Using Unsatisfiable Cores to Estimate Maximum Power in CMOS Combinational Circuits

ID:235

Abstract

With the growing integration scales in circuits, the maximum power estimation in CMOS combinational circuits becomes a major design concern for circuit reliability. Previous works formulate it in 0-1 integer linear programming (ILP). However, their approaches do not explicitly express the constraints among variables in the objective function, namely *power constraints*. The power constraints intuitively characterize the maximum power dissipation. Therefore they cost more time in ILP solving. Motivated by the applications of unsatisfiable cores in optimizing large-scale search problems, we propose a core-guided approach – `TEMPO` to intuitively express the power constraints. `TEMPO` includes two processes – *refine* and *abstract*, the former for extracting power constraints from unsatisfiable cores and the latter for searching new unsatisfiable cores. The experimental results in the typical MCNC benchmarks show that our approach can improve the speed of solution. Especially, our approach has good performance in challenging benchmarks.

1 Introduction

The *maximum power estimation* is to estimate the maximum or peak power in circuits. The growing integration scales in Very-large-scale integration (VLSI) and the recent surge in the deployment and initialization of portable electronic devices has brought power dissipation to the forefront as a major design concern. Circuit reliability is related to maximum or peak power. Excessive power dissipation may cause runtime errors and device destruction due to overheating, because excessive instantaneous current through the power and ground (P&G) nets may result in performance degradation due to large voltage drops along the P&G nets [Sagahyroon and Aloul, 2007]. Hence, the estimation of maximum power in VLSI circuits is essential for determining the appropriate packaging and cooling techniques and for optimizing the power and ground routing networks [Devadas *et al.*, 1992; Halter and Najm, 1997; Li *et al.*, 2002; Pedram, 1996; Wang and Roy, 1996; Wu *et al.*, 2001]. However, maximum power estimation is NP-complete problem [Devadas *et al.*, 1992] in *Complementary Metal Oxide Semiconductor (CMOS)* combinational circuits.

A possible solution is to simulate all possible input vectors exhaustively. Unfortunately, it is impractical for the circuit with a large number of inputs because the number of all possible input vectors grows exponentially as the size of inputs grows. Devadas *et al.* [1992] transformed the maximum power estimation to a weighted MaxSAT problem. Wang and Roy [1996] used an Automatic Test Generation (ATG) based approach to obtain a lower bound of the maximum power dissipation. Li *et al.* [2002] utilized a cluster-based Automatic Test Pattern Generation (ATPG) approach to solve it. Sagahyroon and Aloul [2007] proposed a method that formulates this problem as 0-1 integer linear programming (ILP). It still is a state-of-the-art method with the help of `CPLEX` [Cplex, 2010], a powerful ILP solver. However, unfortunately, there are many hard instances still unsolved within acceptable time by this method.

Sagahyroon and Aloul [2007] searched for an input vector pair $\{V_1, V_2\}$, tending to maximize the dynamic power dissipation instead of estimating the maximum power directly. Dynamic power dissipation is a major contributor to the total power dissipated in CMOS combinational circuits. The circuit's logical behavior is represented by CNF constraints $\mathcal{C}_1$ ($\mathcal{C}_2$) after V_1 (V_2) is applied. CNF constraints $\mathcal{C}_3$ represent XOR gates between the outputs of gates in $\mathcal{C}_1$ and $\mathcal{C}_2$. If an XOR gate outputs logic 1, it means this gate occurs switching when the input vector V_1 is followed by input vector V_2. Otherwise, it means this gate does not occur switching. Here the weighted sum of XOR outputs is defined as the objective function $\mathcal{F}$ where the weight is the fanout of the circuit gate corresponding. Thus, they used an ILP solver to maximize $\mathcal{F}$, subjecting to $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$.

However, the approach mentioned above does not express the power constraint explicitly, when power constraints intuitively characterize the maximum power dissipation (We define a *power constraint* as a clause in which each variable is in the objective function $\mathcal{F}$). $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$ expresses the circuit's logical behavior and implies power constraint after enumerating all of the input vector pairs. There is no clause in $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$ that contains more than one variable of the objective function. That slows down to ILP solving.

We propose a method, `TEMPO`, that focuses on finding the power constraints to improve the performance. We for-

mulate the maximum power estimation problem into ILP by [Sagahyroon and Aloul, 2007] and then use our TEMPO to append power constraints into ILP and solve it.

The main contributions of this paper are as follows. Firstly, we propose a core-guided approach – TEMPO[1] to estimate maximum power in CMOS combinational circuits. TEMPO searches power constraints which intuitively characterize the maximum power dissipation. Secondly, we propose Refine and Abstract processes in TEMPO. The Refine extracts power constraints from unsatisfiable cores and the Abstract uses clause-selector variables to search new unsatisfiable cores. Thirdly, we assess our approach to MCNC benchmarks. The experimental results show that TEMPO is outstanding, especially in challenging instances solved over 1000 seconds by CPLEX.

This paper is organized as follows. Section 2 gives some basic concepts about SAT and ILP, and presents the problem formulation. Section 3 describes our approach TEMPO in detail. Section 4 reports the experimental investigation. Section 5 presents some related works on the maximum power estimation. Section 6 contains the concluding remarks.

2 Preliminaries

In this section, we introduce the notations and backgrounds. The symbols in this section are standard in all.

2.1 Boolean SAT and 0-1 Integer Linear Programming (ILP)

Let a *literal* l be a propositional variable x or its negation $\overline{x}$, $var(l)$ be a variable of the literal l, a *clause* c be a disjunction of literals, and a *Conjunctive Normal Form (CNF)* $\mathcal{C}$ formula be a conjunction of clauses. A CNF formula is also represented by the set of its clauses, and a clause by the set of its literals. A CNF is satisfiable if there exists an assignment satisfy the CNF formula. Otherwise, the CNF is unsatisfiable. The *clause-selector variable* y [Liffiton and Sakallah, 2008] is added to a zclause c, denoted as $\hat{c} = (\overline{y} \vee c)$. Assigning a particular y the value true implies the original clause c, essentially enabling it. Conversely, assigning y false has the effect of disabling c from CNF. An *unsatisfiable core* p of a CNF $\mathcal{C}$ is a subset of $\mathcal{C}$ and p is unsatisfiable. Incremental SAT solvers accept assumption literals [Eén and Sörensson, 2003], which are used as forced decision and are only valid during the next incremental satisfiability check, thus abandoned in later checks. When an incremental SAT solver derives unsatisfiability, it knows which assumption literals are the unsatisfiable core. Such literals are called *failed assumptions* [Eén and Sörensson, 2003].

Pseudo-Boolean (PB) Constraints are linear inequalities with integer coefficients that can be expressed in the normalized form [Barth, 1995; Aloul *et al.*, 2002] of $a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b$ where $a_i, b \in Z$ and x_i are Boolean variables. A clause is a special form of a PB constraint. PB constraints represent 0-1 ILP inequalities. Subject to a given set of PB constraints, 0-1 ILP is the maximization (or minimization) of an objective function which consists of a linear combination of the problem's variables.

2.2 Problem Description and Formulation

In COMS combinational circuits, switching power contribute mostly to the circuit power. The relationship between energy and logic behavior of the circuit is $E = 0.5CV_{\mathrm{dd}}^2 S_{\mathrm{G}}$ where E is the energy dissipated by the CMOS gate, C is the output capacitance for the gate, S_G is the total number of gate output transitions, and V_{dd} is the voltage of the power source and also the assumed voltage swing of the node .

The capacitance C is assumed to be directly proportional to the fanout f of the gate [Devadas *et al.*, 1992; Manich and Figueras, 1997]. Since the product $(f \cdot S_G)$ is proportional to the power consumed, to maximize power dissipation, we search for an input vector pair $\{V_1, V_2\}$, that tends to maximize the weighted sum of the gates output transitions. The weighted switching activity (W) of the circuit can be approximated using the equation: $W = \sum_{\text{all gates}} f_i \left(g_i(V_1) \oplus g_i(V_2)\right)$ where f_i is the fanout of gate g_i, $g_i(V_1)$ is the output of g_i when V_1 is applied, and $g_i(V_2)$ is the output of g_i when V_2 is applied. Here, when the input vector V_1 is followed by the input vector V_2, the summation of "XOR equal to 1" is the same as the number of switching nodes. Note that a *zero-delay model* is assumed for all the gates in the circuit.

Zero-Delay Model: Under the zero-delay model, transitions are assumed to happen instantaneously. Therefore, glitches cannot occur at the outputs of any of the gates, and each gate can make at most one transition: 0 to 1 or 1 to 0.

In [Sagahyroon and Aloul, 2007], they propose a modeling method, namely MPE-ILP, to formulate the maximum power estimation into the ILP problem. We use example 2.1 for further explaining the modeling method.

Circuit A: A CNF $\mathcal{C}_1$ representing the circuit's logical behavior after the application of input vector V_1.

Circuit B: A CNF $\mathcal{C}_2$ representing the circuit's logical behavior after the application of input vector V_2.

Note that the set of constraints in Circuit A and that of Circuit B are identical and corresponding variables of two circuits are named differently.

XOR gates: A CNF $\mathcal{C}_3$ representing XOR gates between the outputs of gates in Circuit A and Circuit B. The number of XOR gates equals the number of gates in the original circuit. An XOR gate output of logic 1 indicates that a transition(0 to 1 or 1 to 0) has occurred at the output of the gate in the original circuit applied the vector V_1 followed by vector V_2.

Objective function: A PB objective function $\mathcal{F}$ which specifies the weights of the XOR outputs. Weights are computed based on the capacitance of the gate, and the capacitance is assumed to be proportional to the fanout of the gate.

Circuit constraints characterize the constraint of logic gates in the circuit. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ are circuit constraints. Let $\mathcal{C}_{all}$ be $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$. Therefore, the maximum power estimation is modeled as the following ILP problem: $Maximize\ \mathcal{F}\ s.t.\ \mathcal{C}_{all}$.

Here we present an illustrative example. The original circuit is shown in Figure 1. What is more, the corresponding circuit A, B and XOR gates are also presented in Figure 1.

[1]Due to space limit, omitted data, code, and supporting materials are provided in the online appendix(http://tinyurl.com/IJCAI19-235)

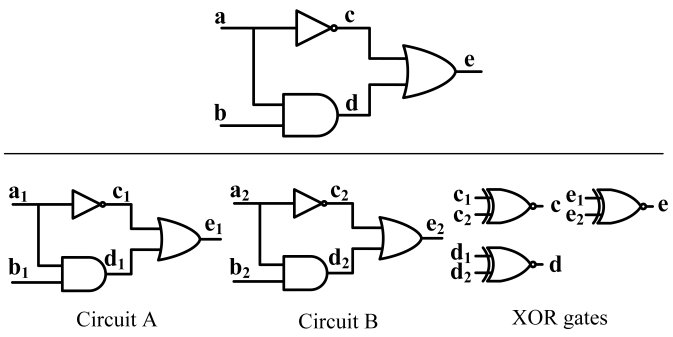

Figure 1: The Circuit of Illustrative Example

Example 2.1. *Given a circuit in Figure 1.*

$\mathcal{C}_1$ *in the circuit A is*

$$\begin{Bmatrix} c_1 : \{\overline{a_1}, \overline{c_1}\}, & c_2 : \{a_1, c_1\} & \\ c_3 : \{\overline{a_1}, \overline{b_1}, d_1\}, & c_4 : \{a_1, \overline{d_1}\}, & c_5 : \{b_1, \overline{d_1}\} \\ c_6 : \{c_1, d_1, \overline{e_1}\}, & c_7 : \{\overline{c_1}, e_1\}, & c_8 : \{\overline{d_1}, e_1\} \end{Bmatrix}$$

$\mathcal{C}_2$ *in the circuit B is*

$$\begin{Bmatrix} c_9 : \{\overline{a_2}, \overline{c_2}\}, & c_{10} : \{a_2, c_2\} & \\ c_{11} : \{\overline{a_2}, \overline{b_2}, d_2\}, & c_{12} : \{a_2, \overline{d_2}\}, & c_{13} : \{b_2, \overline{d_2}\} \\ c_{14} : \{c_2, d_2, \overline{e_2}\}, & c_{15} : \{\overline{c_2}, e_2\}, & c_{16} : \{\overline{d_2}, e_2\} \end{Bmatrix}$$

$\mathcal{C}_3$ *in XOR gates is*

$$\begin{Bmatrix} c_{17} : \{\overline{c_1}, c_2, c\}, & c_{18} : \{c_1, \overline{c_2}, c\} \\ c_{19} : \{\overline{c_1}, \overline{c_2}, \overline{c}\}, & c_{20} : \{c_1, c_2, \overline{c}\} \\ c_{21} : \{\overline{d_1}, d_2, d\}, & c_{22} : \{d_1, \overline{d_2}, d\} \\ c_{23} : \{\overline{d_1}, \overline{d_2}, \overline{d}\}, & c_{24} : \{d_1, d_2, \overline{d}\} \\ c_{25} : \{\overline{e_1}, e_2, e\}, & c_{26} : \{e_1, \overline{e_2}, e\} \\ c_{27} : \{\overline{e_1}, \overline{e_2}, \overline{e}\}, & c_{28} : \{e_1, e_2, \overline{e}\} \end{Bmatrix}$$

$\mathcal{F}$ *is* $(c + d + e)$.

Subjecting to $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$*, the maximum of* $\mathcal{F}$ *is 2. The solution is* $\{a_1, b_1\} = \{1, 1\}$, $\{a_2, b_2\} = \{0, 0\}$.

3 New Approach – TEMPO

In this section, we introduce our approach – TEMPO. We first give the overview of TEMPO, then show the detail.

3.1 Overview

The pseudo-code of the TEMPO is presented in algorithm 1. The inputs of TEMPO are $\mathcal{C}_{all}$ and $\mathcal{F}$, and the output is the maximum of $\mathcal{F}$ subjecting to $\mathcal{C}_{all}$. Note that H expresses the constraint of $\mathcal{F}$, explicitly $\mathcal{C}_{all} \models H$.

There are three phases in TEMPO. The first phase is initialization. We construct a new CNF $\hat{C}$. Each clause $\hat{c_i}$ in $\hat{C}$ is $\overline{y_i} \vee c_i$, where c_i is a clause in $\mathcal{C}_{all}$ (Line 1) and y_i is a clause-selector variable of c_i. Then, Y is the conjunction of all clause-selector variables (Line 2) and O is the conjunction of variables in $\mathcal{F}$ (Line 3). The second phase is finding constraints among variables in $\mathcal{F}$. In each iteration, we get a new unsatisfiable core P by a SAT solver (Line 6). $Y \cup O$ is the assumption. P is the reason why the conjunction of O and the enable clauses in $\mathcal{C}_{all}$ are unsatisfiable. Then, we propose

Algorithm 1: TEMPO $(\mathcal{C}_{all}, \mathcal{F})$

Input: $\mathcal{C}_{all}$ is CNF and $\mathcal{F}$ is objective function
Output: H is the approximate CNF of $\mathcal{C}_{all}$

1 $\hat{C} \leftarrow addClauseSelectorVariables(\mathcal{C}_{all})$
2 $Y \leftarrow \bigwedge_{i=1}^{n} y_i$
3 $O \leftarrow \bigwedge_{l \in \mathcal{F}} l$
4 $H \leftarrow \emptyset$
5 **while** $\hat{C} \cup Y \cup O$ *is* $UNSAT$ **do**
6 $\quad P \leftarrow FailedAssumption(\hat{C}, Y \cup O)$
7 $\quad H \leftarrow Refine(H, P)$
8 $\quad \hat{C}, Y \leftarrow Abstract(\hat{C}, Y, P)$
9 **return** $Maximize(\mathcal{C}_{all} \cup H, \mathcal{F})$

Refine and Abstract processes. In Refine, we find a constraint among variables in $\mathcal{F}$ with the property $\mathcal{C}_{all} \models H$ (Line 7). In Abstract, we block P for avoiding searching it again (Line 8). In the third phase, we invoke an ILP solver to maximize $\mathcal{F}$ subjecting to $\mathcal{C}_{all} \cup H$. We use an example to explain this algorithm 1.

Example 3.1. *Following the example 2.1,* $\hat{C}$ *is*

$$\begin{Bmatrix} \hat{c_1} : \{\overline{y_1}, \overline{a_1}, \overline{c_1}\}, & \hat{c_2} : \{\overline{y_2}, a_1, c_1\}, \\ \hat{c_3} : \{\overline{y_3}, \overline{a_1}, \overline{b_1}, d_1\}, & \hat{c_4} : \{\overline{y_4}, a_1, \overline{d_1}\}, \\ \hat{c_5} : \{\overline{y_5}, b_1, \overline{d_1}\}, & \\ \hat{c_6} : \{\overline{y_6}, c_1, d_1, \overline{e_1}\}, & \hat{c_7} : \{\overline{y_7}, \overline{c_1}, e_1\}, \\ \hat{c_8} : \{\overline{y_8}, \overline{d_1}, e_1\}, & \\ \hat{c_9} : \{\overline{y_9}, \overline{a_2}, \overline{c_2}\}, & \hat{c_{10}} : \{\overline{y_{10}}, a_2, c_2\}, \\ \hat{c_{11}} : \{\overline{y_{11}}, \overline{a_2}, \overline{b_2}, d_2\}, & \hat{c_{12}} : \{\overline{y_{12}}, a_2, \overline{d_2}\}, \\ \hat{c_{13}} : \{\overline{y_{13}}, b_2, \overline{d_2}\}, & \\ \hat{c_{14}} : \{\overline{y_{14}}, c_2, d_2, \overline{e_2}\}, & \hat{c_{15}} : \{\overline{y_{15}}, \overline{c_2}, e_2\}, \\ \hat{c_{16}} : \{\overline{y_{16}}, \overline{d_2}, e_2\}, & \\ \hat{c_{17}} : \{\overline{y_{17}}, \overline{c_1}, c_2, c\}, & \hat{c_{18}} : \{\overline{y_{18}}, c_1, \overline{c_2}, c\}, \\ \hat{c_{19}} : \{\overline{y_{19}}, \overline{c_1}, \overline{c_2}, \overline{c}\}, & \hat{c_{20}} : \{\overline{y_{20}}, c_1, c_2, \overline{c}\}, \\ \hat{c_{21}} : \{\overline{y_{21}}, \overline{d_1}, d_2, d\}, & \hat{c_{22}} : \{\overline{y_{22}}, d_1, \overline{d_2}, d\}, \\ \hat{c_{23}} : \{\overline{y_{23}}, \overline{d_1}, \overline{d_2}, \overline{d}\}, & \hat{c_{24}} : \{\overline{y_{24}}, d_1, d_2, \overline{d}\}, \\ \hat{c_{25}} : \{\overline{y_{25}}, \overline{e_1}, e_2, e\}, & \hat{c_{26}} : \{\overline{y_{26}}, e_1, \overline{e_2}, e\}, \\ \hat{c_{27}} : \{\overline{y_{27}}, \overline{e_1}, \overline{e_2}, \overline{e}\}, & \hat{c_{28}} : \{\overline{y_{28}}, e_1, e_2, \overline{e}\} \end{Bmatrix}$$

Y *is* $\bigwedge_{i=1}^{28} y_i$ *and* O *is* $c \wedge d \wedge e$*. In the first iteration,* P *is* $\{\overline{c}, \overline{d}, \overline{e}, \overline{y_1}, \overline{y_4}, \overline{y_7}, \overline{y_8}, \overline{y_9}, \overline{y_{12}}, \overline{y_{15}}, \overline{y_{16}}, \overline{y_{20}}, \overline{y_{24}}, \overline{y_{27}}\}$. H *is* $\{\{\overline{c}, \overline{d}, \overline{e}\}\}$. Y *is* $\{y_2, y_3, y_5, y_6, y_{10}, y_{11}, y_{13}, y_{14}, y_{17}, y_{18},$

y_{19}, y_{21}, y_{22}, y_{23}, y_{25}, y_{26}, $y_{28}\}$ *and the block clause* $\{\overline{y_1}$, $\overline{y_4}$, $\overline{y_7}$, $\overline{y_8}$, $\overline{y_9}$, $\overline{y_{12}}$, $\overline{y_{15}}$, $\overline{y_{16}}$, $\overline{y_{20}}$, $\overline{y_{24}}$, $\overline{y_{27}}\}$ *is appended to* $\hat{C}$.

3.2 Power Constraints

Definition 1. *A* power constraint c_p *is a clause in which each variable is in the objective function* $\mathcal{F}$.

Intuitively power constraints are constraints in ILP that express which gates to occur switching. In other words, power constraints characterize constraints of variables in $\mathcal{F}$ because they indicate whether corresponding gates occur switching in the circuit applied V_1 followed by V_2. For instance, $\{\overline{c}, \overline{d}, \overline{e}\}$ in example 3.1 is a power constraint, because c, d and e are variables in $\mathcal{F}$. Power constraints intuitively express the maximum power dissipation.

Proposition 1. $\mathcal{C}_{all} \models c_p$.

Proposition 1 shows that $\mathcal{C}_{all}$ completely expresses the constraints of maximum power estimation in a circuit. Therefore, we can find power constraints from $\mathcal{C}_{all}$. There is no power constraint in $\mathcal{C}_{all}$ since MPE-ILP forces on expressing the circuit's logical behavior.

We extract a power constraint from an unsatisfiable core and add it into $\mathcal{C}_{all}$. As shown in example 3.1, extracting unsatisfiable core, we can add a power constraint $\overline{c} \vee \overline{d} \vee \overline{e}$ into $\mathcal{C}_{all}$. It forbids the partial assignment $c \wedge d \wedge e$. In other words, the maximum value of $\mathcal{F}$ is less than 3. Table 1 shows the true table of example 2.1. $\{a_1, b_1\}$ is V_1, $\{a_2, b_2\}$ is V_2. $\{c, d, e\}$ describes the corresponding gates occur switching when the V_1 is followed by V_2, and $\mathcal{F}$ is the value of objective function. As we see, c, d and e cannot be assigned true at the same time for any input pairs, and the maximum of $\mathcal{F}$ is 2. After adding $\overline{c} \vee \overline{d} \vee \overline{e}$, the ILP solver can cut the search space in which c, d and e are assigned true at the same time.

Table 1: The true table of example 2.1.$\{a_1, b_1\}$ is V_1, $\{a_2, b_2\}$ is V_2. $\{c, d, e\}$ describes the corresponding gates occur switching when the V_1 is followed by V_2 and $\mathcal{F}$ is the value of objective function.

a_1	b_1	a_2	b_2	c	d	e	$\mathcal{F}$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	2
0	0	1	1	1	1	0	2
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	1	2
0	1	1	1	1	1	0	2
1	0	0	0	1	0	1	2
1	0	0	1	1	0	1	2
1	0	1	0	0	0	0	0
1	0	1	1	0	1	1	2
1	1	0	0	1	1	0	2
1	1	0	1	1	1	0	2
1	1	1	0	0	1	1	2
1	1	1	1	0	0	0	0

3.3 Refinement and Abstraction

The pseudo-code of the `Refine` is presented in Algorithm 2. The idea of `Refine` is finding a power constraint. The power constraints are stored in H. At the beginning of `TEMPO`, H is $\emptyset$, so we have $\mathcal{C}_{all} \models H$. In `Refine`, the refined clause is a power constraint and also holds the property: $\mathcal{C}_{all} \models refinedClause$. Therefore, we hold $\mathcal{C}_{all} \models H$. Following example 3.1, we introduce the execution of Algorithm 2.

Algorithm 2: `Refine` (H, P)

Input: H is CNF, P is failed assumptions
Output: The H after refinement

```
refinedClause ← ∅
for each literal l ∈ P do
    if var(l) is not clause-selector variable then
        refinedClause ← refinedClause ∪ l
return H ∪ refinedClause
```

Example 3.2. *In the first iteration in* `TEMPO`*, the inputs of* `Refine` *are that* H *is* $\emptyset$ *and* P *is* $\{\overline{c}, \overline{d}, \overline{e}, \overline{y_1}, \overline{y_4}, \overline{y_7}, \overline{y_8}, \overline{y_9}, \overline{y_{12}}, \overline{y_{15}}, \overline{y_{16}}, \overline{y_{20}}, \overline{y_{24}}, \overline{y_{27}}\}$*. The variables in refined clause are not clause-selector variables. So that,* $refindedClause$ *is* $\{\overline{c}, \overline{d}, \overline{e}\}$.

In example 3.2, we have $refindedClause \leftarrow \{\overline{c}, \overline{d}, \overline{e}\}$. It means that c, d and e cannot be true at the same time. Therefore, the maximum of $\mathcal{F}$ is less than 3.

The pseudo-code of the `Abstract` is presented in Algorithm 3. The idea of `Abstract` is to search a new unsatisfiable core rather than the previous unsatisfiable core again. A naive method to do that is to disable one of the clauses in $\mathcal{C}_{all}$ randomly which drives this unsatisfiable core. However, it performances bad. So, instead, we use clause-selector variables to disable clauses. Using clause-selector variables, we can disable dynamically clauses when searching a new unsatisfiable core. When we disable one of the clauses in $\mathcal{C}_{all}$, we abstract the $\mathcal{C}_{all}$. Following above example 3.1, we introduce the execution of Algorithm 3.

Example 3.3. *In the first iteration in* `TEMPO`*, the inputs of* `Abstract` *are mentioned in example 3.1. We remove clause-selector variables in* P *from* Y*, then* Y *becomes* $\{y_2$, y_3, y_5, y_6, y_{10}, y_{11}, y_{13}, y_{14}, y_{17}, y_{18}, y_{19}, y_{21}, y_{22}, y_{23}, y_{25}, y_{26}, $y_{28}\}$*. And the* $blockClause$ *is the set of negative literals of clause-selector variables in* P*, so* $blockClause$ *is* $\{\overline{y_1}$, $\overline{y_4}$, $\overline{y_7}$, $\overline{y_8}$, $\overline{y_9}$, $\overline{y_{12}}$, $\overline{y_{15}}$, $\overline{y_{16}}$, $\overline{y_{20}}$, $\overline{y_{24}}$, $\overline{y_{27}}\}$.

As shown in example 3.3, we add block clause in $\hat{C}$. We can force one of the clause-selector variables in block clause to be false. In other words, we disable one of the clauses which cause an unsatisfiable core P. Therefore, we can avoid searching for P again.

4 Experiment

This section evaluates the algorithm proposed in this paper. The experiments were performed on an Intel Xeon E7-4830 2.10GHz with 126GByte of memory and running Ubuntu 16.04. The time limit was set to 3600s. `TEMPO` was implemented on top of MiniSAT 2.2.0[2] [Eén and Sörensson, 2003]. The ILP solver was `CPLEX` 12.8.0.0[3] [Cplex, 2010], a

[2]https://github.com/niklasso/minisat

[3]https://www.ibm.com/analytics/CPLEX-optimizer

Algorithm 3: `Abstract` $(\hat{C}, Y, P)$

Input: $\hat{C}is$ CNF, Y is the conjunction of clause-selector variables, P is failed assumptions

Output: The $\hat{C}$ after blocking P and the Y after abstraction

$blockClause \leftarrow \emptyset$
for *each literal* $l \in P$ **do**
  if *var(l) is clause-selector variable* **then**
    $Y \leftarrow Y \setminus var(l)$
    $blockClause \leftarrow blockClause \cup l$
$\hat{C} \leftarrow \hat{C} \cup blockClause$
return $\hat{C}, Y$

Table 2: Runtimes [sec] for challenging instances except solved timeout by both CPLEX and TEMPO.. "#V" is the number of variables in ILP, "#C" is the number of constraints in ILP. For CPLEX and TEMPO, "-" means that an instance cannot be solved in 1 hour.

	Instance	#V	#C	CPLEX	TEMPO
Multi-level logic	C1908	2124	6316	**963**	1715
	max1024	5018	15370	1979	**1084**
	table3	10666	32734	1185	**1090**
	apex3	12660	38220	1086	**1052**
	apex1	13590	41302	**1530**	1835
	bcc	14254	44348	**2976**	-
	bcb	14665	45596	2157	**1597**
	bca	16105	50146	-	**2725**
Two-level logic (hard)	max1024	7373	22334	-	**771**
	table3	10348	31796	-	**1069**
	bcc	10978	34058	1101	**914**
	bcb	11518	35738	1150	**1018**
	apex3	11700	35356	1118	**1105**
	apex1	13260	40320	**1588**	1611
	prom2	16227	49152	**2124**	2283

commercial tool that is considered as one of the best available generic ILP solvers. We used the default settings for CPLEX except that the number of threads is limited to 1. We used the typical MCNC [Yang, 1991] combinational benchmark circuits, which contain Multi-level logic, two-level logic (all) and two-level logic (hard). Each instance was sensitized using "ABC" [Brayton and Mishchenko, 2010] into a circuit consisting of 2-input AND, OR and inverter gates. The experimental results are shown as follows. On the one hand, the benchmarks were solved by TEMPO. At line 9 of algorithm 1, it invokes CPLEX to maximize $\mathcal{F}$ subjecting to $\mathcal{C}_{all} \cup H$. On the other hand, the benchmarks were solved by CPLEX.

We show the experimental results in Figure 2. The X-axis indicates the time in seconds taken by TEMPO, and the Y-axis indicates the time taken by CPLEX. Points above the diagonal indicate advantages for TEMPO. Overall, the experimental results show that TEMPO has better performance, especially in two-level logic (all) circuits. It is comparable with CPLEX in Multi-level and two-level logic (hard) circuits.

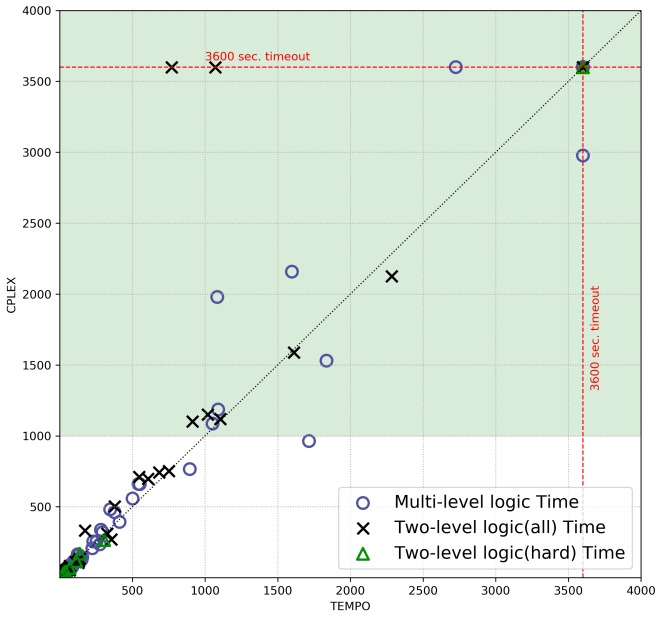

Figure 2: Runtimes [sec] for MCNC benchmarks. The X-axis indicates the time in seconds taken by TEMPO, and the Y-axis indicates the time taken by CPLEX. Points above the diagonal indicate advantages for TEMPO. The instance solved overs 1000 seconds by CPLEX is called a challenging instance. The "challenging instances" are shown in the green area.

We call instances solved over 1000 seconds by CPLEX as "challenging instances". We focus on the challenging instances shown in the green area in figure 2. Most of the points of challenging instances are above the diagonal. Furthermore, we show the running time for each challenging instances in table 2 except instances solved out of 3600 seconds by both CPLEX and TEMPO. As we see, TEMPO has better performance in most instances.

The maximum number of logic gates that are presented (cascaded) between any input and output is two in two-level logic. Because gates having high fanins and fanouts are slow, each instance was sensitized into a circuit consisting of 2-input AND, OR and inverter gates. In other words, each instance was transformed into a multi-level circuit. For example, the boolean algebra $F = (a \cdot b \cdot c) + (d \cdot e \cdot f)$ is transformed into $F = ((a \cdot b) \cdot c) + ((d \cdot e) \cdot f)$. When TEMPO was used to solve an instance of the two-level circuit, it is easy to find small unsatisfiable cores and to get block clauses with a few literals. The block clause with a few literals can make a stronger pruning. For a multi-level circuit, TEMPO ofter searches big unsatisfiable cores. Therefore our approach is comparable with CPLEX. As for instance of Two-level logic (hard), the scale of the problem still exceeds the powers of both approaches. In a word, our approach performs better in instances of the two-level circuit and comparably with CPLEX in instances of the multi-level circuit.

In table 3, we present the number of solved cases of both of two approaches. As we see, the number of solved instances of TEMPO is the same as that of CPLEX in multi-level logic and two-level logic (hard). However, TEMPO can solve more than two instances than CPLEX in two-level (all) circuits. TEMPO

Table 3: The number of solved cases.

	Multi-level	Two-level all	Two-level hard
total instances	219	134	22
CPLEX	**196**	122	**17**
TEMPO	**196**	**124**	**17**

can easily search the unsatisfiable core with a small size from the two-level logic circuit.

For the unsolved instances, both of two approaches can return the lower and the upper bounds of the optimal solution. Here, we use the relative gap to evaluate the solution. The $relative\ gap = \frac{upper\ bound - lower\ bound}{lower\ bound} \times 100\%$. If the relative gap is 0%, it means that the solver finds the optimum solution. In table 4, we present the relative gap of unsolved instances out of 3600 seconds. Our approach has better performance than CPLEX.

5 Related Works and Discussion

Here we recall previous works towards estimating maximum power of CMOS combinational circuit.

Devadas *et al.* [1992] converted a logic description into a mutltiple-output Boolean function of the input vector or vector sequence. It attempted to maximize the function by solving a weighted max-satisfiability problem using exact and approximate algorithms. However, the technique proved to be practically applicable only to small circuits.

Wang and Roy [1996] used an automatic test generation technique to obtain a lower bound of the maximum power consumption. Their approach generates the lower bound with the quality which cannot be achieved using simulation-based techniques.

Test generation-based approaches have also been reported [Li *et al.*, 2002]. Further, they formulate the sequential circuit maximum current problem as a combination automatic test pattern generation problem and solve it.

Sagahyroon and Aloul [2007] formulated maximum power estimation of CMOS combinational circuits as 0-1 ILP. They search for a vector pair $\{V_1, V_2\}$ that tends to maximize switching power rather than estimate maximum power directly. The circuit's logical behavior is represented by CNF constraints called C_1 after the application of input vector V_1. Similarly, CNF constraints C_2 represents the circuit after the application of input vector V_2. And then, CNF constraints C_3 represent XOR gates between the outputs of gates in C_1 and C_2. If XOR gate outputs logic 1, it means this gate occurs switching when the input vector V_1 is followed by input vector V_2. Otherwise, it means this gate does not occur switching. Therefore, the PB objective function is the weighted sum of XOR outputs. The weight is the fanout of the circuit gate corresponding. Thus, they use a ILP solver to maximize PB objective function, subjecting to $C_1 \cup C_2 \cup C_3$.

6 Conclusions

This paper has proposed a new approach for the maximum power estimation of CMOS combinational circuits. The maximum power estimation is modeled to ILP problem [Sagahyroon and Aloul, 2007]. Our approach – TEMPO extracts power constraints from unsatisfiable cores and adds them to ILP. The power constraint is a clause in which each variable is in the objective function $\mathcal{F}$. It can speed up the ILP solving. In TEMPO, we use Refine and Abstract processes. The Refine extracts a power constraint from an unsatisfiable core and the Abstract uses clause-selector variables to search a new unsatisfiable core. The experimental results show that our approach can improve the speed of the solution. Especially, our approach has a good performance in challenging benchmarks. As for Multi-level logic and Two-level logic (hard) problems, our approach is comparable with CPLEX. Because the circuit is two levels in Two-level logic, TEMPO can easily search lots of unsatisfiable cores with small size. Unsatisfiable cores with small size make a stronger pruning. In the future, we will combine our approach with MaxSAT techniques and extend this approach to sequential circuits.

Table 4: The relative gap of the unsolved instances. Solution time is 3600 seconds. "#V" is the number of variables in ILP and "#C" is the number of constraints in ILP. "-" means that TEMPO cannot find a feasible solution that subjects to $\mathcal{C}_{all}$.

	Name	#V	#C	CPLEX	TEMPO
Multi-level	C499	2176	6384	2.08%	**1.81**%
	C2670	4375	11852	1.06%	**0.88**%
	C3540	5359	16100	6.29%	**4.91**%
	pdc	7502	23162	12.03%	**6.79**%
	misex3	7825	23932	**14.06**%	14.62%
	spla	7862	24374	18.37%	**16.39**%
	pair	8167	23856	3.13%	**2.98**%
	bc0	8323	25240	12.46%	**9.22**%
	cps	8799	27174	16.17%	**9.52**%
	C5315	9212	27162	**7.37**%	8.43%
	table5	9247	28542	**13.46**%	24.39%
	C7552	11655	34124	17.58%	**13.67**%
	seq	12070	36790	25.31%	**24.35**%
	i10	12451	37182	12.77%	**10.57**%
	C6288	13942	41682	**21.31**%	25.28%
	ex1010	18248	55288	**55.30**%	57.36%
	apex4	18888	57218	40.34%	**40.14**%
	prom2	19104	57922	28.32%	**25.40**%
	des	19694	59398	**10.90**%	11.24%
	xparc	24028	74054	78.04%	**29.06**%
	mainpla	28389	86252	**34.49**%	72.69%
	prom1	42735	129518	**133.24**%	-
Two-level all	spla	8477	25982	**12.10**%	15.56%
	table5	9100	28108	24.79%	**23.07**%
	seq	12070	36790	**26.17**%	29.00%
	bca	13867	43046	22.91%	**13.79**%
	pdc	15767	48026	23.61%	**20.86**%
	ex1010	16235	49294	49.42%	**45.65**%
	apex4	17151	51956	22.41%	**20.31**%
	prom1	37863	114802	**84.04**%	126.67%
Two-level hard	ex1010	11663	35402	**22.87**%	30.50%
	xparc	21328	65868	79.89%	**77.14**%
	mainpla	23661	72170	**32.82**%	33.23%
	test3	36761	111618	**106.01**%	119.04%
	test2	85705	260464	415.44%	**232.81**%

References

[Aloul *et al.*, 2002] Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Generic ilp versus specialized 0-1 ilp: An update. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 450–457, 2002.

[Barth, 1995] Peter Barth. A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical report, Max-Planck-Institut für Informatik, 1995.

[Brayton and Mishchenko, 2010] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pages 24–40, 2010.

[Cplex, 2010] IBM ILOG Cplex. 12.2 user's manual. *ILOG. See ftp://ftp. software. ibm. com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex. pdf*, 2010.

[Devadas *et al.*, 1992] Srinivas Devadas, Kurt Keutzer, and Jacob White. Estimation of power dissipation in cmos combinational circuits using boolean function manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):373–383, 1992.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518, 2003.

[Halter and Najm, 1997] Jonathan P Halter and Farid N Najm. A gate-level leakage power reduction method for uultra-low-power cmos circuits. In *Proceedings of CICC 97-Custom Integrated Circuits Conference*, pages 475–478, 1997.

[Li *et al.*, 2002] Fei Li, Lei He, and Kewal K Saluja. Estimation of maximum power-up current. In *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*, pages 51–56, 2002.

[Liffiton and Sakallah, 2008] Mark H Liffiton and Karem A Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

[Manich and Figueras, 1997] Salvador Manich and Joan Figueras. Maximizing the weighted switching activity in combinational cmos circuits under the variable delay model. In *Proceedings of the 1997 European conference on Design and Test*, page 597, 1997.

[Pedram, 1996] Massoud Pedram. Power minimization in ic design: Principles and applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1(1):3–56, 1996.

[Sagahyroon and Aloul, 2007] Assim Sagahyroon and Fadi A Aloul. Using sat-based techniques in power estimation. *Microelectronics Journal*, 38(6-7):706–715, 2007.

[Wang and Roy, 1996] Chuan-Yu Wang and Kaushik Roy. Maximum power estimation for CMOS circuits using deterministic and statistical approaches. In *Proceedings of 9th International conference on VLSI design*, pages 364–369, 1996.

[Wu *et al.*, 2001] Qing Wu, Qinru Qiu, and Massoud Pedram. Estimation of peak power dissipation in vlsi circuits using the limiting distributions of extreme order statistics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(8):942–956, 2001.

[Yang, 1991] Saeyang Yang. *Logic Synthesis and Optimization Benchmarks User Guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.