Constructing Stability-based Clock Gating with Hierarchical Clustering

Bao Le[1], Djordje Maksimovic[1], Dipanjan Sengupta[3], Erhan Ergin[3], Ryan Berryhill[1], Andreas Veneris[1,2]

***Abstract*—In modern designs, a complex clock distribution network is employed to distribute the clock signal(s) to all the sequential elements. As the functionality of these sequential elements depends heavily on usage scenarios, it is vital that the clock network is optimized for these scenarios. This paper introduces a clock network power optimization methodology based on design usage patterns and stability based clock gating. Specifically, whenever a register retains its value from the previous cycle, a clock gating implementation shuts off its clock and disables data loading to enable power reduction. We first introduce the notion of a stability pattern and its correlation with clock gating efficiency. Next, we introduce a methodology to identify efficient clock gating implementations. In this framework, a clustering algorithm leveraging stability patterns iteratively computes more effective gating implementations. Each implementation is evaluated further on area overhead and critical path delay. If it satisfies all criteria, it is implemented in the design; otherwise, it is sent back to the clustering algorithm to compute new clock gating implementations. Empirical results show 22.6% reduction in clock network power and 16.0% reduction in total power consumption. This confirms the practicality and robustness of the proposed methodology.**

***Index Terms*—Stability Based Clock Gating, Clock Gating Efficiency, Agglomerative Hierarchical Clustering**

I. Introduction

With the growing demand for mobile and wearable devices, chip designers are under increasing pressure to design ultra-low power chips. As a clock distribution network constitutes more than 30% of the total consumed power, it is imperative that its power consumption is minimized [1]. Clock gating is a well known technique to reduce clock network power [2]. To maximize power reduction, clock gating opportunities must be aggressively utilized. This often requires designer knowledge of the system application. With increasing design complexity and shorter time-to-market, finding effective clock gating conditions continues to be a challenging task.

Clock gating is a circuit transformation where the clock of a register is shut off when there is no useful activity at the register [2]. This enables power savings as unnecessary switching is prevented. Clock gating is implemented at different hierarchies of design. Coarse grained clock gating identifies idle conditions of major functional units at the architecture level [3]. Conversely, fine grained clock gating operates at gate level by identifying idle conditions that are not visible at architecture level [4]. Currently, automated tools identify limited clock gating opportunities at the RTL-level mostly based on circuit topology. As such, significant manual effort is necessary to search for clock gating opportunities at the gate level. Therefore, automation remains crucial to achieve additional power savings from the clock network.

The focus of the work presented here is on fine grained clock gating. It does not address coarse grained clock gating nor power gating, wherein entire functional blocks are powered down when idle. In more detail, fine grained clock gating turns off the clock of a register under the following conditions: i) when the register output is not observed at the primary outputs, a condition known as Observability Don't Care (ODC), or ii) when the register output retains its value for two or more consecutive clock cycles, also known as a stability condition (STC) [5]. ODC is a well studied combinational synthesis technique and can be automatically extracted from steering logic such as multiplexers, tri-state buffers and enable signals [6]. However, STC relates to the sequential behaviour of the design and its detection remains challenging and requires significant design effort [7].

Various methodologies have been proposed to automate clock gating identification and synthesis. The work in [8]–[10] studies physical synthesis for gated clock designs. However, this is out of the scope of this paper as we focus on clock gating opportunity identification, a task typically performed at design or logic synthesis stages. In [11], the authors compute clock gating conditions using zero-skew trees. The work in [5], [12] utilizes symbolic models for clock gating construction. In addition,the authors of [1], [13] construct stability based clock gating from existing internal signals in the design thereby avoiding major area overhead. The authors in [14], [15] discuss grouping registers for clock gating construction. This work deploys the K-Mean algorithm [16] which requires the number of groups to be specified. This is not trivial especially for large and complex designs.

Overall, the goal of prior work is to find stability clock gating implementations that work with multiple registers and require the least amount of additional circuitry to be implemented. As available opportunities may remain unidentified, reaching optimum power savings requires further investigation.

In this paper we propose a clock gating framework utilizing the hierarchical clustering algorithm. We first introduce the concept of a stability pattern, a crucial element in clock gating implementation. For each register, a one dimensional binary array is constructed to represent its stability pattern. Next, we tailor hierarchical clustering to leverage this information to optimize clock gating implementations. Each final implementation has two properties: it shuts off the clock of multiple registers and it has great power saving capability.

Besides power savings, it is crucial to ensure that a clock gating implementation does not impact other design metrics such as area and critical path delay. To address these issues, we further present a clock gating evaluation technique that examines each gating implementation on area overhead and critical path delay. We synthesize each clock gating implementation within the original design and examine its impact. If the criteria are satisfied, the implementation is inserted into the design. Implementations that do not satisfy design constraints are utilized to construct new efficient gating implementations.

When we profile the methodology, experiments demonstrate that the clock gating circuitry generated saves 16.0% of total power consumption of the chip, on the average, and adds only 7.76% of additional area, an attractive design trade-off. In terms of clock power, we reduce the figure by 22.6%. Evidently, these results confirm the effectiveness of the method.

The paper is organized as follows. Section II presents

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

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

[3]Advanced Micro Devices, 33 Commerce Valley Dr. East, Markham, ON L3T 7N6 ({dipanjan.sengupta, erhan.ergin}@amd.com)

background information on clock gating. Section III describes the concept of stability pattern and the cluster algorithm and how they are utilized in clock gating construction. Section IV presents the global evaluation framework. Section V contains experiments and Section VI states the conclusion.

II. Preliminaries

The following notation is used throughout the paper. Given a sequential circuit C, the symbols $X = \{x_1, x_2, \cdots, x_{|X|}\}$, $Y = \{y_1, y_2, \cdots, y_{|Y|}\}$, and $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$ represent the sets of primary inputs, primary outputs and registers in C, respectively. Small letters such as $a, b, c, \cdots$ represent internal signals in C. For any signal z, z^k denotes the value of z at clock-cycle (*i.e.*, time-frame) k. Finally, clk denotes the root (*i.e*, global) clock. We assume the design has one clock domain and all registers to be positive-edge triggered.

A. Clock Gating Synthesis

As mentioned, fine-grained clock-gating is inserted at the gate-level netlist by identifying idle conditions of individual registers. This paper focuses on stability based clock gating and the following example illustrates STC in a design.

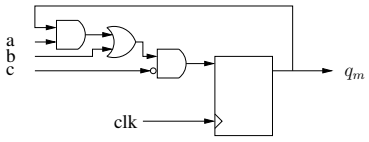


Fig. 1. Stability Condition Example

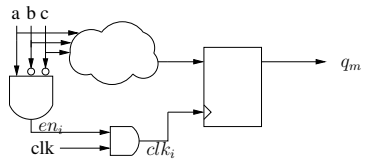


Fig. 2. Clock Gating Example

Example 1 *Figure 1 shows a STC example. Register q_m is stable when the condition $(a = 1 \wedge b = 0 \wedge c = 0)$ is satisfied. In other words, when $a = 1$, $b = 0$ and $c = 0$, q_m retains its value from the previous clock-cycle. The condition $(a = 1 \wedge b = 0 \wedge c = 0)$ can be employed as a clock gating condition to limit unnecessary switching at the register. Figure 2 displays a functionally equivalent clock gating implementation. In this implementation, when $en_i = 0$, the gated clock clk_i stays 0 and thus disables the register data loading. When $en_i = 1$, the clock and the register behave as usual.*

Conceptually, a clock gating implementation utilizes internal signals to compute a clock controlling (*i.e.*, clock enable) signal. In Example 1, en_i is the clock enable signal and clk_i is the gated clock. For the rest of this paper, the symbol $CG = \{cg_1, cg_2, \cdots, c_{|CG|}\}$ denotes the set of clock gating implementations. For an implementation cg_i, the corresponding clock enable, gated clock are denoted as en_i and clk_i, respectively. As the focus of this work is clock gating under STC, for the remaining of the paper, we use the term "clock gating" to denote "stability based clock gating".

B. Clock Gating Efficiency

The objective of a clock gating implementation cg_i is to prevent a portion of the clock network from unnecessary toggling. Consequently, cg_i is evaluated on its ability to block such activity.

Definition 1 *Given a clock gating implementation cg_i, the efficiency of cg_i, denoted as $E(cg_i)$, is defined as the number of times the clock clk has been cut down by cg_i over a period of time.*

The authors in [17] compute clock gating efficiency using the toggle rate of the gated clock:

$$E(cg_i) = 1 - \frac{toggle\_rate(clk_i)}{toggle\_rate(clk)} \quad (1)$$

Quantity $E(cg_i)$ can also be computed using signal probability propagation. As the clock is shut off whenever $en_i = 0$, a clock gating efficiency is the probability the enable signal being equal to zero:

$$E(cg_i) = Pr(en_i = 0) \quad (2)$$

Unfortunately, computing $Pr(en_i = 0)$ through probability propagation is a complex task for large designs [18]. As a result, designers often rely on simulation to obtain $Pr(en_i = 0)$ and $E(cg_i)$. Next, we show a novel approach to obtain $E(cg_i)$ without constructing cg_i.

III. Clock Gating Computation

In this section, we first describe the concept of a stability pattern and how it relates to clock gating efficiency. Next, we present an enhanced hierarchical clustering algorithm that leverages stability patterns to generate clock gating implementations. In more detail, we modify hierarchical clustering so that it not only combines similar implementations but also ensures that a combination would result in an efficient implementation. We also present an evaluation phase where implementations are tested on area and critical path delay before being employed in the design.

A. Stability Pattern

Definition 2 *Given a synchronous design C and a t-cycle stimulus, a stability pattern of a signal, 'a', is a single dimension t-entry binary array, denoted as $SP(a)$. The k-th entry of $SP(a)$ is equal to 1 (*i.e. $SP(a)[k] = 1$*) if and only if a is stable at cycle k, i.e $a^k = a^{k-1}$.*

From Definition 2, it is obvious that $SP(a)[k] = 0$ if and only if signal a is not stable at cycle k. By convention, $SP(a)[0] = 0$, $\forall a$.

We now extend stability pattern definition to a set of registers.

Definition 3 *Given a synchronous design C and a t-cycle stimulus, a stability pattern of a set of registers, Q_i, is a single dimension t-entry binary array, denoted as $SP(Q_i)$. The k-th entry of $SP(Q_i)$ is equal to 1 (*i.e. $SP(Q_i)[k] = 1$*) if and only if all registers in Q_i are stable at cycle k.*

Evidently, $SP(Q_i)[k] = 0$ if there is at least one register in Q_i that is unstable at cycle k.

Example 2 *Given a synchronous design C, a signal $a \in C$ and a stimulus of 5 cycles, if values of 'a' over 5 cycles are*

[0,1,1,0,0,1], its corresponding stability pattern $SP(a)$ would be [0,0,1,0,1,0].

A clock gating implementation cg_i is said to be the *optimal gating implementation* for a set of registers Q_i if and only if cg_i cuts down the clock for all cycles that registers in Q_i are stable. As $SP(Q_i)$ is a binary array and $SP(Q_i)[k] = 1$ for all clock-cycles k that registers in Q_i is stable, $Pr(en_i = 0) = \sum_{k=0}^{t} SP(Q_i)[k]$. Hence, we can compute $E(cg_i)$ as:

$$E(cg_i) = \frac{\sum_{k=0}^{t} SP(Q_i)[k]}{t} \tag{3}$$

In both Eq. 1 and 2, computing the efficiency requires the toggle rate of clk_i or the probability of en_i. These data require cg_i to be implemented in the design. On the other hand, Eq. 3 computes the efficiency directly from register data and does not require cg_i to be implemented. This is beneficial especially early in the design stage. During this stage, a large set of clock gating implementations is available to the designer and not all of them can be implemented due to the design constraints. The task at hand is to select efficient implementations. Evaluating a gating implementation using Eq. 1 or 2 requires inserting it in the design, which may be impractical to do for every available implementation. Eq. 3 therefore provides a cost-effective way to evaluate a large set of clock gating implementations as the efficiency of each implementation can be quickly calculated from simulation data.

Besides computing gating efficiency, stability patterns are crucial in combining clock gating implementations. Due to static power consumption, it is recommended to employ a single gating implementation, with one enable signal for multiple register clocks. In the next section, we describe how stability patterns are used in combining clock gating implementations.

B. Clock Gating Clustering

This section presents a novel clustering algorithm that combines clock gating implementations based on the stability pattern similarity of their registers. We also describe how negative combinations are avoided. The net result is clock gating implementations which are effective in power reduction while still maintaining engineering efforts already invested in the design.

Recall that a stability pattern is a single dimension binary array of length t. This array can be viewed as a data point in a t-dimensional Euclidean space. As such, we can define stability distance between two signals as:

Definition 4 *A stability distance between two signals a and b is defined as:*

$$d(a,b) = \sqrt{\sum_{k=0}^{t} (SP(a)[k] - SP(b)[k])^2} \tag{4}$$

Similarly, stability distance between two sets of registers Q_i and $Q - j$ is:

$$d(Q_i,Q_j) = \sqrt{\sum_{k=0}^{t} (SP(Q_i)[k] - SP(Q_j)[k])^2} \tag{5}$$

where $SP(Q_i)$ and $SP(Q_j)$ are stability patterns of two sets of registers Q_i and Q_j. In essence, $d(Q_i,Q_j)$ is the number of cycles that registers in Q_i have different stability behaviors compared to registers in Q_j.

We extend the stability distance definition further for clock gating implementations. For two clock gating implementations cg_i, cg_j, with corresponding sets of registers Q_i and Q_j, the stability distance between cg_i and cg_j, denoted as $d(cg_i,cg_j)$, is the stability distance of Q_i and Q_j, $d(Q_i,Q_j)$. For the rest of this paper, the term "distance" implies "stability distance".

Intuitively, if two clock gating implementations cg_i and cg_j have a small distance, they can be combined into a single clock gating implementation. In more detail, $(d(cg_i,cg_j))^2$ is the number of cycles where Q_i and Q_j have different stability behaviors. If $d(cg_i,cg_j)$ is small, Q_i and Q_j are more probable to be stable together and thus, can be clock-gated using a combined enable signal. Therefore, combining clock gating implementations can be achieved by grouping corresponding data points based on proximity. This can be achieved by employing a clustering algorithm. More specifically, we employ an Agglomerative hierarchical clustering based algorithm [16] due to the following reasons:

1) Hierarchical clustering does not require the number of final groups to be specified. This is critical in this work as the number of clock gating implementations differs from design to design. Moreover, as complexity varies from implementation to implementation, the final number cannot be determined without knowing which implementations will be utilized in the design. Therefore, it is realistic and beneficial to have the number of implementations determined by clock gating insertion tools.
2) Agglomerative hierarchical clustering is a *bottom up* approach. We start with a clock gating implementation for each register. At each iteration, pairs of implementations close in distance are merged. This iterative approach can quickly identify inefficient combinations, reverse the merging and direct the algorithm toward more effective combinations. This will be presented in more detail next.

As stated, distance plays an important role in combining clock gating implementations, However, distance is not the only criterion to consider. Example 3 shows that both distance and gating efficiency are essential in constructing gating implementations. While distance dictates the number of register clocks controlled by the enable signal, efficiency ensures the power saving capability.

Example 3 *Consider three registers q_m, q_n and q_o and their stability patterns [1, 0, 1, 0, 0, 1], [1, 1, 1, 1, 1, 0], [0, 0, 1, 0, 0, 0], respectively. We can compute that $d(q_m,q_n) = 4$ and $E(cg_i) = 33.3\%$ where cg_i is the clock gating circuitry of q_m and q_n. For q_m and q_o, $d(q_m,q_o) = 2$ and $E(cg_j) = 16.6\%$ where cg_j is the clock gating circuitry of q_m and q_o. In this example, in terms of clock gating efficiency, it is actually more beneficial to clock gate q_m with q_n ($E(cg_i) > E(cg_j)$) together even though q_m is closer to q_o than q_n ($d(q_m,q_o) < d(q_m,q_n)$).*

Motivated by the above observations, we tailor Agglomerative hierarchical clustering. The proposed clustering algorithm not only groups neighbouring data points but also ensures that gating efficiency after combining is higher than a predefined threshold. This threshold prevents the algorithm from suggesting inefficient implementations. Furthermore, the algorithm stops when further merging results in a clock gating implementation efficiency below the threshold.

Algorithm 1 presents the pseudocode of the enhanced hierarchical clustering. Initially, each register has its own clock gating implementation (Line 1). At each iteration, the algorithm first computes the distances between all pairs of implementations (Line 6-7). Pairs of implementations are then

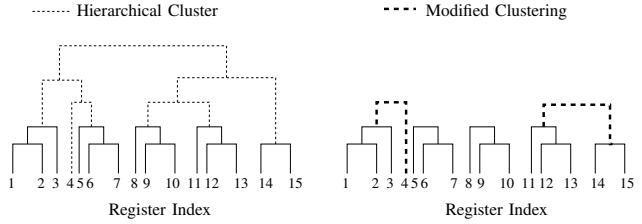
Algorithm 1: Clock gating clustering

input : $Q, sim, minE$
output: CG

```
1  foreach q_m ∈ Q do CG.insert(q_m) ;
2  change ← true ;
3  while change do
4      change ← false ;
5      foreach {cg_i, cg_j} ⊂ CG do
6          d(cg_i, cg_j) ← EuclideanDistance(sim) ;
7          D.push(d(cg_i, cg_j)) ;
8      end
9      D.sort() ;
10     foreach d(Q_i, Q_j) ∈ D do
11         if E(cg_ij) > minE then
12             CG ← Merge(cg_i, cg_j) ;
13             D ← RemovePair(D, Q_i, Q_j) ;
14             change ← true;
15         end
16     end
17 end
18 return clusters ;
```

sorted in ascending order of distance (Line 9). The algorithm next merges pairs of implementations due to their proximity. A merge is allowed if the new implementation efficiency is greater than a predefined threshold. Since the closest pairs are considered first, this is similar to a minimum-spanning tree algorithm with the addition of backtracking. If two implementations have been combined, they are removed from the consideration set (Line 13). Algorithm 1 simply stops when further grouping negatively impacts the clock gating efficiency.

Algorithm 1 is guaranteed to terminate. Initially, $|CG| = |Q|$. At each iteration, the number of implementations is reduced ($E > minE$) or the algorithm terminates. If the number of clock gating implementations keeps reducing the algorithm would eventually stop when there is only one left.



(a) Hierarchical Clustering (b) Clock Gating Clustering

Fig. 3. Hierarchical and clock gating clustering

Example 4 *Figure 3 compares the execution (*i.e.*, dendrograms) of a traditional hierarchical clustering and that of the proposed algorithm on a sample set of registers. Without the efficiency threshold, the classical hierarchical algorithm groups implementations until there is only one. When the number of implementations is too small, each implementation efficiency is insignificant to provide any power savings. The enhanced algorithm stops before clock gating implementations become inefficient. The efficiency threshold can even alter the grouping. Dashed lines indicate groupings differently between two algorithms. More specifically, Register q_4 even though closer to registers q_5, q_6, produces a higher clock gating efficiency when combined with registers q_1, q_2, q_3. For the same reason, registers q_{11}, q_{12}, q_{13} are combined with registers q_{14}, q_{15} in the enhanced algorithm and combined with registers q_8, q_9, q_{10} in hierarchical clustering.*

IV. Clock Gating Evaluation

The algorithm presented in the previous section produces a set of clock gating implementations that shut off multiple register clocks with high efficiency. Before being employed, these implementations must also be verified to respect design constraints, namely area and critical path delay. It is essential to note that the number of implementations derived by Algorithm 1 is not large. This is due to many original implementations being combined or removed due to low inefficiency. Thus, we can analyze each implementation before inserting into the design.

To investigate the impact of an implementation cg_i on area and critical path, we first synthesize it with the original design. After synthesis, we compare the clock gated design and the original design in terms of area and critical path delay. If the insertion of the clock gating circuitry violates the area or critical path constraints, it will not be implemented. In practice, these timing and area constraints can be specified by the designer or provided by the design specifications.

Algorithm 2: Clock gating evaluation

input : C, CG, $maxArea$, $maxDelay$, $sim, minE$
output: $FinalCandidates$

```
1  CG_t ← CG ;
2  CG.clear() ;
3  while Area.isAvailable() do
4      foreach cg_i ∈ CG_t do
5          C_cg ← Synthesize(cg_i, C) ;
6          Area, Delay ← Synthesize(C, cg_i) ;
7          if (Area > maxArea) ∨ (Delay > maxDelay)
           then
8              Q ← Q ∪ Q_i ;
9          else
10             CG.insert(cg_i) ;
11         end
12     end
13     minE ← minE + α ;
14     CG_t ← cluster(Q, sim, minE)
15 end
16 return CG ;
```

Algorithm 2 shows the pseudocode of the evaluation algorithm. Given a set of implementations, the evaluation flow returns the ones that will be inserted into the design. All implementations that are being evaluated are stored in CG_t. For each $cg_i \in CG_t$, the algorithm synthesizes cg_i with C (Line 5). The area overhead and critical path delay are computed next (Line 6). Implementations satisfying design constraints are stored into CG, the final set of implementations. All implementations that violate design constraints, stored in CG_t, are sent back to Algorithm 1 to find new implementations (Line 14). The algorithm terminates when there is no more area available for clock gating implementations.

As stated Algorithm 2 attempts to find new implementations from ones that violate area or critical path delay constraints. This is accomplished by sending them to Algorithm 1 with higher efficiency threshold $minE$. Recall that Algorithm 1 aggressively combines implementations as long as the new implementation efficiency greater than $minE$. These aggressive combinations create implementations that work with a large number of register clocks; however, they can also result in

TABLE I
CLUSTERING CLOCK GATING RESULTS

Design Info		Original Designs			Industrial Clock Gated Designs						Clustering Clock Gated Designs					
Design Name	# Regs	Area (μm^2)	Clock Power (μW)	Total Power (μW)	Area (μm^2)	Area Ovhd (%)	Clock Power (μW)	Total Power (μW)	Clk Pw Impr (%)	Total Pw Impr (%)	Area (μm^2)	Area Ovhd (%)	Clock Power (μW)	Total Power (μW)	Clk Pw Impr (%)	Total Pw Impr (%)
divider	424	9137	238.9	313.7	9894	8.28	213	256	10.8	18.4	10329	13.05	130	213	45.6	32.1
div64bits	5512	142056	410	508	142387	0.23	304	411	25.9	19.1	142865	0.57	287	378	30.0	25.6
deframer	2509	37060	650	870	37060	0	650	870	0	0	37315	0.69	634	810	2.5	6.9
fdct	5717	193067	4440	5320	193067	0	4440	5320	0	0	211029	9.30	3120	4680	29.7	12.0
mrisc	371	3700	62.3	75.93	4783	29.27	25.3	32.1	59.4	57.7	5346	44.49	15.7	40	74.8	47.3
b14	215	8365	176.8	218.6	8365	0	176.8	218.6	0	0	8404	0.47	155.3	195.7	12.2	10.5
b15	416	12427	343.8	411.4	12427	0	343.8	411.4	0	0	12561	1.08	338.8	409.1	1.5	0.6
b17	1314	37527	1078	1250	37527	0	1078	1250	0	0	38870	3.58	992.6	1172	7.9	6.2
b19	6030	207518	4956	5915	207518	0	4956	5915	0	0	209218	0.82	4921	5896	0.7	0.3
b21	430	16306	353.1	445.5	16306	0	353.1	445.5	0	0	16889	3.58	279.2	365	20.9	18.1
AVG.						3.78			9.61	9.52		7.76			**22.6**	**16.0**

complex implementations that violate other design constraints. In order to prevent these combinations, Algorithm 2 iteratively increases the efficiency threshold for Algorithm 1 (Line 13). This effectively prevents Algorithm 1 from making too many combinations. This is a technique we call *recursive clustering*. The motivation behind recursive clustering is to divide large implementations into smaller ones that achieve the same power savings while still satisfying design constraints.

V. EXPERIMENTAL RESULTS

This section presents the experimental results. All experiments are run using a single core of a i5-2500K 3.3 GHz workstation with 8GB of RAM and a timeout of 7200 seconds. Five OpenCore and five ITC-99 benchmark circuits are used to profile the method. All circuits are synthesized with a TSMC 65nm technology library. Place and route for each circuit is then performed. Finally, the power consumption is estimated. All of these tasks are completed using industrial tools.

Each design is simulated using a test bench. This test bench depicts a typical design run and thus is utilized to construct stability patterns for the registers. For Algorithm 1, the effectiveness threshold, $minE$ is set at 40%. $maxArea$ is set at a limit of 5% of the original design area. This means that each clock gating circuitry cannot add more than 5% of the original design area. Parameter $maxDelay$ is set to the same value as the maximum delay of the original design. This ensures that clock gating does not increase the critical path delay. For each circuit, we implement clock gating using an industrial tool and the proposed framework. Area, maximum frequency and power consumption are compared between the original version and two clock gated versions. All data is collected after physical synthesis is performed.

Table I shows the empirical results. The first column gives the design name and the next column contains the total number of registers. The next three columns have the area, the clock network power consumption and the total power consumption of the design, respectively. The next six columns give information on the design when clock gating is implemented by Power Compiler. In more detail, columns six, seven, eight and nine present the area, the area overhead, the clock network power consumption and the total power consumption. Columns ten and eleven display the improvement by employing Power Compiler. The last six columns show data when using the proposed clock gating framework. Columns 12, 13, 14 and 15 give the area, the area overhead, the clock network power consumption and the whole design power consumption. The final two columns show the overall improvement achieved by the presented methodology.

In these experiments, Power Compiler is restricted to only implement stability based clock gating. Even though Power Compiler can also implement ODC based clock gating, those implementations are out of the scope of this paper and hence are not included. It is essential to note that the presented framework does not alter or hinder any ODC based power reduction capability. Hence, any ODC implementations would provide comparable power reductions to the original or STC-based clock gated version of the design. Industrial tools construct stability clock gating by identifying specific design structures exhibiting stability conditions. While this technique works well on processor designs, it cannot be utilized for a wide range of design types. More specifically, when those structures are not available and the stability conditions are complex, Power Compiler cannot identify stability clock gating opportunities. Our technique does not exploit circuit structures and thus can be applied for any design. Even for processor designs, it also identifies more clock gating opportunities. Overall, the proposed methodology saves 22.6% of the clock network power and 16% of the total power compared with the original circuit while Power Compiler only saves around 9.5%. It should be noted, that previous work [1], [13], [19] do not report the actual design area but only the complexity of the AND-INVERTER graph representation of the design. Evidently, these numbers cannot be used and compared to our results that are generated by commercial vendor tools.

Table II displays the run-time of the proposed methodology on different designs. The first column displays the design name. Column two shows the number of clock gated registers and the next column contains the number of clock gating candidates implemented in the design. Column four gives the run-time of the clustering algorithm and Column five shows the run-time of the evaluation framework presented in Section IV. The last column gives the total run-time. It is essential to note that although `fdct` and `b19` time out after 7200 seconds, the framework is still able to construct some gating implementations for those designs. These implementations, as shown in Table I, already provide significant power savings. These designs timeout because many of the merged clock gating implementations have efficiency below the threshold and have to be reverted.

Figure 4 displays the size of CG_t in Algorithm 2 during evaluation runs for the `divider` circuit with and without recursive clustering. Algorithm 1 initially produces 34 clock gating implementations. Recursive clustering sends constraint violating implementations back to Algorithm 1. Without recursive clustering, we simply remove any implementations that violate design constraints. As discussed earlier, recursive clustering prevents large implementations that violate constraints by creating smaller implementations. Thus, $|CG_t|$ can be increased. In this example, the number of implementations at

TABLE II
CLOCK GATING CLUSTERING RUN-TIME

Design Name	# CG Regs	# CGs	Clustering Time (s)	Evaluation Time (s)	Total Time (s)
divider	225	13	0.045	3.14	3.2
div64bits	339	16	93.6	2029.17	2122.8
deframer	46	3	7.43	314.82	322.3
fdct	1936	228	TO	TO	TO
mrisc	1313	20	1.16	9.5	10.7
b14	25	4	0.45	12.22	12.7
b15	6	1	0.91	25.63	26.5
b17	132	33	3.58	107.39	111.0
b19	60	11	TO	TO	TO
b21	101	12	0.9	54.67	55.6

one point increases to 40. More importantly, by dividing large implementations into smaller ones, recursive clustering is able to identify seven `additional` employable implementations. This shows the practicality of recursive clustering.

Finally, Figure 5 displays the area and total power consumption for `divider` under clock gating with different $minE$ thresholds. Recall that $minE$ is the minimum efficiency of an implementation that Algorithm 1 can return. We can see that when $minE$ is set too small ($\leq 20\%$), the power saving is not optimal. This is due to many inefficient clock gatings being implemented. Parameter $minE$ should also not be set too high as efficient implementations may be rejected. For `divider`, when $minE = 90\%$, no clock gating is implemented. For `divider`, $40\% \leq minE \leq 60\%$ gives the most power saving. This range implies that all effective implementations have efficiency greater or equal to 60%. As far as area is concerned, the clock gating implementations do not pose a significant overhead. The maximal overhead for `divider` is 15.8% when $minE = 20$ and only 13.05% for $40\% \leq minE \leq 60\%$. Of all benchmarks, only `mrisc_core` poses a significant area overhead. This is due to `mrisc_core` being a small design and thus clock gating implementations may consume significant area w.r.t to the original design size.

VI. CONCLUSION

This work proposes an automated clock gating construction flow that employs hierarchical clustering to identify efficient gating implementations. In more detail, we introduce the concept of stability patterns and tailor hierarchical clustering algorithm to leverage stability pattern for clock gating optimization. We also propose an evaluation technique which prevents ineffective gating implementations from being implemented. The net result is a set of gating that have good power saving capability and are cost-effective to implement. An extensive set of experiments demonstrates its practicality and effectiveness. In the future, we plan to utilize place and route information and more advanced grouping algorithms to improve power saving capability. An additional target of future work is to avoid the exhaustive search through all pairs of clock gating implementations in Algorithm 1 by employing a more sophisticated data structure.

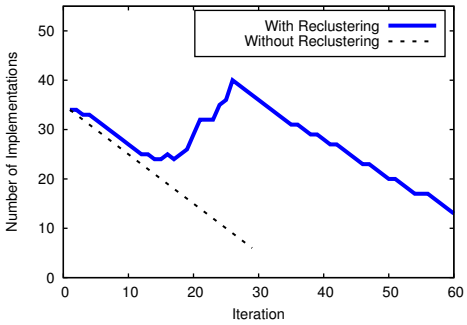


Fig. 4. # of implementations vs. iterations for the `divider` circuit

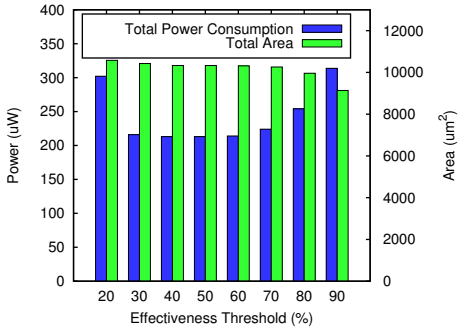


Fig. 5. Power consumption and area of `divider` with clock gating

REFERENCES

[1] A. Hurst, "Automatic synthesis of clock gating logic with controlled netlist perturbation," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, June 2008, pp. 654–657.

[2] W. Qing, M. Pedram, 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.

[3] B. Le, D. Sengupta, and A. G. Veneris, "Reviving erroneous stability-based clock-gating using partial max-sat," in *ASP-DAC*, 2013, pp. 717–722.

[4] N. Banerjee, K. Roy, H. Mahmoodi, and S. Bhunia, "Low power synthesis of dynamic logic circuits using fine-grained clock gating," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 2006, pp. 1–6.

[5] P. Babighian, L. Benini, 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.

[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] R. Fraer, G. Kamhi, and M. Mhameed, "A new paradigm for synthesis and propagation of clock gating conditions," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, 2008, pp. 658–663.

[8] W. Shen, Y. Cai, X. Hong, and J. Hu, "Activity and register placement aware gated clock network design," in *Proceedings of the 2008 International Symposium on Physical Design*, ser. ISPD '08. New York, NY, USA: ACM, 2008, pp. 182–189.

[9] ——, "An effective gated clock tree design based on activity and register aware placement," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 12, pp. 1639–1648, DECEMBER 2010.

[10] ——, "Activity-aware registers placement for low power gated clock tree construction," in *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, March 2007, pp. 383–388.

[11] D. Garrett, M. Stan, and A. Dean, "Challenges in clockgating for a low power asic methodology," in *Low Power Electronics and Design, 1999. Proceedings. 1999 International Symposium on*, Aug 1999, pp. 176–181.

[12] M. Donno, A. Ivaldi, L. Benini, and E. Macii, "Clock-tree power optimization based on rtl clock-gating," in *Design Automation Conference, 2003. Proceedings*, June 2003, pp. 622–627.

[13] T.-H. Lin and C.-Y. Huang, "Using sat-based craig interpolation to enlarge clock gating functions," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, June 2011, pp. 621–626.

[14] S. Wimer and I. Koren, "Design flow for flip-flop grouping in data-driven clock gating," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 771–778, April 2014.

[15] ——, "The optimal fan-out of clock network for power minimization by adaptive gating," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 10, pp. 1772–1780, Oct 2012.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[17] J. Srinivas, M. Rao, S. Jairam, H. Udayakumar, and J. Rao, "Clock gating effectiveness metrics: Applications to power optimization," in *Proceedings of the 2009 10th International Symposium on Quality of Electronic Design*, ser. ISQED '09, 2009, pp. 482–487.

[18] F. N. Najm, "Power estimation techniques for integrated circuits," in *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '95, 1995, pp. 492–499.

[19] R. Wiener, G. Kamhi, and M. Y. Vardi, "Intelligate: Scalable dynamic invariant learning for power reduction." in *Power And Timing Modeling, Optimization and Simulation (PATMOS), 18th Int. Workshop on*, L. Svensson and J. Monteiro, Eds., vol. 5349. Springer Berlin Heidelberg, 2009, pp. 52–61.