Sequential Equivalence Checking of Clock-Gated Circuits

Yu-Yun Dai[1], Kei-Yong Khoo[2],Robert K. Brayton[3]
[1,3]Department of EECS, University of California, Berkeley, U.S.A.
[2]Cadence Design Systems Inc., San Jose, U.S.A.
{[1]yunmeow, [3]brayton} @berkeley.edu;[2]khoo@cadence.com

ABSTRACT

Sequential clock-gating can lead to easier equivalence checking problems, compared to the general sequential equivalence checking (SEC) problem. Modern sequential clock-gating techniques introduce control structures to disable unnecessary clocking. This violates combinational equivalence but maintains sequential equivalence between the original and revised circuits. We propose the use of *characteristic graphs* (CGs) to extract essential LTL clock-gating properties, which when proved imply certain sequential redundancies. The extraction, proof and subsequent removal of the implied redundancies lead to an efficient SEC procedure for clock-gated circuits. Experiments show that the proposed SEC procedure substantially outperforms existing methods in terms of speed and scalability when applied to several difficult cases.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids-Formal Verification

General Terms

Model Checking, Verification

Keywords

Clock-Gating, Equivalence Checking

1. INTRODUCTION

In a modern VLSI design flow, equivalence checking is used between circuit designs in different stages [11]. As the use of sequential synthesis techniques increases, efficient sequential equivalence checking (SEC) methods become not only more necessary but also enable the use of sequential synthesis in the first place. The complexity of general SEC is P-SPACE complete, and hence more complex than combinational equivalence checking (CEC), which is only NP-complete.

This work is motivated by the fact that some sequential synthesis methods only modify a circuit by introducing control structures which are sequentially redundant [9]. Hence equivalence checking can be based on detecting these redundancies, eliminating them and then doing SEC between the resulting circuit and a *golden model*. We propose such a method, apply it to sequentially clock-gated circuits, and give some experiments comparing the new method against existing techniques.

The contributions of this paper are summarized below:

1. We formulate a graph representation (CG) of an AIG (R) circuit. The CG expresses the overlaying control structure of R. An algorithm is provided that constructs CG from R.

2. We show how to use CG to formulate LTL properties (P) about the flow control of R. These properties can express both forward and backward control properties over several cycles. P can be model checked on R easily because their supports rely only on signals highlighted in the CG.

3. We prove that each property in P, if proved, implies that an associated control signal directing the flow is sequentially stuck-at-1. This redundancy is used to simplify R into R'. Similarly, G is simplified to G'.

4. R' and G' are SEC checked. Since R' and G' are typically structurally more similar to each other, the subsequent model checking is very efficient.

5. This method was implemented and applied to a number of academic and industrial clock-gated circuits to SEC them against the original circuits. Experimental results show that this method is much more efficient than existing methods for performing SEC on clock-gated circuits.

DAC'15, June 07 - 11, 2015, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2744769.2744910.

The remainder of this paper is organized as follows: some basic notation is reviewed, and the most relevant previous works are discussed in Section 2. In Section 3, we describe a method for representing the essential features of a clock-gated circuit using a *characteristic graph*. Section 4 discusses the connection between the characteristic graph and sequential redundancy in clock-gated circuits. The overall flow of our SEC method for clock-gated circuits is given in Section 5, and Section 6 compares the performance of the proposed method with previous works using several sets of experiments. Section 7 summarizes the results and some future directions.

2. PRELIMINARIES

2.1 Sequential Circuits

A sequential circuit, as shown in Figure 1, consists of a combinational logic part(C_L), and sets of primary inputs(PIs), outputs(POs) and memory components (flip-flops, FFs). The combinational part represents a functional mapping from PIs and current states of FFs, to POs and the next state of each FF.

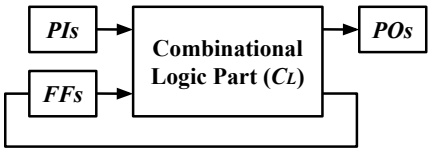

Figure 1: Standard representation of a sequential circuit.

Combinational and sequential synthesis techniques [10] are applied to sequential circuits in modern VLSI design flows to minimize chip area, reduce power consumption, optimize the clock period, etc. Combinational synthesis preserves the functional mapping to reduce the cost functions, while sequential synthesis provides more flexibility by possibly changing the next state functions and thereby producing further reductions in the costs.

2.2 Sequential Synthesis for Low-Power Circuits

Static power refers generally to the power required to maintain the state of a circuit, and dynamic power refers to power needed during switching activity. Here we only focus on the dynamic power consumption.

To reduce the power consumed for updating FFs, synthesis tools apply *forward* and *backward clock-gating* [9]. Forward clock-gating is used to disable the clock of a FF when all of its supporting dependencies (supports) are known to remain unchanged during the current clock period. Thus, a FF need not be updated if the FFs in its support remain the same as in the previous timeframe. Backward clock-gating turns off the clocks of FFs when the updates of the FFs can never be observed at the outputs.

Generally, forward and backward clock-gating techniques modify the clocks of FFs by using enabling signals. These are sequentially redundant in that they can be removed without modifying any observed behavior. These redundancies are used to minimize the frequency of updating some of the FFs, hence reducing dynamic power consumption. Therefore, the clock-gated circuit will keep the same sequential properties but with fewer updates of the memory components.

As shown in Figure 2, clock-gating a FF can be modeled with a feedback loop through a multiplexer, where the enable signal, E, controls the switch of O between I and its old value. Thus the modeling of clock-gated circuits only requires general FFs.

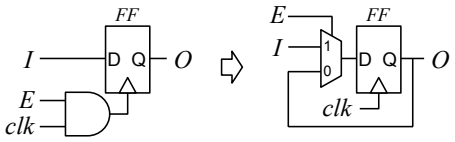

Figure 2: An equivalent model for clock-gating a FF.

2.3 Previous Work on Sequential Equivalence Checking

After synthesis, we need to ensure that the circuits before and after synthesis are sequentially equivalent [4]. SEC for general circuits is typically formulated as a model checking problem on the miter between two sequential circuits (where the PO pairs are XORed to form the outputs of the miter circuit). Thus, sequential model checking techniques, including induction [15], bounded model checking (BMC) [7] and property-directed reachability(PDR) [8], can be applied to check if the outputs of the two circuits are always identical. If an output of the miter can ever become *1*, sequential equivalence is violated, and model checking can provide an input sequence leading to the violation. Otherwise, the two circuits are proved sequentially equivalent.

However, due to the P-SPACE complexity of model checking, applying this to SEC problems may be too hard. Of course, CEC can be tried and if successful, the two circuits are also sequentially equivalent. However, most effective sequential synthesis techniques tend to change the next state functions.

Savoj at el. [13, 14] proposed a combinational approach to SEC for clock-gating synthesis. This approach aimed at circuits synthesized using satisfiability and observability don't cares (SDC and ODC) [12]. Although it worked well compared to existing methods, there are several weaknesses in this approach: (1) it cannot conclude non-equivalence, and therefore cannot supply a witness to help the designer understand the reason for this, (2) it requires unrolling and there is no suggested number of timeframes for unrolling, (3) it still has a scalability problem, especially when the combinational logic is extremely complicated.

To mediate those problems and improve the efficiency of SEC for clock-gated circuits, we propose the concept of the characteristic graph for clock-gated circuits.

3. CHARACTERISTIC GRAPH

The characteristic graph(CG) is an abstraction of its corresponding circuit. It is a high-level description, which represents only the essential properties needed for SEC.

3.1 Characteristic Graph

A characteristic graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is a directed graph, where a vertex stands for a group of PIs, POs, FFs or internal signals in the corresponding circuit. Each directed edge represents a signal dependency from one group to another. There are three types of edges: *selection-edge*, *on-edge* and *off-edge*. A selection-edge connects exactly one signal to the target group, to indicate the conditional switch of the group dependency. On-edges and off-edges connect the different sets of support groups to the target group if the selection signal is 1 or 0, respectively. The selection-edge of each PI is driven by a constant *True*, which means the value of each input signal is unconditionally updated. A group containing PIs cannot be driven by other groups. A group covering POs can have edges to other vertices, and such POs are treated as support FFs as well. A precise algorithm is given in Figure 5 for constructing the CG of a sequential circuit.

Figure 3 depicts a sequential circuit with its corresponding characteristic graph. A line with a white arrow represents a selection-edge, while solid and dotted lines with black arrows are on-edges and off-edges. The circles (vertices) stand for sets of signals in the original circuit, including PIs, E, A and B, the PO, Q, and FFs, F_1 and F_2. Note that there is no on-edge or off-edge into each PI, and their selection-edges are driven by constant *True*. Thus the value of each PI is not driven by any other signal, and it is updated at every clock tick. For the output, Q, the value of F_1 (selection-edge) determines if its value is driven by B (on-edge) or F_2 (off-edge). Finally, F_1 is only driven by E, so it has no off-edge and its selection is $True$.

To sum up, a characteristic graph abstracts the data dependency among 'essential' signals, while ignoring irrelevant combinational logic parts. Although this abstraction is especially motivated by the SEC problem for clock-gated circuits, it can be used in similar problems.

3.2 Construction of Characteristic Graph

Given a sequential gate-level circuit, we construct the characteristic graph as follows: (1) recognize selection signals (2) create vertices (3) build dependencies.

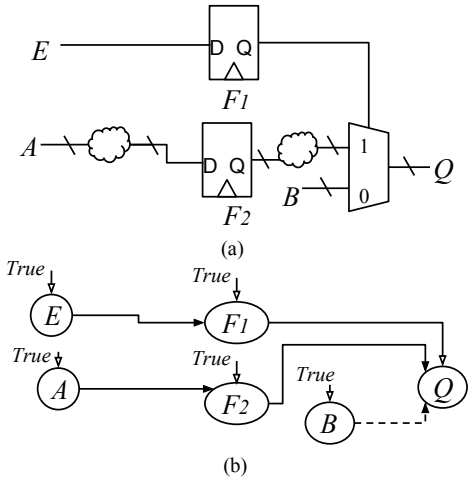

Figure 3: A sequential circuit (a) with its characteristic graph (b).

Recognize selection signals: When the input circuit is generated from a synthesis tool, in which 2-to-1 multiplexers (MUXes) are supported and explicitly expressed, the selector inputs of the MUXes are easily recognized and designated as selection signals. If the input circuit is an and-inverter graph(AIG), then we need to recognize instances of the MUX structure shown in Figure 4. The output signal O is controlled by S and conditionally switched between A and B. Therefore, the signal S here is recognized as the selection signal for the group consisting of output O. This structural matching can be performed over the AIG very quickly. However, it is possible that some essential MUX controls could be missed.

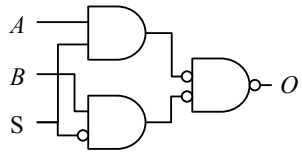

Figure 4: A typical 2-to-1 multiplexer represented as an AIG, where A and B are the two inputs, S is the selector, and O is the output: $O = SA + S'B$.

Create vertices: Initially, all POs and FFs are grouped by their common selection signals, while those signals without selection conditions are put into individual groups. For example, in Figure 3, E, A, B and F_1, F_2 are put in individual vertices. Each selection signal occupies an individual vertex. Each signal can be included in no more than one vertex.

Given a sequential circuit, **Cir**, with the sets of primary inputs **PI**, outputs **PO** and flip-flops **FF**, the algorithm for constructing the characteristic graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is shown in Figure 5. The function $recognize(\mathbf{Cir})$ at Line 2 is used to detect the MUX structures in **Cir** and collect the set of selection signals **S**. Based on **S**, **PI**, **PO** and **FF**, vertices are created and added into **V** through Line 3 to Line 11. The function $findTarget(\mathbf{Cir}, s_i)$ is used to collect the set of signals, which are controlled by s_i. From Line 12 to 26, the vertices are connected according to the data dependencies in **Cir**. For the function $connect(...)$ at Line 14, 19, 21 and 26, the first argument is the target vertex, the second is the support vertex, and the third is the edge type. The function $getVertex(\mathbf{V}, sup)$ returns the vertex which covers sup.

The function $backtrack(...)$ at Line 16, 17 and 24 goes back one timeframe from target t and returns the supports (PIs or FFs) on the boundaries. If the third argument, s, and fourth argument, *on* or *off*, are specified, this function will only backtrack the specified input side of each target MUX, and collect the corresponding supports.

Once the characteristic graph is constructed, it is used to detect sequential redundancy candidates.

4. SEQUENTIAL REDUNDENCY AND CLOCK-GATING

Algorithm: **Characteristic Graph Construction**
Input:
Cir: a gate-level sequential circuit with the sets of primary inputs **PI**, primary outputs **PO** and flip-flops **FF**
Output:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: characteristic graph for **Cir**, with the sets of vertices, **V** and edges, **E**.

01. $\mathbf{V} = \emptyset$ and $\mathbf{E} = \emptyset$
02. $\mathbf{S} = recognize(\mathbf{Cir})$ // S is the set of selection signals
03. **for** each PI_i in **PI**
04. $\mathbf{V} = \mathbf{V} \cup \{PI_i\}$;
05. **for** each s_i in **S**
06. $\mathbf{V} = \mathbf{V} \cup \{s_i\}$;
07. $\mathbf{T} = findTarget(\mathbf{Cir}, s_i)$
//**T** is the set of FFs and POs controlled by s_i
08. $\mathbf{V} = \mathbf{V} \cup \{\mathbf{T}\}$
09. **for** each ff_i in **PO** or **FF**
10. **if** ff_i is not covered by any v in **V**
11. $\mathbf{V} = \mathbf{V} \cup \{ff_i\}$
12. **for** each v in **V**
13. **if** v is controlled by selection signal s
14. $connect(v, getVertex(\mathbf{V}, s), selection)$
15. **for** each t covered by v
16. $\mathbf{Sup}_{on} = \mathbf{Sup}_{on} \cup backtrack(\mathbf{Cir}, t, s, on)$
17. $\mathbf{Sup}_{off} = \mathbf{Sup}_{off} \cup backtrack(\mathbf{Cir}, t, s, off)$
18. **for** each support sup_{on} in $\mathbf{Sup}_{on}$
19. $\mathbf{E} = \mathbf{E} \cup connect(v, getVertex(\mathbf{V}, sup_{on}), on)$
20. **for** each support sup_{off} in $\mathbf{Sup}_{off}$
21. $\mathbf{E} = \mathbf{E} \cup connect(v, getVertex(\mathbf{V}, sup_{off}), off)$
22. **else**
23. **for** each t covered by v
24. $\mathbf{Sup} = \mathbf{Sup} \cup backtrack(\mathbf{Cir}, t)$
25. **for** each support sup in **Sup**
26. $\mathbf{E} = \mathbf{E} \cup connect(v, getVertex(\mathbf{V}, sup), on)$

Figure 5: Algorithm for constructing a characteristic graphs.

Given a sequential circuit, *sequential redundancy* refers to a set of signals, which can be replaced by other signals or constant values (1 or 0), while preserving sequential equivalence to the given circuit. In other words, the fanouts of such signals can be moved to other existing signals or to constants without changing the observed behavior.

Clock gating synthesis can be either backward or forward. During this, additional control signals are placed into a sequential circuit to reduce the frequency of updating the FFs. These extra signals by definition, must be sequentially redundant in order to preserve sequential equivalence.

In this section, we will define and prove sufficient conditions for legal forward and backward clock-gating on sequential circuits. We use CGs to identify potential sequential redundancies.

4.1 Forward Clock-Gating

Forward clock-gating aims at turning off clocks for FFs when their support FFs remain at their previous states. Thus, the clock-gating is legal if all target FFs are guaranteed to update their states when their support FFs are updated. Otherwise, if none of the supports are updated, it is immaterial (don't care) if the target FFs are updated. Under proper initial states, the signal for disabling a clock is redundant and can be set to a constant.

Given a CG and a target signal C_f, which is the selection signal for a vertex V_f, the algorithm in Figure 6 is used to formulate a set of sufficient properties for C_f to be sequentially redundant to a constant *1*. The function $supportVertex(...)$ at Line 10 and 12 returns the set of supports, which are driving V_f by on-edges and off-edges. Sup_n stands for the supports of V_f obtained by backtracking exactly n timeframes. Notice that here we only collect supports from on-edges for the first timeframe, but for the further backtracking, those from off-edges are also included. Each sufficient property is added into the set **P** of accumulated properties at Line 16.

Theorem: If C_f is 1 at the first timeframe, a sufficient condition across 1 timeframe for C_f to be sequentially redundant to a constant 1 is

$$\mathbf{G}((C^1_{v1} \vee C^1_{v2} \vee ... \vee C^1_{vk}) \Rightarrow XC_f), \quad (1)$$

where $\{C^1_{vi}\}$ is the set of updating conditions for the support vertices of V_f, and XC_f is the value of C_f in the next clock cycle.

Notice that if V_f is driven by PIs, those inputs are taken as supports with control signals as $True$, which means C^1_{vi} equals *1*.

Proof: For timeframe 0, if the initial state results C_f to be 1, it is safe to replace the selector of V_f with constant 1. Then, the LTL property in Equation 1 guarantees the FFs covered by V_f (called FF_f) must be updated in the next timeframe whenever any support FFs of FF_f gets updated in the current timeframe. If the support includes a PI then at least one of the C_{v_i} is 1 so the formula states that FF_f is updated. In all other cases, all support FFs are unchanged from the previous clock cycle and thus the new value for FF_f is the same as the old, and we don't care if FF_f is updated or not. Thus C_f can be 1 or 0 in those cases. By choosing it to be 1 in those cases as well, C_f becomes constant 1, i.e. C_f is stuck-at-1 sequentially redundant. **Q.E.D.**

Note that Equation 1 is not a necessary condition because the support FFs may change to new states, but the combinational logic may compute a next state for FF_f which is the same as its current state.

The above condition states the legality of a single timeframe forward clock gating. For multi-timeframe clock-gating, supports are collected by backtracking more than one timeframe.

Theorem 1: when C_f is proved to be 1 at timeframe 0 to $n-1$, the following property is sufficient for C_f being sequentially stuck-at-1 redundant:

$$\mathbf{G}((C^n_{v1} \vee C^n_{v2} \vee ... \vee C^n_{vm}) \Rightarrow X^n C_f), \quad (2)$$

where $\{C^n_{vi}\}$ is the updating conditions for the support vertices, V^n_i, obtained by backtracking CG across n timeframes as detailed in Figure 6.

Proof: If $X^n C_f$ is 0, Equation 2 implies all C^n_{vi} are 0 in the previous n^{th} timeframe, which indeed guarantees all FFs covered by V^n_i remain in the same states. Therefore, C_f can be 1 or 0 (V_f can be updated or not) in these cases. By choosing it to be *1* in those cases, C_f becomes constant *1*. Based on the assumption that C_f is 1 at timeframe 0 to $n-1$, C_f is sequentially redundant stuck-at-1. **Q.E.D.**

The theorem generates a set of properties, any one of which is sufficient for C_f to be sequentially redundant under the corresponding condition of first n states. Note that the algorithm in Figure 6 stops generating new properties when the condition at Line 17 is violated. The *while* loop from Lines 5 to 16 is terminated when the n^{th} set of supports contains a primary input, which is always updated at each timeframe. Also, if the n^{th} set is the same as any previous support set, there is no need to backtrack **CG** anymore. This *while* loop is guaranteed to terminate because the number of vertexes in **CG** is limited.

Theorem 2: For a target signal C_f, if there exists some n, where the corresponding n-timeframe forward clock-gating condition is satisfied, the target signal C_f is sequentially stuck-at-1 redundant.

The assumption that C_f is 1 in the first n timeframes cannot be checked by the above LTL properties directly. Section 4.4 will explain the practice of the above theorems.

Algorithm: **Forward Properties**
Input:
C_f //the target signal which must be a control signal
$\mathbf{CG} = (\mathbf{V}, \mathbf{E})$ // characteristic graph containing C_f
Output:
P // set of sufficient properties for C_f being stuck-at-1

01. $V_f = controlledVertex(C_f)$
02. $\mathbf{P} = \emptyset$
03. $n = 0$
04. $Sup^0 = \{V_f\}$
05. **do**
06. $n = n + 1$
07. $Sup^n = \emptyset$
08. $C^n = False$
09. **for** each v_i in Sup^{n-1}
10. $Sup^n = Sup^n \cup supportVertex(v_i, \mathbf{CG}, on)$
11. **if** $n \neq 1 \wedge hasOffEdge(v_i)$
12. $Sup^n = Sup^n \cup supportVertex(v_i, \mathbf{CG}, off)$
13. **for** each v_k in Sup^n
14. $C^n_{vk} = controlSignal(v_k, \mathbf{CG})$
15. $C^n = C^n \vee C^n_{vk}$
16. $\mathbf{P} = \mathbf{P} \cup \{\mathbf{G}(C^n \Rightarrow X^n C_f)\}$
17. **while** $(Sup^n \cap \mathbf{PI} = \emptyset) \wedge (Sup^n \neq Sup^k, \forall k < n)$
// **PI** is the set of vertices containing primary inputs

Figure 6: For the target signal C_f, this algorithm defines a set of sufficient properties for legal forward clock-gating, where C_f is sequential stuck-at-1.

4.2 Backward Clock-Gating

Backward clock-gating is used to disable updating FFs when these updates are not observable.

Given a target control signal C_b, which determines if a set of FFs, V_b, is updated, the algorithm in Figure 7 formulates a set of sufficient properties for C_b to be sequential stuck-at-1 redundant.

The function $targetVertex(...)$ at Line 9 starts from v_i to find the target vertices (those driven by v_i) in CG, and returns *(vertex, type)* pairs, where *type* can be on-edge or off-edge. When the

input argument n is 1, it only collects those driven by on-edges (because we are checking this selection signal). If v_i is driving v_k through an off-edge, we need to negate the control signal C^n_{vk} in a property as Line 15 shows. This algorithm stops generating new properties when the n^{th} set of targets is repeating any previous set (Line 23), which means no new properties can be formulated.

Algorithm: **Backward Properties**
Input:
C_b //the candidate signal which must be a control signal
$\mathbf{CG} = (\mathbf{V}, \mathbf{E})$ // characteristic graph containing C_b
Output:
$\mathbf{P}$ // set of properties for proving C_b is stuck-at-1

01. $V_b = controlledVertex(C_b)$
02. $\mathbf{P} = \emptyset$
03. $n = 0$
04. $Target^0 = \{V_b\}$
05. **do**
06. $n = n + 1$
07. $FanoutMap^n = \emptyset$ // map of (vertex, type)
08. **for** each v_i in $Target^{n-1}$
09. $FanoutMap^n.add(targetVertex(v_i, \mathbf{CG}, n))$
10. $Target^n = \emptyset$
11. $C^n_v = True$ //control signals for FFs
12. $C^n_o = True$ //control signals for POs
13. **for** each pair $(v_k, type_k)$ in $FanoutMap^n$
14. $C^n_{vk} = controlSignal(v_k, \mathbf{CG})$
15. $C^n_{vk} = (type_k == off)? \sim (C^n_{vk}) : C^n_{vk}$
15. **if** v_k contains PO
16. $C^n_o = C^n_o \wedge (X^n C^n_{vk} \Rightarrow C_b)$
17. **else**
18. $C^n_v = C^n_v \wedge (X^n C^n_{vk} \Rightarrow C_b)$
19. $Target^n = Target^n \cup \{v_k\}$
20. $\mathbf{P} = \mathbf{P} \cup \{\mathbf{G}(\ C^n_v \bigwedge C^n_o\ )\}$
21. **if** $n > 1$
22. $\mathbf{P} = \mathbf{P} \cup \{\mathbf{G}(\ C^{(n-1)}_o\ )\}$
23. **while** $(Target^n \neq Target^j, \forall j < n)$

Figure 7: For the target signal C_b, this algorithm defines a set of sufficient properties for legal backward clock-gating, where C_b is sequential stuck-at-1.

Based on this algorithm, the sufficient property across one timeframe for C_b being sequentially redundant to constant *1* is

$$\mathbf{G}((XC^1_{o1} \Rightarrow C_b) \wedge ... \wedge (XC^1_{ok} \Rightarrow C_b) \bigwedge (XC^1_{v1} \Rightarrow C_b) \wedge ... \wedge (XC^1_{vr} \Rightarrow C_b)), \quad (3)$$

where $\{C^1_{vi}\}$ stands for the updating conditions of target vertices, and $\{C^1_{oi}\}$ is for target vertices containing POs, which are driven by V_b across one timeframe (because any PO is observable).

Proof: The LTL property in Equation 3 guarantees the FFs (FF_b) contained by V_b are updated when any of its targets V^1_i gets updated in the next timeframe. If C_b is 0, this property implies that none of V^1_i gets updated in the next timeframe. This blocks any new state of FF_b to be observed at any outputs. Thus C_b can be 0 or 1 in these cases. By choosing it to be *1*, C_b is sequentially stuck-at-1 redundant.

Similarly, we can write down a sufficient condition for backward clock-gating across n timeframes, which justifies target signal, C_b, as sequential stuck-at-1:

$$\mathbf{G}((X^n C^n_{o1} \Rightarrow C_b) \wedge ... \wedge (X^n C^n_{ok} \Rightarrow C_b) \bigwedge (X^n C^n_{v1} \Rightarrow C_b) \wedge ... \wedge (X^n C^n_{vr} \Rightarrow C_b)), \quad (4)$$

where C^n_{vi} stands for the updating condition of the i^{th} target vertex, V^n_i, which is driven by V_b across n timeframes, while C^n_{oi} is for the i^{th} target vertex containing PO. The two properties in Equation 3 and 4 are formulated at Line 20 in Figure 7.

Notice that the property across n timeframes is sufficient only when the following property holds for all m, $1 \leq m \leq n-1$:

$$\mathbf{G}((X^m C^m_{o1} \Rightarrow C_b) \wedge ... \wedge (X^m C^m_{o1} \Rightarrow C_b)), \quad (5)$$

where C^m_{oi} refers to the updating condition of the i^{th} vertex containing POs, V^m_i, which is driven by V_b across m timeframes. Once one of these properties is violated, it implies the clock-gating condition (C_b equals to 0) on V_b is observable at an output (at least one of $X^m C^m_{oi}$ equals to *1*), so it might be an invalid clock-gating case.

Theorem 3: If all the properties of Equation 5 hold for $1 \leq m \leq n-1$, then Equation 4 is sufficient for C_b to be sequentially stuck-at-1 redundant.

Proof: When $C_b = 0$, Equations 4 and 5 imply that none of the POs for the next n timeframes is observable. In addition when $C_b = 0$, the first part of Equation 4 implies that none of the final target FFs driven across n timeframes is observable as well. Since these represent all possible ways that the value in controlled vertex V_b can be observed, it would be correct if C_b were 1 as well. Hence C_b is sequentially stuck-at-1 redundant. **Q.E.D.**

Note that the stopping criterion for n is when the target set $Target^n$ is equal to one of the previous $Target^j$. This is because continuing to trace forward would just reproduce one of the conditions already seen in Equation 4 for some n.

Theorem 4: For a target signal C_b, if there exists some n, such that the n-timeframe backward clock-gating condition, all corresponding properties in Equation 4 and 5, are satisfied, the target signal C_b is sequentially stuck-at-1 redundant.

Note that there is no initial state requirement for backward clock-gating cases; the n-timeframe backward clock-gating condition alone is sufficient for redundancy.

4.3 Using the Characteristic Graph

A characteristic graph exposes the essential properties of the corresponding circuit, including signal dependency and control signals. It contains information required to formulate properties for the legality of a clock-gated circuit. The on-edges and off-edges of a CG connect the targets and supports across each timeframe, while each selection-edge indicates the updating condition for a group of signals. As discussed in Section 4.1 and 4.2, sufficient LTL properties for the legality of forward and backward clock-gating can be formulated. Each selection signal associated with a proved property has its corresponding signal in the original circuit which can be replaced by 1.

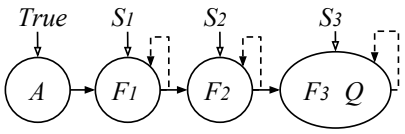

Figure 8: A CG with three stages of FFs. The vertex A is a PI, where the selection-edge is driven by $True$. These signals, S_1, S_2 and S_3 represent the updating conditions for FFs F_1, F_2 and F_3, respectively.

Consider the CG in Figure 8, where F_1, F_2 and F_3 are FFs, Q is a PO and S_1, S_2 and S_3 are corresponding control signals. The control signal S_1 only can be the backward clock-gating case, where a sufficient condition for S_1 being sequential stuck-at-1 is $\mathbf{G}(XS_2 \Rightarrow S_1)$. Because the PO is not driven by F_1 across one timeframe, another sufficient condition is $\mathbf{G}((XXS_3 \Rightarrow S_1) \wedge (XXS_2 \Rightarrow S_1))$.

For S_2, it can be a forward clock-gating case with the property $\mathbf{G}(S_1 \Rightarrow XS_2)$, or backward clock-gating with $\mathbf{G}(XS_3 \Rightarrow S_1)$. S_3 can only be a forward clock-gating case, where sufficient properties are either $\mathbf{G}(S_2 \Rightarrow XS_3)$ or $\mathbf{G}(S_1 \vee S_2 \Rightarrow XXS_3)$.

As implied by the LTL safety properties, only the fanin cones of those control signals need to be considered by model checking, and hence irrelevant combinational logic will be excluded automatically by state-of-the-art model checking methods. Thus, when model checking the generated properties on the original circuit, the problem size is effectively much less.

There are some limitations of finding sequential redundancy on CGs. Currently each vertex on a CG covers all signals controlled by the same selection signal, so some sequential redundant points may be missed. For example, if a set of FFs is clock-gated by forward condition with a certain control signal, while another set of FFs clock-gated by backward condition with the same signal, both the two cases are legal but cannot be proved by the current formulation. This issue can be resolved by separating all FF into different vertices, but that may result duplicate properties for proving.

4.4 Property Modeling and Proving

We describe the circuit-based modeling and proving for the LTL properties in Equation 1 to 5. In sequential circuits, the notation *next* (X) is represented by adding FFs in front of other signals. For example, the property in Equation 1 becomes

$$\mathbf{G}(Prev[C^1_{v1} \vee C^1_{v2} \vee ... \vee C^1_{vk}] \Rightarrow C_f). \quad (6)$$

Model checking of a safety property tries to find an execution path from the initial state which leads to a bad state, where the property gets violated. If no such path exists, the safety property holds. Therefore, the circuit structure shown in Figure 9 is added as an output for verifying the property in Equation 1, where an execution path (counter-example) leading to $O = 1$ would violate the safety property.

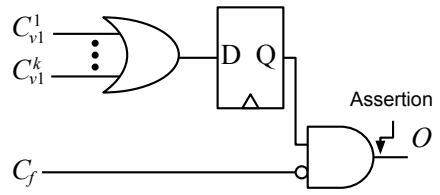

Figure 9: The circuit-based modeling for justifying the property in Equation 1. If there exists an execution path leading O to be 1, the safety property is violated.

As mentioned in Section 4.1, to check the value of C_f from timeframe 0 to $n-1$ in forward clock-gating properties, the initial states of extra FFs should be 1. For backward clock-gating, the X's are on the left hand side of Equations 4 and 5, so extra FFs need to be added to delay the C_b signal on the right to delay clock gating for n cycles. These FFs need to be initialized to 1 to avoid spurious counter-examples.

The potential redundant signals for clock-gating that are proved thus are applied to simplify the clock-gated circuit.

5. OVERALL FLOW

Given two sequential circuits, golden and clock-gated designs (**G** and **R**), with mapped PIs and POs, SEC verifies if the output sequences are identical when the same input sequences are applied. If **R** is known to be clock-gated from **G**, instead of applying general SEC methods, the difficulties of SEC can be reduced using the CG approach. Here we outline the overall flow of this method.

As our use-model, we assume that the golden model **G** may already be clock-gated in the RTL possibly manually by the designer. Therefore in comparing **G** and **R**, we reduce them both with the CG method and apply the proved redundancies to get **G'** and **R'**.

5.1 Comparisons between Characteristic Graphs

Comparisons between the characteristic graphs C_G and C_R of the golden **G** and clock-gated **R** circuits can be used to identify candidates for sequential redundancy. Since it is assumed that a correspondence between flops and POs in **G** and **R** is given, we can create a correspondence between nodes in C_G and C_R leading to pairs $\{(V_G, V_R)\}$. The following situations may exist for the pairs.

1. V_G and V_R are driven by exactly the same set of vertices (set of signals) with identical edges, which implies the two sets of signals might be combinationally equivalent.
2. V_G and V_R are driven by the same set of vertices with on-edges (off-edges), but the vertices driving off-edges (on-edges) are different. In this case, the selection signals are regarded as candidates for sequential redundancy.
3. V_G is only driven by a set of vertices with on-edges, while the supports of V_R are conditionally switched between this support set and V_R itself. In this case, the selection signal of V_R is a candidate for sequential redundancy.
4. V_G and V_R are driven by different sets of vertices with on-edge, but there is no selection signal. If the paired signals are combinationally equivalent, then the two FFs are already proved sequentially equivalent. Otherwise, the difference must be resolved by model-checking, but possibly on a greatly simplified problem where other proved redundancies are used.

5.2 Algorithm Flow

Figure 10 outlines an algorithm to perform SEC between two circuits, **G** and **R**, and report if they are sequentially equivalent(**EQ**) or not(**NON-EQ**).

The function $CEC(...)$ at Line 1 performs the general combinational equivalence checking between corresponding signals in each pair and returns a set of unproved pairs. The function *charGraph*(...) returns the corresponding characteristic graph for the specified circuit.

The loop between Line 7 and 15 verifies each candidate and revises the corresponding CG by the proved redundancy one by one . At Line 7, the function *comparison*(...) analyzes these unresolved pairs as described in Section 5.1 and propose a candidate of sequential redundancy. The function $defineProperty(...)$ applies the ideas in Section 4 for each target signal, and generates the set of corresponding LTL properties (**P**) to be proved. Notice that not only **R** but also **G** can contain sequential redundancy, so from Line 8 to 11, properties are defined for C_G or C_R, respectively.

```
Algorithm: Proposed SEC
Input:
  G and R // two circuits with mapped PIs, POs and FFs.
Output:
  EQ or NON-EQ

01. nonEQ = CEC(G, R)
02. if nonEQ = ∅
03.     return EQ;
04. C_G = charGraph(G)
05. C_R = charGraph(R)
06. proved = ∅
07. while (candidate = comparison( nonEQ, C_G, C_R))
08.     if candidate ∈ G
09.         P = defineProperty(candidate, C_G)
10.     else
11.         P = defineProperty(candidate, C_R)
12.     proof = multiProve(P)
13.     if isLegal(proof)
14.         revise(candidate, C_G, C_R)
15.         proved = proved ∪ candidate
16. (G', R') = simplify(proved, G, R)
17. return SEC(G', R')
```

Figure 10: The proposed SEC algorithm for G and the clock-gated circuit R.

The function $multiProve(...)$ at line 12 verifies a circuit with multiple outputs. Given a set of properties **P**, $multiProve(\mathbf{P})$ verifies all properties simultaneously, and then returns the result set **proof**, which lists both proved and disproved properties. At Line 13, $isLegal(\mathbf{proof})$ analyzes the result and determines if **candidate** is sequentially redundant using **Theorems 1** to **4**. Due to the possible dependencies among candidates, the corresponding CG should be revised by proved redundancy before the next run. All proved candidates are used to simplify **G** and **R** into **G'** and **R'** at once (Line 16).

Finally, we perform $SEC(\mathbf{G'}, \mathbf{R'})$ to check if **G'** and **R'** are sequentially equivalent. If **G'** and **R'** are proved **NON-EQ**, $SEC(\mathbf{G'}, \mathbf{R'})$ can return a counter-example, which is also valid for **G** and **R**. Therefore, the proposed algorithm can provide counter-examples to users for debugging.

It might be that previous synthesis done on **R** has destroyed some of the MUX structure in the circuit, and then some MUXes might not be recognized during the CG construction. Thus, our algorithm might fail to identify all sequentially redundant points. In this case, we identify as much redundancy as possible and still use SEC for the final checking.

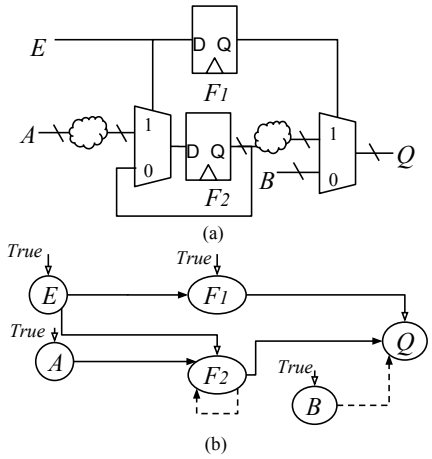

Figure 11: The revised circuit for Figure 3 with its characteristic graph, where the inputs, outputs and FFs are perfectly mapped to those in the golden design.

To show how this algorithm works, consider the circuits in Figure 3 and 11 as **G** and **R**. At line 1, only the pair for F_2 fails CEC and is added into **nonEQ**. From Line 4 to 7, E, the selection signal of F_2, is a candidate of sequential stuck-at-1 redundancy. Since it can only be backward clock-gating, the corresponding property, $\mathbf{G}(XF_1 \Rightarrow E)$, is created and proved. Then **R** is simplified into **R'** by replacing the selector of the MUX of F_2 by constant 1, while **G'** is the same as **G**. Therefore **G'** and **R'** become identical to Figure 3, and can be proved equivalent by SEC easily. Finally, the algorithm returns that **G** and **R** are

Table 1: Comparisons with *super_prove* and *Absec* on three OpenCores [3] cases and two synthetic cases.

Circuit	Clock-Gating Techniques	AND #	FF #	*super_prove* (s)	*Absec* (s)	CG method(s) *Simplify*	CG method(s) *SEC*
aes.Round	Backward	125k	645	208.2	5.31	0.67	2.95
Md5Core	Forward	95k	40k	80.33	N/A	0.92	7.92
CLA_fixed	Backward	3k	97	T.O.	1169.78	0.66	1.97
Synthetic_1	Backward	4k	73	T.O.	166.28	0.56	0.23
Synthetic_2	Both	877	74	177.06	N/A	0.65	0.43

sequentially equivalent.

6. EXPERIMENTAL RESULTS

We compare the CG method against two state-of-the-art methods, (1) model checker *super_prove* [6], which won the single-output track in the Hardware Model Checking Competition 2014 (HWMCC'14) [2], and (2) *Absec*, a command implemented in ABC [5] which performs the algorithm in [14].

The CG method, $SEC(\mathbf{G}, \mathbf{R})$ is implemented in ABC. The $multiProve(...)$ function used is *multi_prove*, which won the multi-output track in HWMCC'13 [1] (not held in HWMCC'14). We also apply *super_prove* to the final SEC between **G'** and **R'**.

All experiments were performed on a 16-core 2.60GHz Intel(R) Xeon(R) CPU with a 1500 second time limit. The example circuits were clock-gated at the RTL, and then synthesized into AIGs to create **R**. Each input for *super_prove* is a multi-output miter between a golden design (**G**) and its clock-gated circuit (**R**). The inputs for *Absec* and the CG method are **G** and **R** given before mitering.

6.1 Performance for General Clock-Gated Cases

First we examine the applicability and efficiency of the three methods applied for general clock-gated cases. Table 1 lists five cases with their circuit sizes, along with how they are clock-gated. The first three circuits were downloaded from OpenCores [3](**G**) and modified (**R**), while the last two cases were created (both **G** and **R**) for this comparison. The CG results are separated into two stages: *Simplify* includes Line 1 to 16 in Figure 10, while *SEC* refers to the final SEC at Line 17.

Because *Absec* is implemented in ABC only for backward clock-gating, it is not applicable to the forward clock-gating cases (indicated with N/A in Table 1).

As can be seen in Table 1, the proposed CG method significantly outperforms the other two methods. Although the general model checking method *super_prove* can prove some of the forward and backward clock-gating cases, it requires much more run time than the specialized methods. We see that *Absec* can reduce the sequential complexity and prove equivalence for backward cases. The final two columns of Table 1 show that the CG method is very efficient in both the redundancy finding phase and the final SEC proof after the redundancies have been removed.

6.2 Comparisons of Scalability

The second set of experiments compares the scalability of above methods by applying them the same design (*qmult* taken from OpenCores [3]), but with varying bit-widths. As the widths increase, the combinational part gets more complex. All of these cases were modified using only backward clock-gating in order to allow *Absec* to be applied.

Table 2: Comparisons with *super_prove* and *Absec* on *qmult*, a design from OpenCores [3], with varying bit-widths.

Circuit	AND #	FF #	*super_prove* (s)	*Absec* (s)	CG method(s) *Simplify*	CG method(s) *SEC*
qmult_8	487	25	0.35	0.90	0.58	0.33
qmult_9	632	28	4.09	3.61	0.64	0.34
qmult_10	791	31	23.14	10.50	0.65	0.34
qmult_11	964	34	61.28	98.28	0.65	0.35
qmult_12	1151	37	113.92	153.96	0.65	0.35
qmult_13	1352	40	T.O	229.61	0.66	0.36
qmult_14	1567	43	T.O.	1308.66	0.65	0.37
qmult_15	1796	46	T.O.	425.79	0.54	0.36
qmult_16	2039	49	T.O.	620.23	0.65	0.37

Table 2 shows that as the complexity of the circuit increases, the runtimes of *super_prove* and *Absec* increase sharply. In contrast, the CG method is not affected by the increased complexity because the complicated combinational logic is effectively excluded.

7. CONCLUSIONS

This paper presents a novel SEC method for clock-gated circuits. The proposed method is based on constructing a characteristic graph (CG) to model essential signals of a circuit. It uses CG to formulate sufficient properties for sequential redundancies. These properties are proved and used to simplify the circuit after which SEC becomes easy. The experimental results show that the CG method is scalable and effective, and substantially outperforms existing techniques.

Future research will include experiments on benchmarks which have been more additionally modified before or after clock-gating to see how missing some essential signals in the CG will effect performance. Also we need to experiment with errors inserted in the clock-gating to see how we can detect these and feedback useful counter-examples.

In addition, we want to explore how the CG concept might be applicable to other verification and synthesis problems. For example, based on the ideas in Section 4, a CG might be useful to synthesize clock-gating on a circuit. It might be used or extended to extract control paths from circuits with many arithmetic operators, and then used to mediate equivalence checking for these.

8. ACKNOWLEDGE

This work is supported in part by SRC contract 2265.001. We also thank industrial sponsors of BVSRC: Altera, Atrenta, Cadence, Calypto, IBM, Intel, Jasper, Mentor Graphics, Microsemi, Real Intent, Synopsys, Tabula, and Verific for their continued support.

9. REFERENCES

[1] *Hardware Model Checking Competition 2013*. http://fmv.jku.at/hwmcc13/.

[2] *Hardware Model Checking Competition 2014*. http://fmv.jku.at/hwmcc14cav/.

[3] *OpenCores*. http://opencores.org/, 1999-2014.

[4] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen. Scalable sequential equivalence checking across arbitrary design transformations. In *International Conference on Computer Design*, pages 259–266. IEEE, 2006.

[5] R. Brayton and A. Mishchenko. Abc: An academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40. Springer, 2010.

[6] R. K. Brayton, N. Een, and A. Mishchenko. Using speculation for sequential equivalence checking. In *International Workshop on Logic and Synthesis*, pages 139–145, 2012.

[7] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[8] N. Een, A. Mishchenko, and R. K. Brayton. Efficient implementation of property directed reachability. In *Formal Methods in Computer-Aided Design*, pages 125–134, 2011.

[9] A. P. Hurst. Automatic synthesis of clock gating logic with controlled netlist perturbation. In *Design Automation Conference*, pages 654–657, New York, NY, USA, 2008. ACM.

[10] V. Kravets, A. Mishchenko, S. Krishnasamy, N. Modi, R. Brayton, R. Puri, K. Gulati, and S. Khatri. *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[11] A. Kuehlmann and C. A. J. van Eijk. Combinational and sequential equivalence checking. In *Logic Synthesis and Verification*, pages 343–372. Kluwer Academic Publishers, Norwell, MA, 2001.

[12] A. Saldanha, A. R. Wang, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-level logic simplification using don't cares and filters. In *Design Automation Conference*, pages 277–282, New York, NY, USA, 1989. ACM.

[13] H. Savoj, D. Berthelot, A. Mishchenko, and R. Brayton. Combinational techniques for sequential equivalence checking. In *Formal Methods in Computer-Aided Design*, pages 145–150. FMCAD Inc, 2010.

[14] H. Savoj, A. Mishchenko, and R. Brayton. Sequential equivalence checking for clock-gated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(2):305–317, 2014.

[15] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *Formal Methods in Computer-Aided Design*, pages 108–125, London, UK, UK, 2000. Springer-Verlag.