Scalable Sequence-Constrained Retention Register Minimization in Power Gating Design

Ting-Wei Chiang[†], Kai-Hui Chang[‡], Yen-Ting Liu[‡] and Jie-Hong R. Jiang[†§]

[†]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
[‡]Avery Design Systems, Inc., Tewksbury, MA, USA
[§]Computer Science Laboratory, Tomsk State University, Tomsk, 634050, Russia
{r02943088@ntu.edu.tw, changkh@avery-design.com, pheonix@avery-design.com, jhjiang@ntu.edu.tw}

ABSTRACT

Retention registers are utilized in power gating design to hold design state during power down and to allow safe and fast system reactivation. Since a retention register consumes more power and costs more area than a non-retention register, it is desirable to minimize the use of retention registers. However, relaxing retention requirement to a minimal subset of registers can be computationally challenging. In this paper, we adopt satisfiability solving for scalable selection of registers whose retention is unnecessary and exploit input sequence constraints to increase the number of non-retention registers. Empirical results on industrial benchmarks show that our proposed methods are efficient and effective in identifying non-retention registers.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*

General Terms

Algorithms, Design

Keywords

Partial retention, Synthesis, Formal methods

1. INTRODUCTION

Static power consumption is one of the major concerns for modern mobile devices powered by batteries. Power gating (PG) is a well-known technique to reduce static power consumption by shutting off current to unused blocks within a design. However, the state of power-down blocks will become non-deterministic after the block is reawakened, producing unpredictable circuit behavior. This problem can be addressed by using external memory to preserve data during power-down mode (sleep mode) at the expense of long latency to scan out and restore the data. For timing critical applications, an alternative solution is to use retention registers that are specially designed to have low-leakage and can retain their values in sleep mode. By replacing all registers in the block with retention registers, full block state can be quickly restored when the block is reactivated — this method is called full retention. However, full retention may require more power than necessary because some registers may not need retention if their values will always be updated before they are first read after power up. Therefore, it is desirable to retain only a subset of all registers. This design methodology is called partial retention.

In [15], Tian *et al.* reported that partial retention (19% non-retention) reduced leakage power consumption by 10% even though the new version of the design has 16% more registers. In their flow, designers need to manually inspect the design to select registers that do not need retention, which is time-consuming and error-prone. To address this problem, we present a flow to select non-retention registers automatically.

There have been several prior efforts on retention synthesis. In [7], Darbari *et al*. presented a case study using symbolic simulation for assisting the designers to implement selective retention correctly. The main finding is that the programmer visible state or the architectural state needs to be retained, while other micro-architectural enhancements such as pipeline registers, TLBs and caches can be implemented using normal registers without retention. This work provides a useful guideline for selective state retention implementation, but no automated methods are proposed. In [3], Chen *et al*. proposed the concept and usage of multi-bit retention register, which can reduce the leakage power and area considerably. However, additional circuitry and clock cycles are needed for mode transition and are not compatible with most design methods. Greenberg *et al.* recently proposed methods [9, 10] for automatically classifying all the flip-flops (FFs) of a given design into two categories: essential FF (e-FF) and redundant FF (r-FF). [10] performs the analysis according to read/write criteria on the synthesized gate-level netlist with Binary Decision Diagram (BDD) as the data structure, while [9] represented the criteria as a set of formal properties using propositional formulas and then uses common formal verification tools to drive the identification of the e-FFs. Even though the proposed approaches are formally sound, they are difficult to be applied by non-formal experts due to the required effort to provide proper input constraints using BDDs. In addition, They need massive parallel processing to achieve reasonable runtime, leaving scalability a major concern. Compared with [9, 10], our proposed method verifies power intention and retention

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

http://dx.doi.org/10.1145/2744769.2744905.

scheme in the early phase of design flow by symbolic simulation [4, 5], which allows us to handle not only gate-level but also RTL designs without logic synthesis. Performing the analysis at the RTL instead of gate-level makes it easier for the designer to inspect the results. Another advantage of symbolic simulation is that the SAT instance is constructed with the state variables retained, making it easy to implement our algorithm and can reduce the effort to analyze additional non-essential FFs introduced during logic synthesis.

In contrast to existing literature, our proposed algorithm [2] uses real test sequences that most designers are familiar with to perform the analysis, making it considerably easier to use compared with formal methods. Our algorithm utilizes high-level symbolic simulation on the RTL design and relies on the Conjunctive Normal Form (CNF) as the formula representation. It can automatically identify the set of non-retention registers during typical VLSI design flow efficiently and effectively.

The rest of the paper is organized as follows. Section 2 provides some preliminaries. Section 3 presents the problem statement and the detailed algorithm of retention synthesis. Finally, experimental results and conclusions are shown in Section 4 and Section 5, respectively.

2. PRELIMINARIES

As conventional notation, symbols $\neg$, $\wedge$, $\vee$ and $\Rightarrow$ stand for logical connectives negation, conjunction, disjunction, and implication, respectively. The cardinality of a set S is denoted as $|S|$. Let $V = \{v_1, \ldots, v_k\}$ be a finite set of Boolean variables. A literal l is either a Boolean variable v_i or its negation $\neg v_i$. A clause C is a disjunction of literals. A conjunction of clauses is in the so-called CNF. In the sequel, a clause set $C = \{C_1, \ldots, C_k\}$ shall mean to be the CNF formula $C_1 \wedge \ldots \wedge C_k$. A total (partial) assignment σ over V gives every (some specific) variable v_i a Boolean value either 0 or 1. Assignment corresponds to cube; total assignment correspond to minterms. A CNF formula is satisfiable if there exists a satisfying assignment such that the formula evaluates to 1; otherwise it is unsatisfiable.

2.1 SAT Solving and Incremental SAT

We assume the reader's familiarity with Boolean satisfiability (SAT) solving [13, 8] and the conversion from circuit to CNF formula [16]. A more detailed exposition can be found, e.g., in [11]. Assumptions-based incremental SAT searches for an assignment that satisfies the current set of clauses under the unit assumptions $assumps = \bigwedge_{i=0}^{n} a_i$. The assignment that satisfies all the clauses and the unit literals a_i will be returned if it exists. If the problem is UNSAT under the given assumptions, the subset of those assumptions used in the proof of UNSAT will be returned in the form of a final conflict clause.

2.2 Retention Register and Partial Retention

A Retention Register (RR) in general has a control, called RET in this paper, that enables the register to retain a state. When RET is low (sample mode), the register works like a normal register (without retention). When RET is set high (hold mode), the register retains the state that it kept just before RET was held high. Retention registers are used in designs that require fast resumption of operation after wakeup to preserve states in power-down blocks. However, a retention register occupies larger area than a normal register and consumes power in sleep mode. The former also increases wire length that can further degrade design performance.

2.3 State Transition Systems

A *state transition system* consists of a state transition relation $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$ and a set $I(\vec{s})$ of initial states, where $\vec{s}$, $\vec{s}'$, $\vec{x}$, and $\vec{y}$ are referred to as the current-state variables, next-state variables, input variables, and output variables, respectively. In the sequel, state sets are represented with characteristic functions. We shall not distinguish between a characteristic function and the set that it represents. For a deterministic system as we shall assume, the transition relation $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$ can be alternatively treated as the transition function $T : [\![\vec{x}]\!] \times [\![\vec{s}]\!] \rightarrow [\![\vec{y}]\!] \times [\![\vec{s}']\!]$.

A time-frame expansion of the state transition system $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$ is the time unrolling of T into multiple time-indexed copies, denoted $T^t = T(\vec{x}^t, \vec{s}^t, \vec{y}^t, \vec{s}^{t+1})$, the transition relation at time t. In contrast, in the sequel $T^* = T(\vec{x^*}, \vec{s^*}, \vec{y^*}, \vec{s^*}')$ denotes a renamed copy of T with variables $\vec{x}$, $\vec{s}$, $\vec{y}$, and $\vec{s}'$ of T substituted with fresh new variables $\vec{x^*}$, $\vec{s^*}$, $\vec{y^*}$, and $\vec{s^*}'$, respectively.

3. RETENTION SYNTHESIS

The main objective of retention synthesis is to obtain the set of non-Retention Registers (non-RRs) in a power-gating design. In this section, we proposed two algorithms to recognize the non-RRs: one for finding an optimal solution and the other one is a heuristic that is more efficient in both memory usage and runtime. The notation described in Section 2 will continue to be used.

3.1 Problem Statement

Given a state transition system with transition relation $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$, initial states $I(\vec{s})$, power-up sequence $\sigma(\vec{x})$ and power specification (typically in Unified Power Format (UPF)), the retention synthesis problem determines whether non-retention registers exist, and furthermore how to synthesize the new power specification with fewer retention registers. Power-up sequence denotes the specific assignments over primary inputs of several time-frames and is always applied after wakeup.

3.2 Existence of Partial Retention

Given a system starting from a known initial state and a power-up sequence, while assuming the design register r_i (r_i^*) corresponds to state variable s_i^0 (s_i^{*0}) for i ranges from 1 to the total number of registers, the following proposition states the necessary and sufficient condition that the system can be partial retention.

PROPOSITION 1. *Given a system with transition relation $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$, initial states $I(\vec{s})$ and power-up sequence $\sigma(\vec{x})$, let formula $\varphi_{M(p)}$ be*

$$\bigwedge_{t=0}^{p-1} \left(T^t \wedge T^{*t} \wedge (\vec{x}^t = \vec{x^*}^t) \right) \wedge \neg \left(\bigwedge_{t=0}^{p-1} (\vec{y}^t = \vec{y^*}^t) \wedge (\vec{s}^{p-1} = \vec{s^*}^{p-1}) \right) \quad (1)$$

where predicate "=" asserts the bit-wise equivalence of its two argument variable vectors and p denotes the length of $\sigma(\vec{x})$. Assuming the initial state variable vector $\vec{s}^0$ and $\vec{s^*}^0$ are bit-wise equivalent and $\vec{s^*}^0$ can be divided into two subsets $\vec{s_r}$ and $\vec{s_n}$ such that $\vec{s_r} \cup \vec{s_n} = \vec{s^*}^0$ and $\vec{s_r} \cap \vec{s_n} = \phi$;

then the registers $[r_i^* \mid s_i^{*\,0} \in \vec{s_n}]$*, whose corresponding state variables belong to* $\vec{s_n}$*, can be non-retention registers if and only if the formula*

$$\Psi = \varphi_{M(p)} \wedge I(\vec{s}^0) \wedge I(\vec{s_r}) \wedge \sigma(\vec{x}) \tag{2}$$

is unsatisfiable (UNSAT).

In the sequel, we shall call Formula (1), $\varphi_{M(p)}$, the miter formula, which is also shown in Figure 1. In Figure 1, $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$ denotes the full retention system with k registers and initial state $I(\vec{s}^0)$, while $T(\vec{x^*}, \vec{s^*}, \vec{y^*}, \vec{s^*}')$ denotes a partial retention system with n non-RRs whose initial values are not defined. The other m $(= k - n)$ registers have initial values that are the same as the corresponding registers in T. $\vec{s_n}$ and $\vec{s_r}$ represent the initial state variables of non-RRs and RRs respectively. The output of AND gate is asserted to be false.

Proof. ($\Longrightarrow$) Assume Formula (2) is satisfiable with respect to p under assignments $\vec{i} \in [\![\vec{s}^0]\!]$, $\vec{i^*} \in [\![\vec{s_n}]\!]$, then either $(\vec{y}^t = \vec{y^*}^t)$ for some t or $(\vec{s}^{p-1} = \vec{s^*}^{p-1})$ will be unsatisfied; namely, either the primary output or final state has different response between full retention system T and partial retention system T^*. The registers $[r_i^* \mid s_i^{*\,0} \in \vec{s_n}]$ cannot be non-RR.
($\Longleftarrow$) Assume the registers whose corresponding state variables are in $\vec{s_n}$, cannot be non-RR; hence, by definition the unknown value will affect the primary output or final state variable. At least one of the $(\vec{y}^t = \vec{y^*}^t)$ or $(\vec{s}^{p-1} = \vec{s^*}^{p-1})$ will be unsatisfied. Then Formula (2) will be satisfied consequently. ■

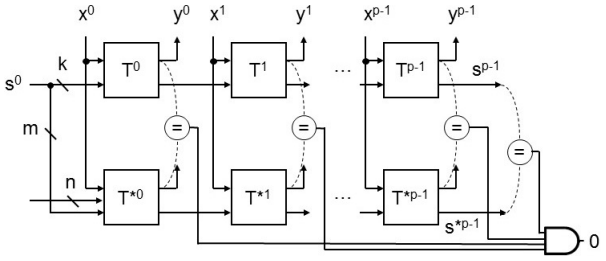

Figure 1: circuit representation of $\varphi_{M(p)}$

3.3 Finding Non-Retention Register

To model the status of registers, additional control variables $\vec{c^*}$ are added such that the following relation hold,

$$F_i^{ctrl} = \{(c_i^* == 0) \Leftrightarrow (s_i^{*\,0} == s_i^{\,0})\},\ \forall i \in [1, |\vec{s}|] \tag{3}$$

$$\Psi_c = \Psi \bigwedge_{i=1}^{|\vec{s}|} \left(F_i^{ctrl}\right) \tag{4}$$

For all the legal i, while c_i^* is equal to 1, the initial state $s_i^{*\,0}$ of register r_i^* is not restricted; otherwise, the $s_i^{*\,0}$ is restricted to the initial state of r_i which is in the full retention system. In other words, register r_i^* is non-RR if and only if c_i^* is equal to 1. The value of control variable models the retention status of the corresponding register.

$$\varphi_A(\vec{c^*}) = \bigwedge_{i=1}^{|\vec{s}|} lit(c_i^*) \tag{5}$$

We use (5), which is the conjunction of literals of all control variables, as unit assumption while verifying the existence of partial retention system. The $\varphi_A(\vec{c^*})$ can be regarded as assignment over $\vec{c^*}$ and represents a candidate of register status. A system with assignment $\varphi_A(\vec{c^*})$ can be partial retention system if not all the variables are equal to zero.

$$\Psi_c \wedge \varphi_A(\vec{c^*}) \tag{6}$$

Formula (6) is UNSAT if and only if the given system can accept the set of register r_i, whose corresponding control variable c_i^* is positive in the unit assumption $\varphi_A(\vec{c^*})(lit(c_i^*) = 1)$, as non-RRs. All the unit clauses in $\varphi_A(\vec{c^*})$ need to be satisfied when solving the instance (6). By changing the assignment $lit(c_i^*)$ alternately, we can model and verify the existence of different choices of non-RRs without the reconstruction of miter formula; therefore, the clauses learned from the previous SAT solving can be reused. While (6) is UNSAT under a specific assumption, the assumptions-based conflict core denotes the unit clauses which take responsibility. The other variables in $\varphi_A(\vec{c^*})$ are assumed to be positive (non-retention) in subsequent SAT solving.

The pseudo-code of the algorithm is presented in Fig. 2. Function Initialize(Ψ_c, $\vec{c^*}$, $\vec{s_n}$) constructs Ψ_c by high-level symbolic simulation [4]. It also sets the initial value of $\vec{c^*}$ to all zeros and $\vec{s_n}$ to empty set. φ_{cc} denotes the returned conflict core which comprises the subset of unit assumptions in $\vec{c^*}$. Function Update($\varphi_A(\vec{c^*})$) chooses a register that has not been verified yet as the candidate of non-RR and modifies the $\varphi_A(\vec{c^*})$ accordingly for incremental SAT solving. The procedure is repeatedly performed until all the registers have been verified. This algorithm takes the advantages of assumptions-based incremental SAT solving that reuse the learned clauses and the conflict core analysis that denote the cause of UNSAT to find the result effectively and efficiently.

FindingNonRetentionRegister
input: $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$, $I(\vec{s})$ and $\sigma(\vec{x})$
output: set of non-retention registers $\vec{s_n}$
begin
01 Initialize(Ψ_c, $\vec{c^*}$, $\vec{s_n}$);
02 φ_{cc} = sat_solve(Ψ_c, $\varphi_A(\vec{c^*})$);
03 **while** (True)
04 **if** (φ_{cc} != NULL)
05 $\vec{s_n} = \vec{s_n} \cup \{\vec{c^*} \setminus \varphi_{cc}\}$;
06 **if** (Update($\varphi_A(\vec{c^*})$) == 0)
07 **break**;
08 φ_{cc} = sat_solve(Ψ_c, $\varphi_A(\vec{c^*})$);
09 **return** $\vec{s_n}$;
end

Figure 2: Algorithm: Finding Non-Retention Registers

3.4 Finding Maximum Number of Non-RRs

The above algorithm is heuristic and can find a set of non-RRs efficiently. On the basis of the result obtained from the above algorithm, we can find the most (optimal) non-RRs by using the cardinality constraints [14], implemented by an adder network that performs the bit-wise addition of the elements in $\vec{c^*}$ and outputs the sum m. Consider the cardinality constraints $\varphi_C(\vec{c^*}, m)$, which restrict the number of control variables that can be set to 1 simultaneously to m,

the formula

$$F_{opt} = \Psi_c \wedge \varphi_C(\vec{c^*}, m) \quad (7)$$

can be applied to find the optimal solution. According to Proposition 1, there are at least m legal non-RRs if there exist an assignment that satisfies $\varphi_C(\vec{c^*}, m)$ and unsatisfies Ψ_c simultaneously. To find such assignments, we continuously compute satisfying assignments of (7) and add blocking clauses to F_{opt}, thus preventing these assignments from being enumerated again until F_{opt} becomes UNSAT. The problem refers to the All-SAT problem [12, 17], which enumerates all satisfying assignments of a propositional logic formula. Previous work [17] indicates that existing solutions to the All-SAT problem are likely to perform unnecessary work because they produce solutions comprise pairwise disjoint cubes instead of overlapping partial assignments. Our proposed algorithm produce the blocking clauses from efficient partial assignments and thus reduce the execution time and memory usage notably.

3.4.1 Algorithm

Fig. 3 shows the pseudo-code of our proposed algorithm for finding the maximum number of non-RRs. In the algorithm, SS_m denotes the solution space that contains different combinations of exactly m registers that should be non-RRs and φ_{block} is the conjunction of blocking clauses. Function Initialize(Ψ_c, $\varphi_C(\vec{c^*}, m)$, φ_{block}) is the procedure to construct Ψ_c and the cardinality constraint $\varphi_C(\vec{c^*}, m)$. Function solve(F) returns a satisfying assignment to F if it exists or a empty set, otherwise. Generalize(φ_{assign}) transforms the satisfying assignment φ_{assign} to blocking clause φ_{gen}. In naive transformation, φ_{gen} is the complement of φ_{assign} : $\varphi_{gen} = \neg\varphi_{assign}$. Some techniques for efficiency enhancement are stated in Section 3.4.2. Function Update($\varphi_{block}, \varphi_{gen}, SS_m$) adds the blocking clause φ_{gen} to φ_{block} and excludes the solution space covered by φ_{gen} from SS_m.

```
FindingOptimalSolution
  input: T(x⃗, s⃗, y⃗, s⃗'), I(s⃗) and σ(x⃗)
  output: s⃗_n with Ψ_c UNSAT
  begin
  01  for (m = |s⃗_n| + 1; m < |c⃗*|; m++) do
  02    Initialize(Ψ_c, φ_C(c⃗*, m), φ_block);
  03    φ_assign = solve(Ψ_c ∧ φ_C(c⃗*, m));
  04    while (True)
  05      if (φ_assign == ∅)
  06        break;
  07      φ_gen = Generalize(φ_assign);
  08      Update(φ_block, φ_gen, SS_m);
  09      φ_assign = solve(Ψ_c ∧ φ_C(c⃗*, m) ∧ φ_block);
  10    if (SS_m == ∅)
  11      return SS_{m-1};
  12  return SS_m;
  end
```

Figure 3: Algorithm: Finding the Optimal Solution

The procedure is repeated until there is no assignment that satisfies (7). If SS_m is not empty, we confirm at least m non-RRs exist. The maximum value of m that satisfies $\varphi_C(\vec{c^*}, m)$ and unsatisfies Ψ_c simultaneously is the maximum number of non-RR and thus is the optimal solution. The desired assignment over $\vec{c^*}$ can be obtained by solving the following formula,

$$\varphi_C(\vec{c^*}, m) \wedge \varphi_{block} \quad (8)$$

3.4.2 Efficiency Enhancement

Since the number of satisfying assignments of a Boolean formula can be exponential with respect to the size of the formula, the above procedure suffers from the problem of memory explosion. From [17], blocking clauses that do not contain the implied variables are extremely short and very beneficial for solver performance. The SAT solving result of our instance depends on the retention status of registers; in addition, the control variables $\vec{c^*}$ imply the other variables. Hence we only need to consider the blocking clauses comprise the control variables. Therefore, instead of the naive technique that constructs blocking clauses from minterms directly, we proposed two new methods: partial and generalized. The partial technique constructs the clauses only from the control variables, while the generalized technique probes each control variable to see whether it can be removed from the blocking clause generated by the partial technique. The procedure Generalize() uses the generalized technique that extracts the partial assignment over the control variables and complement them to construct shorter blocking clause. Empirical comparison among these techniques are provided in Section 4.2.

Fig. 3 shows that the cardinality m is increased iteratively. However, in practice the solution space SS_m grows sharply as m is close to half the number of control variables m_{mid}. In order not to get stuck in the for-loop due to the immense searching space which comprises enormous number of satisfying assignments, we use a heuristic technique to choose m back and forth. The cardinality m closer to the m_{mid} will be chosen later. Thus the for-loop with higher time complexity will not be executed unless the easier ones can be completed.

Even with the above performance enhancements, the optimal algorithm is still considerably more computational intensive than the proposed heuristic algorithm. This is because the solution space in the optimal algorithm increases exponentially with the number of design registers, producing tremendous counts of SAT solving that leads to unacceptable runtime on industrial designs. From our empirical results we observe that the proposed heuristic algorithm can obtain good results within reasonable time; therefore, we suggest using the heuristic algorithm for industrial designs. Comparison between the two proposed algorithms is shown in Section 4.

3.5 Scalability Enhancement

To improve the performance of our algorithm on long sequences and large designs, we use "temporal partitioning" [6]. The long sequence $\sigma(\vec{x})$ is divided into shorter intervals which are analyzed separately. We perform our algorithms on the intervals iteratively with the following criteria. First, a register requires retention if its status is retention in any interval. Second, if the register is over-written before its status becomes retention, it does not need retention. Last, the status of the other registers are inconclusive. To determine the status of inconclusive registers, a different trace that exercises related logic needs to be analyzed. In this manner, the complexity of symbolic simulation and SAT solving can be reduced, thus improving runtime and memory usage. Another improvement is "spatial partitioning",

which divides the design into smaller blocks to handle large designs.

3.6 Discussions

Our analysis is based on input sequences and the results are correct only for the analyzed sequence. If multiple power-up sequences exist, they all need to be analyzed and the non-retention registers are the intersection of all identified non-retention registers in different sequences. If power-up sequences may vary, then the returned analysis is a suggestion to the designer, and further verification is required. Therefore, the proposed algorithms are meant to be used as an analysis tool for designers instead of a verification tool. Compared with pure-formal methods based on constraints, our proposed solution is easier to use and more easily adoptable to industrial simulation-oriented verification flows. Our empirical results show that the analysis results are highly accurate even if only one sequence is analyzed, which can considerably reduce designers' efforts in identifying non-retention registers. For future work, partial Max-SAT solver may have advantages on finding optimal solutions. In addition, performing structural analysis to guide our selection of candidates may also be useful.

4. EXPERIMENTAL RESULTS

The proposed algorithms were programmed in the C language within a commercial symbolic simulator called Insight [1]. MiniSat [8] was selected as the SAT solver. The experiments were conducted on a Linux machine with 2 GHz Xeon processor and 24 GB RAM. The reported runtime includes symbolic simulation. In addition to internally-created and public-domain benchmarks, we also report two case studies on industrial designs from our partner.

4.1 Optimal and Heuristic Algorithms

To compare the optimal and the heuristic algorithms, we created a set of simple designs and chose some pubic-domain designs, (i2c and ecg) from OpenCores and (s13207 and s15850) from ISCAS benchmark. The benchmark circuits are listed in Table 1, where the numbers of registers, primary inputs and primary outputs are also shown. Note that the numbers of registers shown in the fourth column are the numbers of word-level variables, not the total bits of registers. The runtime is constrained to 30 minutes. "CNC" represents "can not complete ": in this case the optimal algorithm cannot find the best solution. The symbolic simulator we used performs certain word-level optimizations before the problem is converted to bit-level Boolean expressions. Therefore, registers whose corresponding symbols are not involved in Ψ automatically become part of non-RRs. Hence the total numbers of non-RRs can be more than that returned by the algorithms. In the table, "NON" reports the number of non-retention registers, "RET" shows the number of retention registers, and "Runtime" shows runtime of the algorithms.

The result shows that even though the optimal algorithm can find the maximal set of non-RRs, its runtime is considerably longer than the heuristic algorithm due to its exhaustive nature and iterative scheme. According to Table 1, the number of non-RRs found by the heuristic method is close to the optimal solution for most cases. Moreover, runtime of the heuristic algorithm is considerably shorter than the optimal algorithm. Therefore, we suggest using the heuristic algorithm for industrial designs.

4.2 Analysis of Optimal Algorithm

We use the same set of designs in the previous experiment to compare the enhancements used in the optimal algorithms, which are naive, partial and generalized, respectively. The numbers of blocking clauses and runtime of the three techniques are shown in Table 2. The naive technique constructs blocking clauses from minterms, while the partial technique constructs the clauses from only the control variables. The generalized technique probes each control variable to see whether it can be removed from the blocking clause generated by partial technique. Blocking clauses with fewer literals are beneficial for finding all satisfying solutions. From Table 2, we can observe that the number of blocking clauses needed to enumerate all the satisfying solutions and runtime are greatly reduced by the partial and generalized techniques.

4.3 Case Studies

We have applied the proposed algorithms to two designs from our industrial partner. In the setup, we make all clocked registers except clock gaters non-retention register candidates. The first design is a gate-level IO block with 5885 FFs. We analyzed a power-up sequence that is 1445 cycles long and identified 1569 non-retention FFs, 4146 retention FFs, and 170 inconclusive FFs. The designer inspected the results and found that all but five non-retention FFs are correct. For the five in question, four of them are confirmed can be non-retention after detailed analysis, and one of them is a control signal that is used before the start of our analysis window of the given trace. Because that signal is no longer needed when our analysis starts, our algorithm determined it to be non-retention.

The second design is a wireless communication block in RTL with 3021 word-level variables (21817 bits) and several clock domains. We analyzed a trace that is approximately 10K cycles long using 3h31m and found 1805 non-retention variables (12819 bits), 495 retention variables (3518 bits), and 718 inconclusive variables (5480 bits). The result suggests that 59% of design FFs do not need retention, which can considerably reduce leakage power when the block is powered down.

5. CONCLUSIONS

Using partial instead of full retention in low-power designs not only reduces static power consumption but also improves design performance due to smaller area. In this paper we proposed an efficient algorithm to automatically identify non-retention registers with a unified problem formulation. We also proposed an optimal algorithm that can find the maximal set of non-retention registers. Several all-SAT techniques are explored for obtaining optimal solution. Our experimental results show that the heuristic solution provides high quality results that are on a par with the optimal results, and two case studies from our partner show that our techniques can identify non-retention registers effectively and efficiently in industrial designs.

Acknowledgment

The authors thank Yushi Tian and Jeff (Zhenfei) Lu from Broadcom for their valuable feedback and comments on this work. T.W.C. and J.H.J. were supported in part by MOST under grant 103-2221-E-002-273.

Table 1: COMPARISON OF OPTIMAL AND HEURISTIC ALGORITHMS

Design	Benchmark Characteristics			Optimal Algorithm			Heuristic Algorithm		
	#PI	#PO	#Reg	NON	RET	Runtime	NON	RET	Runtime
retsyn1	4	3	5	1	2	0.00045s	1	2	0.00025s
retsyn4	5	3	14	2	11	2.33213s	2	11	0.09935s
retsyn6_2	3	2	8	5	3	0.02128s	5	3	0.00500s
retsyn7_1	3	2	6	2	4	0.00469s	2	4	0.00089s
retsyn8_1	3	2	7	3	4	0.00865s	2	5	0.00092s
retsyn9_1	2	1	7	2	5	0.01157s	2	5	0.00213s
retsyn9_2	2	1	7	5	2	0.01443s	5	2	0.00445s
retsyn10_1	2	1	8	5	3	0.01843s	4	4	0.00332s
retsyn12	6	2	24	16	8	CNC	16	8	0.17430s
i2c_bit_controller	10	7	18	6	8	173.42674s	6	8	0.17143s
ecg	6	4	74	16	44	CNC	16	44	3.59300s
s13207*	62	152	638	–	–	–	567	71	3.83200s
s15850*	77	150	534	–	–	–	495	39	6.86100s
case_study_1*	317	179	5885	–	–	–	1569	4146	1h40m
case_study_2	111	83	3021	–	–	–	1805	495	3h31m

*bit-level design

Table 2: COMPARISON OF TECHNIQUES USED IN OPTIMAL ALGORITHM

Design	Naive		Partial		Generalized	
	Runtime (sec)	#block	Runtime (sec)	#block	Runtime (sec)	#block
retsyn1	0.00111	38	0.00045	5	0.00042	3
retsyn4	CNC	>80000	2.33213	455	5.12242	376
retsyn6_2	55.93945	62080	0.02128	92	0.03402	63
retsyn7_1	1.30241	8504	0.00469	55	0.00394	31
retsyn8_1	11.32251	36168	0.00865	115	0.00748	63
retsyn9_1	73.79333	72288	0.01157	84	0.01454	63
retsyn9_2	14.83431	23808	0.01443	28	0.01816	21
retsyn10_1	91.91383	72224	0.01843	92	0.0235	63
retsyn12	CNC	>60000	CNC	>120000	CNC	>120000
i2c_bit_controller	CNC	>60000	173.33446	12910	188.26154	7098
ecg	CNC	>10000	CNC	>20000	CNC	>15000

6. REFERENCES

[1] Avery Design Systems Inc., `http://www.avery-design.com`.

[2] K.-H. Chang, Y.-T. Liu, C. S. Browy, and C.-L. Huang. Systems and methods for partial retention synthesis, US Patent 8938705, 2014.

[3] Y.-G. Chen, Y. Shi, K.-Y. Lai, G. Hui, and S.-C. Chang. Efficient multiple-bit retention register assignment for power gated design: Concept and algorithms. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 309–316, 2012.

[4] H.-Z. Chou, K.-H. Chang, and S.-Y. Kuo. Handling don't-care conditions in high-level synthesis and application for reducing initialized registers. In *Proceedings of the Design Automation Conference (DAC)*, pages 412–415, 2009.

[5] H.-Z. Chou, K.-H. Chang, and S.-Y. Kuo. Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(4):646–651, April 2010.

[6] H.-Z. Chou, H. Yu, K.-H. Chang, D. Dobbyn, and S.-Y. Kuo. Finding reset nondeterminism in RTL designs - scalable X-analysis methodology and case study. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 1494–1499, 2010.

[7] A. Darbari, B. Al-Hashimi, D. Flynn, and J. Biggs. Selective state retention design using symbolic simulation. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 1644–1649, 2009.

[8] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 502–518, 2003.

[9] S. Greenberg, J. Rabinowicz, and E. Manor. Selective state retention power gating based on formal verification. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(3):807–815, December 2014.

[10] S. Greenberg, J. Rabinowicz, R. Tsechanski, and E. Paperno. Selective state retention power gating based on gate-level analysis. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4):1095–1104, April 2014.

[11] J.-H. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. Huang. To SAT or not to SAT: Scalable exploration of functional dependency. *IEEE Transactions on Computers*, 59(4):457–467, April 2010.

[12] K. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 250–264, 2002.

[13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference (DAC)*, pages 530–535, 2001.

[14] A. Smith, A. Veneris, and A. Viglas. Design diagnosis using Boolean satisfiability. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 218–223, 2004.

[15] Y. Tian, A. Nilipour, and A. Thiriveedhi. Partial state retention verification - challenges and techniques. In *Proceedings of the Design Automation Conference (DAC)*, Designer Track Poster 15D.6, 2013.

[16] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 466–483, 1970.

[17] Y. Yu, P. Subramanyan, N. Tsiskaridze, and S. Malik. All-SAT using minimal blocking clauses. In *Proceedings of the International Conference on VLSI Design*, pages 86–91, 2014.