_{IS} are the constraints and thus are hard clauses while T_{en} represents the soft clauses. As the circuit is erroneous, it cannot produce the expected output from the counter-example and therefore $Debug$ is unsatisfiable [14]. For the instance $Debug$, a Partial Max-SAT solver returns a maximal set of satisfied clauses. The complement of such solution presents a set of clauses whose removal satisfies $Debug$. Evidently, these sets of clauses may correspond to the error. As we are only interested in the complement of the solution of the Partial Max-SAT problem, the phrase "Max-SAT solution" in the remainder of this paper refers to the set complement of the Partial Max-SAT solution. Furthermore, [12] proposes many techniques to reduce the complexity of the overall design debugging Partial Max-SAT instance. One such technique is to set trusted modules/components of the circuit as hard clauses in order to prune down the solution space of the instance. Based on this idea, in the following section, we propose a technique

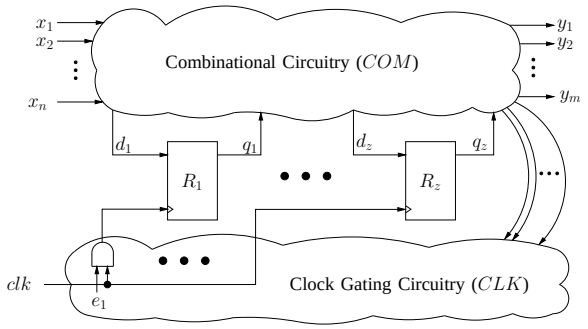


Fig. 2. A Clock Gated Circuit : C

to identify trusted (*i.e.,* not erroneous) components in an erroneous clock-gating circuit.

III. Clock Gating Debugging Pre-processing

Given an erroneous clock-gated circuit C and a counter-example of length k, potential nodes that are responsible for the observed erroneous behavior of C are identified first. Moreover, if the potential erroneous node belongs to the clock-gating circuitry, possible rectifications are derived and ranked based on the power saving level.

The overall flow consists of two phases: pre-processing and debugging. The pre-processing step aims at identifying the erroneous component(s) where the erroneous gate is located. Once the erroneous components are identified, the other components can be set as trusted. It must be noted that the greater the number of trusted components in a design, the easier is the debugging step. The trusted components are encoded as hard clauses in the Partial Max-SAT instance and each solution potentially corresponds to the erroneous gate.

In general, the pre-processing step aims at simplifying the problem by increasing the number of hard clauses. This technique, while does not pose significant overhead, can help in reducing the complexity of the problem considerably. We first formally define different components in a clock-gating design. Then a simulation based method is presented to identify potential erroneous components.

A. Definitions

In this subsection, the components of a clock-gating design are formally defined.

Definition 2 *A node l is called a clock-gating node if every path from l to a primary output contains at least one clock signal. A clock-gating node can be formally expressed as*

$$l \in CLK \Leftrightarrow \forall P(l \rightsquigarrow y)\ (\exists p, p \in CLK \wedge p \in P(l \rightsquigarrow y))$$

By default, all clock signals are clock-gating nodes. The set of all clock-gating nodes is denoted as CLK.

For example, e_1 in Figure 2 is a clock-gating node since every path from e_1 to a primary output includes the clock signal of R_1. By definition, the clock signal of R_z is also a clock-gating node although R_z is not clock-gated.

Definition 3 *A node l is called a combinational node if there exists a path from l to a primary output node that does not contain a clock-gating node. This can be formally expressed as*

$$l \in COM \Leftrightarrow \exists P(l \rightsquigarrow y)\ (\forall p \in P(l \rightsquigarrow y), p \notin CLK)$$

By default, all output nodes are combinational nodes. The set of all combinational nodes is denoted as COM.

The following lemma is a direct implication from combinational node definition.

Lemma 1 *If node l is a combinational node, at least one fanout of l is not a clock-gating node.*

Proof: This is trivial since every path from l to a primary output has to pass through one of the fanouts of l. Therefore, if all fanouts of l are clock-gating nodes then every path from l consists at least one clock-gating node. As a result, l cannot be a combinational node if all of its fanouts are clock-gating nodes. ■

From the above definitions, it is trivial to observe that $CLK \cap COM = \emptyset$. In Figure 2, CLK contains all the nodes in the clock-gating circuitry including enable signals (e.g, e_1) the clock input clk and all the clock signals of all registers. COM consists of all the nodes in the combinational circuitry, including all the inputs $x_1, ..., x_n$, outputs $y_1, ..., y_m$ and register data nodes $d_1, ..., d_z$. As a result of this partitioning, state elements become the remaining component in the circuit. As mentioned earlier, latches are removed from the design and hence, state elements are simple registers.

B. Erroneous Node Identification

This section presents a technique to determine the components in which the erroneous gate resides. First, we demonstrate scenarios in which clock-gating circuitry (CLK) can be trusted. Consequently, if CLK is proved to be trusted, the erroneous component can only be in the combinational circuitry (COM).

To verify the functionality of clock-gating nodes, we construct an enhanced circuit C_{en} from the erroneous circuit C such that C_{en}, while maintaining the functionality of C, does not employ any clock-gating. Figure 3 illustrates how C_{en} is constructed from C in Figure 2. To maintain the functionality of C, C_{en} retains all registers $R_1, ..., R_z$ and the combinational logic COM from C; however, all clock signals of C_{en} are replaced by clk. As a result, none of the clock signals of C_{en} are turned off at any clock cycle. Using C_{en}, we are able to determine whether a modification in CLK can fix the erroneous behavior presented by the counter-example. If a change in CLK cannot fix the erroneous behavior, it can be trusted.

Given a sequential clock-gated circuit C, its enhanced circuit C_{en}, and a counter-example length k, the following theorem gives the condition for which CLK can be trusted.

Theorem 1 *Given a counter-example trace of length k, if C and C_{en} produce the same states and outputs for all time-frame t less than or equal to k, then any changes in gates of CLK of C cannot fix the error in C. This assumption can be formally expressed as:*

$$\forall t \leq k,\ (S^t_{en} = S^t \wedge Y^t_{en} = Y^t) \tag{2}$$

where S_{en}, Y_{en} are states and PO of C_{en}.

Proof: We prove this by contradiction. For the sake of contradiction, let us assume that there exists a circuit C_c, a rectified (correct) version of C, such that C_c is constructed by modifying gates in CLK of C. From the construction of C_{en} and C_c, we know that the combinational circuitry and state elements of C_{en} and C_c are the same as C.

Assume time-frame t is the first time-frame that C_c has a discrepancy from C that eventually propagates to the PO and fixes the erroneous behavior. Consequently, for time-frame $t-1$, the states and outputs of C, C_{en} and C_c are the same. Therefore, we have:

$$(S^{t-1}_c = S^{t-1} = S^{t-1}_{en}) \wedge (Y^{t-1}_c = Y^{t-1} = Y^{t-1}_{en}) \tag{3}$$

We know that C, C_c, and C_{en} share the same combinational circuitry and state elements. Since they have the same inputs from the counter-example and same previous states (as indicated in (3)) their combinational nodes should have the same values at time-frame t. As a result, the discrepancy should originate from a clock-gating node and propagate to one or more primary output(s). According to Defn. 2, the error must propagate through a clock signal of a register. Let this register in C be denoted as R.

Since we know the discrepancy propagates to the primary output through the clock signal of R at time-frame t, the output q^t_c of R_c should be different from its counterparts q^t and q^t_{en}. Otherwise, the difference in the clock signal of R and R_c does not propagate to the output and cannot fix the erroneous behavior as stated:

$$(q^t_c \neq q^t) \wedge (q^t_c \neq q^t_{en}) \tag{4}$$

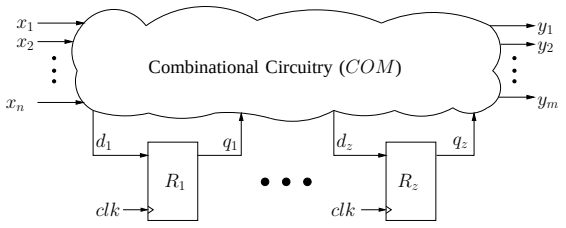


Fig. 3. Enhanced Circuit: C_{en}

Consider the case where clock signal of R_c is turned on. At time-frame t, C_c should function exactly as C_{en}. This is because C and C_{en} share the same PI from the counter-example, same last states as indicated in (3) and the same clock signal for R_c and R (both *ON*). Therefore, q_c^t has the same value as q_{en}^t. This is a contradiction with (4).

Next, we consider the case where clock signal of R_c is turned off at time-frame t and R has its clock signal off. Using the same argument as the previous case, R and R_c have the same output as well, i.e. $(q_c^t = q^t)$. Therefore, this also leads to a contradiction.

Now we examine the case in which the clock signal of R_c is off while the clock signal of R is on. We know that if clock signal of register R_c is turned off under STC, its output should be the same as when the clock is turned on (subsection II-A). However, in this case because of (4), they are different. Therefore, C_c is still erroneous in this case. This also leads to a contradiction since we assume that C_c is a rectified version of C. ■

Corollary 1 *Given an erroneous clock-gated circuit C, its enhanced circuit C_{en}, and a counter-example trace of length k, if C_{en} and C produce the same states and primary outputs for the counter-example trace, then the erroneous gate g is a combinational node.*

Proof: This corollary is a direct implication from Theorem 1. Theorem 1 proves that if C and C_{en} produce the same states/outputs, any change in CLK cannot fix the error in C. Hence, given such condition, the erroneous node cannot be in the clock-gating circuitry. Thus, the erroneous node has to be a combinational node. ■

Algorithm 1 distinguishes clock-gating and combinational nodes in C. First, all the outputs in C are added to the combinational set (COM) (line 1- 4). Next, the circuit is traversed from the PO to PI to find all combinational nodes (line 5-16). Lemma 1 is used to decide if a node belongs to COM (line 8-9). After all the combinational nodes are found, CLK is simply all the other nodes in the circuit C except state elements.

Algorithm 1 is guaranteed to terminate because each node is visited only once (line 12) and the number of nodes is finite. Moreover, as Algorithm 1 visits all nodes except the clock-gating nodes, it successfully finds all combinational nodes.

Algorithm 2 shows our pre-processing step after CLK and COM are computed. First, the enhanced circuit C_{en} is constructed from from C (line 1) by copying all nodes and edges from C except for CLK nodes and the edges between them. The clock signals of all state elements in C_{en} are replaced by the primary clock input clk. Next we run simulation on both circuits using the trace from the counter-example and compare their output and states for all time-frames t less than or equal to k (line 5-8). If there is a mismatch between C and C_{en}, the algorithm returns false indicating that we

Algorithm 1: Clock Gating Nodes Computation

```
   input : C
   output: CLK, COM
 1 foreach y ∈ Y do
 2 |   Nodeslist.pushback(y);
 3 |   COM.pushback(y);
 4 end
 5 while Nodeslist is not empty do
 6 |   l ← Nodeslist.top();
 7 |   visited ← l;
 8 |   if l is not a clock signal then
 9 |   |   if l is not a state element then
   |   |   COM.pushback(l);
10 |   |   foreach i fanin of l do
11 |   |   |   if i ∉ visited then
12 |   |   |   |   Nodeslist.pushback(i);
13 |   |   |   end
14 |   |   end
15 |   end
16 end
17 CLK ← allnodes \ (COM ∪ S) ;
```

Algorithm 2: Clock Gating Debug Pre-processing

```
   input : C, CLK, COM, Trace
 1 C_en ← C \ CLK ;
 2 clk_en ← clk ;
 3 simulate(C_en, Trace) ;
 4 simulate(C, Trace) ;
 5 for t = 1 to k do
 6 |   if Y^t ≠ Y_en^t then return false ;
 7 |   if S^t ≠ S_en^t then return false ;
 8 end
 9 return true ;
```

cannot assume anything about the erroneous region. Otherwise, if all the states and outputs are equal, the algorithm returns true indicating that the erroneous gate *must* be a combinational node.

IV. Clock Gating Debugging

In this section, the debugging technique using Partial Max-SAT is presented. The problem is encoded as a Partial Max-SAT instance $Debug$ as shown in (1). Each solution of the Partial Max-SAT instance corresponds to a potential erroneous gate in circuit C. If the potential erroneous gate is in CLK, possible fixes are suggested based on their power saving.

A. Clock Gating Debugging using Partial Max-SAT

Given an unsatisfiable instance of $Debug$, the goal is to find sets of clauses that correspond to potential erroneous gates. Each set is known as a solution of the Partial Max-SAT instance $Debug$. Hence, the overall design debugging problem can be solved by finding all such solutions. In order to accomplish this, we incrementally call the Partial Max-SAT solver until all solutions are found. With regards to preventing duplicate solutions, all solutions found in the previous iterations are set as hard clauses in the subsequent iterations.

Algorithm 3 outlines our debugging process. Our algorithm takes in the unsatisfiable instance of $Debug$, the mapping from CNF clauses to gates and the error cardinality N. Each window is analyzed by traversing from the last window to the first one (line 2–13). When a window is analyzed, all other windows are set to hard clauses (line 5). In each window, several solutions are found. For each solution, it is mapped back to a gate in C and stored in the erroneous gates set, called $ERROR$ (line 9). Next the instance $Debug$ is updated with new hard clauses which are the previous solution (line 10). The algorithm stops when no more solution can be found or all the solutions of the error cardinality N have been discovered (line 7).

Algorithm 3: Clock Gating Fault Localization

```
   input : CNF Debug, mapping M, N, Trace
   output: set of erroneous gates ERROR
 1 current ← h ;
 2 repeat
 3 |   for t = 1 to h do
 4 |   |   if t = current then setSoftClause (window t);
 5 |   |   else setHardClause (window t);
 6 |   end
 7 |   repeat
 8 |   |   sol ← Max-SAT(Debug) ;
 9 |   |   ERROR ← ERROR ∪ M (sol) ;
10 |   |   Debug ← setHardClause (sol) ;
11 |   until (sol ≠ ∅) ∧ (size(M (sol)) <= N);
12 |   current −− ;
13 until current = 1;
```

B. Clock Gating Node Rectification

In this subsection we propose a technique that allows us to suggest rectifications for erroneous clock-gating nodes. First, all the available

fixes for the erroneous clock-gating node are listed. Each fix is verified to check if it corrects the erroneous behavior exhibited by the counter-example. Moreover, the fix is verified to ensure that it does not introduce any new design error. All qualified fixes are short-listed, and ranked based on their power saving level, estimated using conventional tools [16]. These fixes are called potential fixes.

Unlike the combinational circuitry, clock-gating circuitry is much smaller in terms of the number of gates involved and they are usually added manually by the designer. Due to the nature of clock-gating circuitry construction, it is evident that in most of the cases a dictionary-based rectification technique is adequate to rectify the error. Gates are grouped into sets based on the number of inputs and outputs. Our verification technique replaces potential erroneous gates with gates residing in the same set and verifies if it is a good fix. Thus, in our approach, we simply switch the candidate erroneous gate using a dictionary-based approach and verify the correctness of the clock-gating cone containing the erroneous node.

As a fix is derived from the dictionary of gates, it is not guaranteed to be the appropriate correction for the erroneous node. Therefore, a post-processing step is required to filter out spurious fixes. First, the potential fix has to correct the erroneous behavior of C. To ensure this, we apply the fix to C and simulate it using the counter-example. If the simulated outputs do not agree with the values from the golden model, the fix is discarded and the next available fix is examined. Otherwise, it is examined further to assert that it does not introduce new errors.

Of course, Sequential Equivalence Checking (SEC) can be used to ensure with confidence that the fix does not introduce new errors. However, a full-blown SEC step can be costly and there may be multiple corrections that must be checked. Since our fixes are only on clock-gating nodes, we can simply focus on the corresponding CGC. After the fix is applied, for any arbitrary trace (different from the counter-example trace), the fixed clock-gating circuitry can:

1) produce the same output as CLK of C.
2) turn on the clock signal while CLK turns it off.
3) turn off the clock signal while CLK turns it on.

Then the following can be deduced for the above three scenarios:

1) Here, as the clock signals are same, no error is introduced.
2) In this case, the clock signal in C is correctly turned off under STC. Based on Defn. 1 of STC, turning on the clock signal does not modify any states/outputs. Therefore, no error is introduced.
3) In this case, the clock signal in C is turned on, while the fixed clock-gating circuitry turns it off. This could alternate the register value in the fixed circuit compared to C and hence, potentially introduce new errors.

In order to conservatively ensure that no error is introduced, one can simply assert that the third scenario, as discussed above, does not occur. To prevent it, the fixed CLK must not turn off the clock signal when the clock-gating circuitry in C has it on.

Let us denote E_{fix} as the rectified CGC and E as the erroneous CGC. The symbols e and e_{fix} are used to denote the enable signals computed by E and E_{fix}, respectively. If E computes more than one enable signal, we extend our check for each enable signal and its correspondence in E_{fix}. However, to simplify our presentation without the loss of generality, let us assume that E only computes one enable signal e. Our goal is to assure that when $e = 1$ (ON), $e_{fix} \neq 0$, i.e $e_{fix} = 1$ (ON). This can be formally expressed as

$$e = 1 \Longrightarrow e_{fix} = 1 \tag{5}$$

Overall, if Property 5 is satisfied, the fix does not introduce new errors and hence, it is a potential fix.

Algorithm 4 shows our rectification process for clock-gating circuitry. After the debugging flow returns all possible erroneous gates, we only select the gates in CLK for further analysis (line 1). In order to fix the erroneous gate, we replace it with different available gates in the dictionary. Each possible fix is verified such that it corrects the erroneous behavior excited by the counter-example. If the fix passes the first check, we construct E_{fix} and check if the condition in (5) is satisfied as well. All qualified fixes are stored in $CORRECTION$ (line 6). The loop terminates when all the available fixes are tried (line 3). Next, all potential fixes are ranked based on their power saving level (line 12) and the fix corresponding to the greatest power savings is selected as the solution.

Algorithm 4: Clock Gating Node Rectification

```
   input : C, CLK, ERROR, dictionary
   output: gates, CORRECTION
1  foreach g ∈ ERROR do
2      if g ∈ CLK then
3          repeat
4              E_fix ← Rectify (dictionary, E) ;
5              if Verify (E, E_fix) then
6                  CORRECTION ← E_fix ;
7              end
8              trial++ ;
9          until trial = dictionary.size();
10     end
11 end
12 Rank (CORRECTION) ;
```

V. Experimental Results

This section presents the experimental results for the proposed clock-gating debug framework. The presented technique is implemented on state-of-art Max-SAT solver from [17]. All experiments are run on an Intel Core i5 3.1 GHz quad-core workstation with 8 GB of RAM; timeout is set at 7200 seconds. Several ISCAS-89 and ITC-99 benchmark circuits are used in our experiments. The clock-gated version of the circuits is implemented using Synopsys Design Compiler utilizing the STC condition. The gate level power consumption was measured using Power Compiler under normal loading. For each design, debugging instances are generated by injecting different design errors such as incorrect gate function, wrong gate input etc., both in the combinational logic as well as in the clock-gating circuitry. Each circuit is simulated using a test bench. The simulated output values are compared with the expected output values of the circuit. As soon as an inconsistency is detected between the expected and observed values of the circuit, the simulation is terminated and the trace is recorded. The circuit is then converted into CNF using the method in [13]. For all experiments, we compare our debugging methodology with the state-of-the-art presented in [12]. Also we compare the power consumption of the circuit using our rectification technique with the power consumption of the circuit when clock-gating implementations are removed.

Table I shows the results of Algorithm 2 where the error resides in the combinational logic (COM). The first four columns of Table I list the instance name, number of gates, number of time frames in the counter-example trace and the number of solutions found by Partial Max-SAT. Experiments are conducted with and without a pre-processing step. The next three columns specify the number of soft clauses, number of hard clauses and runtime without the pre-processing step. With pre-processing, the number of hard and soft clauses is specified in Column 10 Column 11 respectively. The next two columns report the pre-processing time and the total runtime. As stated earlier, the pre-processing time overhead is extremely small. The improvement in the number of hard clauses and the runtime is stated in Column 12 and 13. With extremely small runtime overhead for pre-processing step, our overall runtime is improved on-average by 12% with 15X increase in the number of hard clauses.

Table II shows the results when the error exists in the clock-gating circuitry (CLK). Column 2 provides the results from Max-SAT, in terms of number of possible gates that may be fixed to correct the error. Our methodology tries to rectify the error by modifying the CGC of the erroneous gate. Of all the rectification solutions, only a few of them satisfy the clock-gating condition and are listed next. The next three columns state the effect on power consumption. The power consumption of the circuits when all the clock-gating conditions are maximally utilized is stated in Column 4. The next column states the power consumption of the circuit with the erroneous clock-gating removed. Obviously the power demand of this circuit is more as compared to Column 4. Using the proposed rectification technique, some of the power savings can be reclaimed back and is stated in Column 6. The overall improvement, in terms of power is mentioned in the next column. The last column specifies the runtime of the entire design flow. A **TO** (**MO**) entry in the tables is used to refer

TABLE I
PRE PROCESSING RESULTS

Circuit Info				w/o Pre-processing			w Pre-processing				Improvement	
Instance Name	# gates	# total cycles	total # solns.	# hard clauses	# soft clauses	time (s)	# hard clauses	# soft clauses	Pre-pros. time (s)	time (s)	hard clause(X)	runtime (%)
s382	172	15	36	165	11176	3	1570	9771	1	2	9.5	33.3
s298	147	43	21	473	28274	13	9394	19353	1	10	19.8	23
s344	154	225	15	4950	140916	22	30576	115290	2	20	6.1	9
s349	133	39	34	858	22773	9	4344	19287	1	9	5	0
s9234	1125	51	240	3927	247493	467	7787	243633	3	378	1.9	19
s1423	989	77	>238	1848	298915	TO	67503	233260	2	TO	36.5	-
s838	412	17	64	629	27668	TO	10837	17460	1	2135	17.2	∞
s420	205	1977	>526	41517	1620932	TO	642461	1019988	14	TO	15.4	-
b03	101	27	53	270	17480	35	2306	15444	1	24	8.5	31.4
b04	695	27	43	567	72884	115	2603	70848	1	105	4.5	8.7
b05	1477	9	333	351	41285	164	1019	40617	1	161	2.9	1.8
b07	437	89	165	979	157555	1336	11101	147433	2	1282	11.3	4
b08	141	94	46	1410	62399	127	8538	55271	1	126	6	0.7
b09	171	39	105	156	31408	240	1630	29934	1	230	10.4	4.1
b12	1063	153	726	1989	711695	1665	144236	569448	5	1571	72.5	5.6

to a time-out(memory-out) condition for that experiment and partial results are presented. On average the rectified circuit can have 98% of the power savings revived. Thus, the benefit of the proposed method can be seen in the amount of power savings that would have remained unrealized otherwise.

TABLE II
CLOCK GATING DEBUG RESULTS

Instance Name	# total solns	# rect. solns.	w/o error (μW)	error removed (μW)	with rect. (μW)	improv (%)	time (s)
s382	62	3	8.8	13.6	9.0	97.7	10
s298	186	2	4.78	5.5	5.34	89.5	485
s344	30	3	5.8	6.02	5.8	100	17
s349	25	1	5.81	6.0	5.81	100	70
s9234	33	1	69.7	69.9	69.7	100	65
s1423	MO	-	38.1	40.7	-	-	-
s838	261	2	14.1	14.4	14.36	98.1	417
s420	48	3	7.45	8.33	7.46	99.8	88
b03	40	2	12.47	14.99	13.9	89.7	491
b04	27	2	57.03	62.90	57.03	100	55
b05	67	3	9.8	12.4	9.8	100	671
b07	57	3	15.8	18.3	15.8	100	69
b08	190	2	10.07	11.2	10.1	99.7	116
b09	180	3	15.10	15.3	15.16	99.6	66
b12	>124	-	52.07	52.4	-	-	TO

The scatter plot in Figure 4 shows the power saving of some of the circuits under different rectification solutions. The top and bottom end of each line represents the percentage of power savings with correct clock-gating implementation and with the removal of erroneous clock-gating respectively as compared to a non clock-gating design. The scatter points are the power savings achieved with rectification. Any solution within the range is a valid solution. In majority of the circuits power savings can be maximized using the proposed technique.

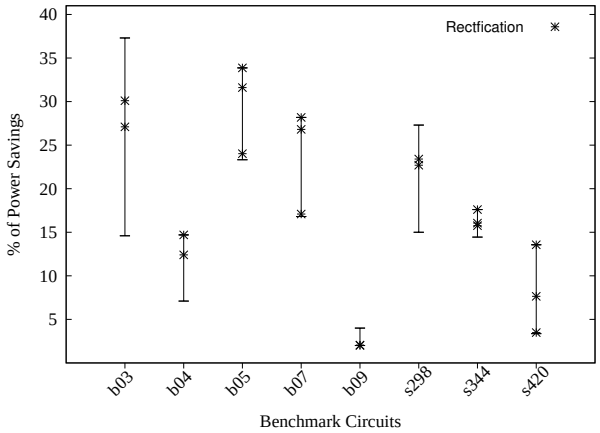


Fig. 4. Power Savings

VI. CONCLUSION

This work proposes a methodology to debug and rectify erroneous clock-gated designs. The first step is to distinguish the relative erroneous region. The debugging problem is then encoded as a Partial Max-SAT instance where each solution corresponds to a possible candidate of the error. After all the candidates are collected, qualified rectifications are suggested for ones in the clock-gating logic. Finally, an extensive set of experiments on standard benchmark circuits demonstrates the practicality of the presented framework. In the future, we would like to study the effects of other debugging formulations on enhancing clock-gating design debugging.

REFERENCES

[1] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, no. 1, pp. 3–56, 1996.

[2] L. B. P. Babighian and E. Macii, "A scalable algorithm for RTL insertion of gated clocks based on ODCs computation," *IEEE Trans. on CAD*, vol. 24, no. 1, 2005.

[3] G. K. Ranan Fraer and M. K. Mhameed, "A new paradigm for synthesis and propagation of clock gating conditions," in *Design Automation Conf.*, 2008, pp. 658–663.

[4] A. Hurst, "Fast synthesis of clock gates from existing logic," in *Int'l Workshop on Logic Synth.*, 2007.

[5] M. P. W. Qing and W. Xunwei, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. on Circuits and Systems I*, vol. 47, no. 3, 2000.

[6] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and R. Scarsi, "Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers," *ACM Trans. on Design Automation of Electronic Systems*, vol. 4, no. 4, 1999.

[7] L. Benini and G. D. Micheli, "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Trans. on CAD*, vol. 15, no. 6, pp. 630–643, 1996.

[8] E. Arbel, C. Eisner, and O. Rokhlenko, "Resurrecting infeasible clock-gating functions," in *Design Automation Conf.*, 2009, pp. 160–165.

[9] T.-K. Lam, X. Yang, W.-C. Tang, and Y.-L. Wu, "On applying erroneous clock gating conditions to further cut down power," in *ASP Design Automation Conf.*, 2011, pp. 509–514.

[10] C. M. Li and F. Manyà, *MaxSAT, Hard and Soft Constraints*. IOS Press, 2009, vol. 185, pp. 613–631.

[11] M. Evaluation, "Maxsat," *http://www.maxsat.udl.cat/*, 2012.

[12] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris, "Automated design debugging with maximum satisfiability," *IEEE Trans. on CAD*, vol. 29, pp. 1804–1817, November 2010.

[13] G. S. Tseitin, "On the complexity of derivations in the propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic*, 1968, pp. 115–125.

[14] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[15] B. Keng, S. Safarpour, and A. Veneris, "Bounded model debugging," *IEEE Trans. on CAD*, vol. 29, no. 11, pp. 1790–1803, Nov. 2010.

[16] Synopsys Inc., "Power Compiler", http://www.synopsys.com/.

[17] MSUnCore, *http://logos.ucd.ie/wiki/doku.php?id=msuncore*, 2012.