

Exact Synthesis with Optimal Switching Activity

Marcel Walter^{*†}, Michael Feldmeier^{*}, and Robert Wille^{*†‡}

^{*}Chair for Design Automation, Technical University of Munich, Germany

[†]Munich Quantum Software Company GmbH, Garching near Munich, Germany

[‡]Software Competence Center Hagenberg GmbH, Austria

Email: {marcel.walter, mi.feldmeier, robert.wille}@tum.de

<https://www.cda.cit.tum.de/research/logicsynth/>

Abstract—Power consumption is a primary constraint in modern digital circuit design, with switching activity being a major contributor to dynamic power dissipation. While exact synthesis methods guarantee optimality for metrics such as gate count or delay, they typically do not directly target switching activity. This paper presents a novel SAT-based exact synthesis approach designed to minimize switching activity in combinational logic circuits. We extend existing SAT encodings for logic synthesis, incorporating new constraints and variables to model and constrain the switching behavior of the circuit. Different SAT encoding strategies, including BDD-based approaches for handling cardinality constraints, as well as various search algorithms, are explored. Experimental results on NPN benchmark functions demonstrate the effectiveness of the proposed method in identifying circuits with, on average, 6.7% (over 30% in the best case) reduced switching activity compared to traditional exact synthesis techniques, often achieving this reduction with no or minimal area overhead. While runtime remains challenging, this work establishes a foundation for power-aware exact synthesis.

Index Terms—Exact Synthesis, Logic Synthesis, Switching Activity, Power Minimization, Boolean Satisfiability (SAT), Low Power Design, EDA.

I. INTRODUCTION

The relentless scaling of semiconductor technology has made power dissipation a first-order design constraint, with the *power wall* presenting a formidable challenge [1]. In modern CMOS technologies, dynamic power, which stems from the charging and discharging of load capacitances during logic transitions, is a dominant contributor [2]. The frequency of these transitions, termed *switching activity*, is therefore a critical target for power reduction.

Logic synthesis aims to bridge the gap between a functional specification and an optimized gate-level netlist. While heuristic algorithms offer practical solutions balancing area, delay, and power, they provide no guarantee of optimality [3]–[6]. In contrast, *exact synthesis* methods leverage formal techniques like *Boolean Satisfiability* (SAT) to find implementations that are provably optimal for a given cost metric [7]–[10]. However, exact synthesis has traditionally focused on minimizing area or delay [8], [11], [12], largely overlooking that an area- or delay-optimal circuit may exhibit substantial unnecessary switching, leading to poor power efficiency. Unlike area or delay, which are direct structural properties, switching activity is a complex functional property of the network, making its integration into exact synthesis a non-trivial challenge.

This paper addresses this challenge by introducing a novel SAT-based exact synthesis framework, designed to minimize total switching activity as its primary objective. Our approach extends the established *Distinct Input Truth Tables* (DITT) encoding [8], [9] by incorporating bespoke variables and constraints that model and limit the cumulative switching activity of the network. This allows a SAT solver to systematically identify logic structures that implement the desired function with provably minimal switching activity.

As a foundational study addressing this gap, this work provides essential proof of concept that formally optimizing for switching activity is feasible. The primary contributions of this work are 1) the formulation of a SAT-based exact synthesis methodology explicitly

targeting switching activity minimization, including novel *Binary Decision Diagram* (BDD) [13] based encoding techniques; 2) a comparative analysis of three different SAT-based search strategies; and 3) comprehensive experimental validation demonstrating average switching activity reductions of 6.7% (over 30% in the best case) compared to ABC’s conventional area-focused exact synthesis `twoexact`, often with no or minimal area overhead.

The remainder of this paper outlines our approach and presents our findings. Section II provides essential background on SAT-based exact synthesis principles and the formal definition of switching activity. Section III elaborates on the proposed SAT encodings and search algorithms. Section IV presents and analyzes the experimental results of applying our method to benchmark functions. Finally, Section V summarizes the key contributions, discusses limitations, and outlines promising directions for future research.

II. PRELIMINARIES AND RELATED WORK

This section lays the groundwork for our proposed methodology by reviewing the principles of SAT-based exact synthesis and defining switching activity.

A. SAT-based Exact Synthesis

Exact synthesis aims to discover a logic network implementation of a given Boolean function f that is provably optimal with respect to a predefined cost metric, most commonly gate count (area) or logic depth (delay). Unlike heuristic methods, which employ greedy strategies and offer no guarantee of optimality, exact synthesis performs an exhaustive search, often leveraging the power of *Boolean Satisfiability* (SAT) solvers [7], [14]. Such methods are common and established in the design of conventional circuits and systems [8], [11], [12], but also in the design for other and emerging circuit technologies such as field-coupled nanotechnologies [15], [16], quantum computing [17], [18], reversible logic [19], [20], or microfluidics [21], [22].

The core idea is to translate the problem of finding a logic network into a SAT instance in *Conjunctive Normal Form* (CNF). This involves defining Boolean variables to represent structural and functional aspects of the network (e. g., gate functions, interconnections) and clauses to encode the rules of a valid circuit (e. g., functional correctness, structural consistency, acyclicity). A SAT solver then determines if a network with the specified properties exists. An iterative approach is typically employed to find the minimum-size network, where the SAT solver is queried with sequentially increasing gate counts r until a solution is found.

Several SAT encodings have been developed; this work builds upon the *Distinct Input Truth Tables* (DITT) encoding, which is used in ABC’s `twoexact` implementation [8], [23]. In DITT, variables encode the truth table of each gate and its connections to other gates or primary inputs. Associated clauses ensure functional consistency, correct signal propagation, and topological ordering. While powerful for area and delay optimization, standard SAT-based exact synthesis

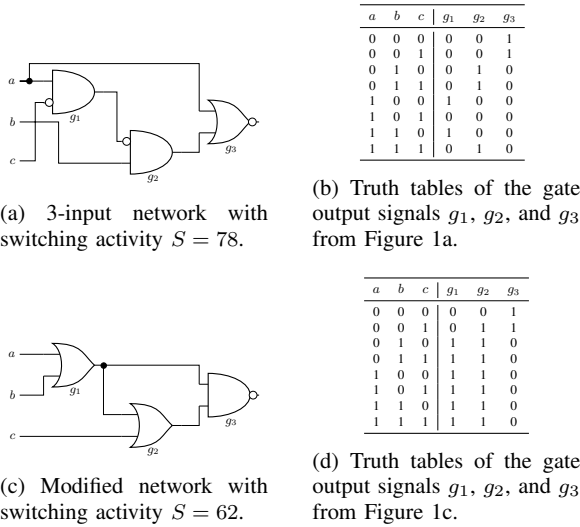


Figure 1: Logic networks and their corresponding truth tables showing different switching activities.

formulations, including DITT, do not inherently model or optimize for switching activity, representing the gap this work aims to fill.

B. Switching Activity

Switching activity is a primary determinant of dynamic power dissipation in CMOS circuits, representing the frequency of logic transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) at gate outputs [24]. For a network with k inputs, the activity S_i of a gate i depends on its truth table's balance. Let $p_i = \min(n_{i,0}, n_{i,1})$ be the count of the minority logic value in the gate's 2^k -entry truth table. The switching activity of gate i is:

$$S_i = 2p_i(2^k - p_i) \quad (1)$$

The total switching activity S for a network of r gates is the sum of the individual gate activities:

$$S = \sum_{i=1}^r S_i = \sum_{i=1}^r 2p_i(2^k - p_i) \quad (2)$$

Crucially, different logic networks implementing the same function can exhibit significantly different switching activities.

Example 1. The network in Figure 1a has gate minority counts of $p_1 = 2$, $p_2 = 3$, and $p_3 = 2$. Per Equation 2, the total activity is $S = 24 + 30 + 24 = 78$.

In contrast, the restructured network in Figure 1c has minority counts of $p_1 = 2$, $p_2 = 1$, and $p_3 = 2$, resulting in a total activity of $S = 24 + 14 + 24 = 62$, i. e., a 20.51% reduction. This demonstrates that restructuring logic can significantly alter the internal p_i values and thereby reduce the overall switching activity S .

While the switching activity model in Equation 2 is a simplification, it serves as an effective proxy for physical design outcomes. To validate this, we analyzed the relationship between this abstract switching activity metric and post-layout dynamic power dissipation for the EPFL benchmarks [25]. As shown in Figure 2, which was generated using the OpenROAD [26] flow with randomized input patterns, there is a strong correlation between the two. This result confirms that minimizing switching activity at the logic level is a fruitful approach for reducing post-layout dynamic power, motivating the methodology presented in this paper.

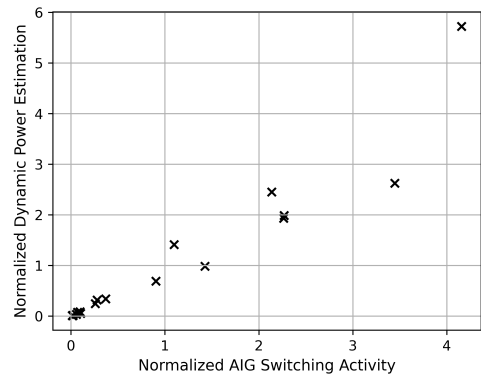


Figure 2: Correlation of AIG switching activity to post-layout dynamic power for EPFL benchmarks [25].

III. PROPOSED METHODOLOGY

Our methodology extends standard SAT-based exact synthesis techniques, specifically the DITT encoding, to explicitly target the minimization of switching activity. This requires introducing new variables to represent gate-level switching characteristics and formulating constraints that accurately capture and limit the total network switching activity. The core components of our approach are detailed below.

A. Bounds on Switching Activity

Since the network with minimal switching activity may be larger than the area-optimal one, the search space for our synthesis problem must be carefully bounded. The theoretical minimum switching activity for a single non-constant gate occurs when its function is highly unbalanced ($p_i = 1$), while the maximum occurs for a balanced function ($p_i = 2^{k-1}$). For a network of r gates, this yields theoretical lower and upper bounds on the total switching activity S :

$$S_{\min}(r) = 2r(2^k - 1) \quad (3)$$

$$S_{\max}(r) = r \cdot 2^{2k-1} \quad (4)$$

These theoretical bounds define a wide search space. However, it can be practically constrained. First, an area-optimal synthesis tool like ABC's `twoexact` provides an initial solution whose gate count and switching activity serve as effective upper bounds for our search, helping to establish a finite search space.

Furthermore, the lower bound $S_{\min}(r)$ can be tightened by considering structural constraints. The initial assumption that any gate can realize the minimum possible p_i value of 1 is often too optimistic. For example, in a network with $k = 4$ inputs, the first gate in the topology is driven only by primary inputs, which restricts its possible p_i values. This constraint propagates through the network, as subsequent gates are driven by a combination of primary inputs and other gates, similarly restricting their achievable p_i values. By accounting for these topological limitations, a more realistic and tighter lower bound on S can be calculated, which helps to prune the search space more effectively.

With the search space defined and constrained, we now describe the proposed SAT encoding for finding a network with optimal switching activity.

B. Encoding Switching Activity (p Variables)

To encode switching activity in SAT-based exact synthesis, we extend the DITT encoding by introducing new variables summarized in Table I and detailed in the following. As defined in Section II-B, the switching activity S_i of a gate i is determined by

Table I: Interpretation of encoding variable semantics.

SYMBOL	TYPE	DESCRIPTION
S	Integer	Global switching activity
S_i	Integer	Switching activity associated with gate i
S_a	Integer	Switching activity of gates with a minority entries
$n_{i,0}$	Integer	Count of zero outputs in the truth table for gate i
$n_{i,1}$	Integer	Count of one outputs in the truth table for gate i
p_i	Integer	Minority truth table count corresponding to gate i
$p_{i,a}$	Boolean	Gate i possesses a minority truth table count of a
s_a	Integer	Gates characterized by minority count a

$p_i = \min(n_{i,0}, n_{i,1})$. We introduce Boolean variables $p_{i,a}$ such that $p_{i,a} = 1$ if and only if gate i has a minority truth table count of a , where $1 \leq a \leq 2^{k-1}$.

To link these new $p_{i,a}$ variables to the gate's actual function (represented by its truth table variables $x_{i,t}$ in the DITT encoding), we require specific constraints. We explore two main approaches:

1) *Naive p Variable Encoding*: This method directly encodes the relationship using CNF clauses based on each gate's minterms. For each possible truth table, we determine its minority count a and generate clauses to enforce the mapping. For example, if a truth table assignment for gate i corresponds to $p_i = 2$, a clause is added to enforce that if those truth table variables are selected, then $p_{i,2}$ must be true. This is done for all minterms, encoding each possible value of a .

Additional constraints are needed to ensure that exactly one $p_{i,a}$ variable is selected for each gate i . This is achieved using standard *exactly-one* cardinality constraints [14], [27]:

$$\bigvee_{a=1}^{2^{k-1}} p_{i,a} \quad \forall i \quad (5)$$

$$\neg p_{i,a} \vee \neg p_{i,b} \quad \forall i, a \neq b \quad (6)$$

While conceptually simple, this naive encoding generates many clauses, potentially hindering SAT solver performance as k increases.

2) *BDD-based p Variable Encoding*: A more sophisticated approach utilizes *Binary Decision Diagrams* (BDDs) [13] to represent the function that maps the gate's truth table ($x_{i,t}$ variables) to the corresponding $p_{i,a}$ variable. To this end, a single BDD is constructed where the decision variables are the (ordered) truth table bits $x_{i,t}$, and the leaves correspond to the $p_{i,a}$ variables. The resulting structure is the (non-reduced) BDD shown in Figure 3.

This BDD is then reduced and translated into CNF. This involves introducing auxiliary Boolean variables: for each node m controlled by input variable c with low-child l and high-child h , clauses representing the multiplexer function $m \equiv (c \wedge h) \vee (\neg c \wedge l)$ are generated. For example, clauses like $(\neg c \vee \neg m \vee h)$ and $(c \vee \neg m \vee l)$ enforce the implication $m \implies ((c \wedge h) \vee (\neg c \wedge l))$, while clauses like $(\neg c \vee \neg h \vee m)$ and $(c \vee \neg l \vee m)$ enforce the reverse implication. This BDD-based encoding typically results in significantly fewer clauses than the naive method, though it introduces additional variables. The *exactly-one* constraint on $p_{i,a}$ (Equation 5 and 6) is still required.

C. Constraining Total Switching Activity (p Cardinality)

Having encoded p_i of each gate, we must constrain the network's total switching activity S , as given by Equation 2. This allows the SAT solver to search explicitly for implementations that adhere to a specific switching activity budget or minimize the value of S . Two primary strategies emerge for formulating these constraints within the SAT framework.

1) *Binomial Cardinality Encoding*: A straightforward approach involves directly constraining the number of gates of each specific value p_i within the network. Let s_a denote the desired count for

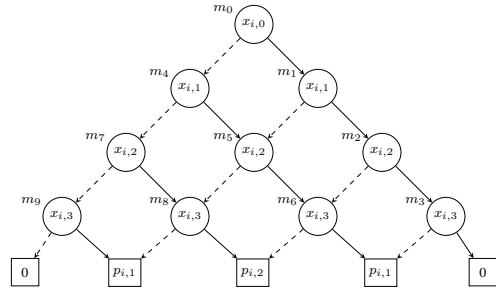


Figure 3: BDD template for p variable encoding.

gates exhibiting a minority truth table count of a . Using cardinality constraints, we can enforce this requirement for each possible value of a ($1 \leq a \leq 2^{k-1}$). Formally, for each a , we require:

$$\sum_{i=1}^r p_{i,a} = s_a \quad (7)$$

These equality constraints can again be translated into CNF using standard techniques for encoding *k-out-of-n* constraints. A common method involves combining *at-most-k* and *at-least-k* constraints [14], [27].

While conceptually simple, this approach necessitates an external search strategy. To minimize the total switching activity S , one must iterate through different combinations of target counts $(s_1, s_2, \dots, s_{2^{k-1}})$ that satisfy $\sum s_a = r$, calculate the corresponding target S for each combination using Equation 2, and query the SAT solver for each potentially improving combination. This requires pre-computing feasible combinations and managing the search process, noting that multiple distinct combinations of s_a values might yield the same total switching activity S . Corresponding search strategies are discussed in Section III-D.

2) *BDD-based Pseudo-Boolean Encoding*: An alternative, potentially more integrated approach leverages *Pseudo-Boolean* (PB) constraints to directly constrain the total switching activity S . Recalling Equation 2, the total switching activity can be interpreted as a weighted sum of the counts of each individual gate's switching activity:

$$S = \sum_{a=1}^{2^{k-1}} S_a \cdot \left(\sum_{i=1}^r p_{i,a} \right) = \sum_{a=1}^{2^{k-1}} \sum_{i=1}^r (2a(2^k - a)) \cdot p_{i,a} \quad (8)$$

where $S_a = 2a(2^k - a)$ is the switching activity contributed by a single gate ($p_i = a$).

This linear equation involves Boolean variables $p_{i,a}$ and integer coefficients S_a . Such PB constraints can be encoded into CNF. A powerful technique for this involves again constructing a BDD that represents the constraint $\sum_{i,a} (S_a \cdot p_{i,a}) = S_{\text{target}}$. The BDD nodes correspond to intermediate sums, with edges determined by the values of the $p_{i,a}$ variables. The leaves represent whether the constraint is satisfied (1-terminal) or violated (0-terminal).

This BDD-based PB encoding directly integrates the total switching activity constraint into the SAT formula. It allows for setting upper bounds ($S \leq S_{\text{target}}$) or exact values ($S = S_{\text{target}}$), facilitating search algorithms like iterative minimization or binary search on the target switching activity S . This approach avoids the explicit enumeration of s_a combinations required by the binomial cardinality constraint method.

D. Search Algorithms for Minimum Switching Activity

With the SAT encodings for switching activity (p variables) and the constraints on total activity (p constraints) established, we now

Algorithm 1: Queue Search

Input: Boolean function $f : \mathbb{B}^k \rightarrow \mathbb{B}$
Output: Network $N_{\text{switch}}^* \models f$ with minimal switching activity S

```
1  $N_{\text{area}}^* \leftarrow \text{twoexact}(f)$ 
2  $r \leftarrow N_{\text{area}}^* \cdot r$ 
3  $S_{\text{target}} \leftarrow 0$  // or  $S_{\text{min}}(k)$  (Equation 3)
4  $Q \leftarrow \text{allCombinations}(r)$  // sorted by  $S$ , then  $r$ 
5 while true do
6   if  $S_{\text{target}} \geq S_{\text{min}}(r+1)$  then
7      $r \leftarrow r+1$ 
8      $\text{comb} \leftarrow \text{allCombinations}(r)$  // all  $s_a$  comb. for
9       size  $r$ 
10    insert  $\text{comb}$  into  $Q$ 
11  end if
12  if  $Q \neq \emptyset$  and  $\text{head}(Q).S = S_{\text{target}}$  then
13     $C \leftarrow \text{pop}(Q)$  // fetch best candidate
14     $\phi \leftarrow \text{CNF}(C.S, C.r)$  // binomial constraints
15     $(\text{result}, \text{model}) \leftarrow \text{solveSAT}(\phi)$ 
16    if  $\text{result} = \text{SAT}$  then
17      // optimal solution found
18       $N_{\text{switch}}^* \leftarrow \text{constructNetwork}(\text{model})$ 
19      return  $N_{\text{switch}}^*$ 
20    end if
21  else
22     $S_{\text{target}} \leftarrow S_{\text{target}} + 1$ 
23  end if
24 end while
```

discuss algorithms to guide the SAT solver towards finding a network implementation that minimizes S . Since the network size r yielding the minimum S is unknown a priori, and may be larger than the minimum size required for functional implementation (area optimum), the search must explore different values of both r and potential S . In the following, we examine three distinct algorithmic approaches: *Queue Search*, *Free Search*, and *Binary Search*.

1) *Queue Search*: This algorithm, outlined in Algorithm 1, utilizes the binomial cardinality constraints introduced in Section III-C1. It employs a priority queue Q to manage potential solutions. Each element in the queue represents a specific combination of gate counts $(s_1, s_2, \dots, s_{2^k-1})$ for a certain network size r . The queue is ordered primarily by the resulting total switching activity S (calculated via Equation 2) and secondarily by r , prioritizing combinations yielding lower S and smaller r .

The search proceeds iteratively, incrementing a target switching activity level, S_{target} . In each step, it first checks if the theoretical minimum activity for a network of size $r+1$, $S_{\text{min}}(r+1)$ (Equation 3), is less than or equal to the current S_{target} . If this condition holds, it implies that larger networks might potentially achieve the target activity level with fewer gates of high switching activity. Therefore, r is incremented, and all valid s_a combinations (comb) for this new size r (satisfying $\sum s_a = r$) are generated and inserted into the priority queue Q .

The algorithm then inspects the candidate combination at the head of the priority queue ($\text{head}(Q)$). If the queue is not empty and the candidate's switching activity S matches the current S_{target} , that candidate C is extracted ($\text{pop}(Q)$). A CNF formula ϕ is generated, using the binomial cardinality constraints ($\sum_{i=1}^r p_{i,a} = s_a$) corresponding to the gate counts stored within candidate C . The SAT solver is invoked on ϕ . If a satisfying assignment ($\text{result} = \text{SAT}$) is found, the corresponding model represents a valid network N_{switch}^* implementing the target function f with the optimal switching activity. This network is constructed and returned, terminating the algorithm.

If the queue head's activity does not match S_{target} , or if the SAT call for the popped candidate returns UNSAT, the algorithm proceeds. If the queue check condition (Line 11) was *false*, S_{target} is incremented, and the loop continues. If the SAT call returned UNSAT,

Algorithm 2: Free Search

Input: Boolean function $f : \mathbb{B}^k \rightarrow \mathbb{B}$
Output: Network $N_{\text{switch}}^* \models f$ with minimal switching activity S

```
1  $N_{\text{area}}^* \leftarrow \text{twoexact}(f)$ 
2  $r \leftarrow N_{\text{area}}^* \cdot r$ 
3  $S_{\text{target}} \leftarrow 0$  // or  $S_{\text{min}}(k)$  (Equation 3)
4 while true do
5   while  $S_{\text{target}} \geq S_{\text{min}}(r+1)$  do
6      $r \leftarrow r+1$ 
7   end while
8   for  $i \leftarrow N_{\text{area}}^* \cdot r$  to  $r$  do
9      $\phi \leftarrow \text{CNF}(S_{\text{target}}, i)$  // BDD-PB constraints
10     $(\text{result}, \text{model}) \leftarrow \text{solveSAT}(\phi)$ 
11    if  $\text{result} = \text{SAT}$  then
12      // optimal solution found
13       $N_{\text{switch}}^* \leftarrow \text{constructNetwork}(\text{model})$ 
14      return  $N_{\text{switch}}^*$ 
15    end if
16  end for
17   $S_{\text{target}} \leftarrow S_{\text{target}} + 1$ 
18 end while
```

the loop naturally continues, effectively discarding the unsatisfiable candidate combination C .

This method guarantees the finding of the optimal solution (the minimum S for the minimum r) because it explores candidates in increasing order of S and r . However, it requires explicitly generating and storing all s_a combinations in the priority queue and, by design, relies on the less effective binomial cardinality encoding.

2) *Free Search*: Leveraging the BDD-based PB encoding for total switching activity (introduced in Section III-C2) allows for the search strategy outlined in Algorithm 2. This approach avoids the explicit management of s_a combinations required by Queue Search. Since the BDD-PB encoding directly constrains the total switching activity S , all combinations of p_a gate counts yielding the target S are implicitly handled, simplifying the overall search logic.

This algorithm also iterates on the target activity S_{target} and maintains the maximum network size r explored so far. It first ensures that r is large enough to potentially achieve the current S_{target} , checking against the theoretical minimum $S_{\text{min}}(r+1)$ (Equation 3) and incrementing r if necessary. Then, for the current S_{target} , it iterates through all possible network sizes i up to the current maximum r . For each size i , it generates a CNF formula ϕ encoding the circuit constraints along with the PB constraint enforcing $S = S_{\text{target}}$.

The SAT solver is invoked on ϕ . If a solution is found ($\text{result} = \text{SAT}$) for any size i , this represents the optimal network N_{switch}^* : it achieves the lowest possible S_{target} , for the smallest network size i tested within that target level. The network is constructed from the model and returned, thereby terminating the algorithm. If no solution is found for any size i up to r for the current S_{target} , its value is incremented, and the process repeats for the next activity level. This algorithm delegates the complexity of handling s_a combinations to the PB-BDD encoding, hence to the SAT solver.

3) *Binary Search*: The Binary Search algorithm, outlined in Algorithm 3, checks intervals of switching activity values to further accelerate the discovery process compared to the linear Free Search. This approach also leverages the BDD-PB encoding to represent range constraints, such as $S \in [S_{\text{lower}}, S_{\text{upper}}]$ (Section III-C2). The algorithm takes a step size ΔS as an additional input.

The main loop first ensures r is sufficiently large for the current S_{lower} by checking against $S_{\text{min}}(r+1)$ (Equation 3) and incrementing r as needed. It then defines the current search interval $[S_{\text{lower}}, S_{\text{upper}}]$, where $S_{\text{upper}} = S_{\text{lower}} + \Delta S - 1$.

The algorithm iterates through possible network sizes i up to the current maximum r . For each size i , it generates a CNF for-

Algorithm 3: Binary Search

Input: Boolean function $f : \mathbb{B}^k \rightarrow \mathbb{B}$
Input: Step size ΔS
Output: Network $N_{\text{switch}}^* \models f$ with minimal switching activity S

```

1  $N_{\text{area}}^* \leftarrow \text{twoexact}(f)$ 
2  $r \leftarrow N_{\text{area}}^* \cdot r$ 
3  $S_{\text{lower}} \leftarrow 0$  // or  $S_{\text{min}}(k)$  (Equation 3)
4 while true do
5   repeat
6      $r \leftarrow r + 1$ 
7   until  $S_{\text{lower}} < S_{\text{min}}(r + 1)$ 
8    $S_{\text{upper}} \leftarrow S_{\text{lower}} + \Delta S - 1$ 
9   for  $i \leftarrow N_{\text{area}}^* \cdot r$  to  $r$  do
10     $\phi \leftarrow \text{CNF}(S_{\text{lower}}, S_{\text{upper}}, i)$  // BDD-PB constraints
11     $S \in [S_{\text{lower}}, S_{\text{upper}}]$ 
12     $(\text{result}, \text{model}) \leftarrow \text{solveSAT}(\phi)$ 
13    if  $\text{result} = \text{SAT}$  then
14      // optimal solution lies in interval
15       $\text{model} \leftarrow \text{binarySATSearch}(S_{\text{lower}}, S_{\text{upper}}, i, r)$ 
16       $N_{\text{switch}}^* \leftarrow \text{constructNetwork}(\text{model})$ 
17      return  $N_{\text{switch}}^*$ 
18    else
19       $S_{\text{lower}} \leftarrow S_{\text{lower}} + \Delta S$ 
20    end if
21  end for
22 end while

```

mula ϕ using the BDD-PB range encoding to check if *any* solution exists with switching activity S within the current interval. If the SAT solver finds a solution ($\text{result} = \text{SAT}$), it confirms that the optimal switching activity must lie within $[S_{\text{lower}}, S_{\text{upper}}]$. At this point, *binarySATSearch* is invoked to efficiently pinpoint the exact minimum value of S within the determined interval $[S_{\text{lower}}, S_{\text{upper}}]$ using binary search. The procedure involves further SAT calls. It returns the final model corresponding to the true minimum S . The network N_{switch}^* is then constructed and returned.

If the inner for-loop completes without finding a satisfiable assignment for any network size i within the current interval $[S_{\text{lower}}, S_{\text{upper}}]$, it means no solution exists in this range. The algorithm then advances to the next interval by incrementing S_{lower} by the step size ΔS and continues the search.

This approach can potentially converge faster than linear search methods, particularly when the optimal switching activity S is large, by eliminating large ranges of S values with each interval check and employing efficient binary search for refinement.

IV. EXPERIMENTAL EVALUATION

We evaluated the proposed methodology, named *pexact*,¹ by comparing it against the area-focused *twoexact* algorithm on all 222 canonized 4-input NPN class representatives [28]. We implemented *pexact* in C on top of ABC and measured runtimes on an AMD Ryzen 9 5900X CPU with 32 GB of RAM.

A. Search Algorithm Comparison

The runtime required to find the minimum-switching-activity network correlates strongly with the resulting minimum switching activity S , as illustrated in Figure 4a. Runtime generally increases with S , with a noticeable threshold around $S \approx 500$, beyond which runtimes escalate.

The Free Search algorithm was consistently the slowest. As shown in Figure 4b, both Queue Search and Binary Search offered comparable and substantial speedups, reducing runtime by approximately 60% compared to Free Search. No single algorithm proved universally superior; however, we note that Binary Search often quickly

¹Publicly available on GitHub: <https://github.com/cda-tum/ext-pexact/>

finds an interval containing the optimum but then spends considerable time pinpointing the exact value.

B. Impact of Encoding Choices and Bounds

Employing structurally derived, tightened $S_{\text{min}}(r)$ bounds generally improves runtime by reducing the search space, most consistently for Free Search (over 20% improvement). As anticipated, the BDD-based encoding for the $p_{i,a}$ variables vastly outperformed the naive approach. As shown in Figure 5, the BDD method yields significantly faster solving times.

C. Comparison Against Area-driven Synthesis

We compared our best-performing configuration of *pexact* (using Binary Search, S_{min} estimation, and BDD-based encoding) against *twoexact*.² Of the 222 NPN classes, *pexact* synthesized 211 within a 3-day timeout.³ The results in Figure 6 show that *pexact* reduced switching activity by 6.7% on average (from 443.67 to 413.97), with a best-case reduction of 30.9% in case of the function 0x189 (detailed in the following Section IV-D).

This reduction was achieved with minimal area overhead: 153 of the 211 synthesized classes ($\approx 72\%$) required no extra gates compared to *twoexact*. Of the remainder, 43 classes used one extra gate, 13 used two, and only 2 used three. The 11 unsynthesized classes likely represent functions that are difficult to synthesize exactly or have very high minimum switching activity.

D. Circuit-level Analysis of a High-Gain Benchmark

The function 0x189 represents a high-gain benchmark for the proposed methodology as it achieves a 30.9% reduction in switching activity compared to *twoexact*. Figure 7 visualizes the resulting exact networks obtained by the two algorithms, respectively. For the *twoexact* approach, the extracted parameters are $p_1 = 4$, $p_2 = 6$, $p_3 = 4$, and $p_4 = 8$, culminating in a switching activity of $S = 440$. In contrast, the proposed *pexact* methodology yields parameters of $p_1 = 4$, $p_2 = 4$, $p_3 = 2$, and $p_4 = 2$, resulting in a switching activity of $S = 304$. It is noteworthy that in both instances, the variable p_5 is excluded from the calculations, as it contributes equally to the switching activity of both circuit configurations and is predefined by the truth table 0x189.

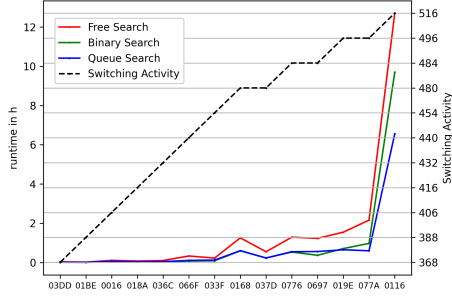
E. Discussion

The experimental results validate the proposed SAT-based methodology for minimizing switching activity. Our approach operates on a model that assumes uniform input probabilities. While this is a common simplification, our analysis in Figure 2 confirms this abstract metric serves as a strong proxy for post-layout dynamic power, making it a valuable target for optimization at the logic level.

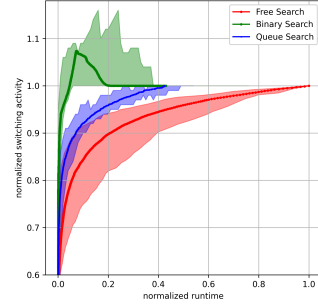
BDD-based encodings are essential for practical performance. While Free Search is conceptually simpler, Queue Search and Binary Search provide significant runtime advantages. The benchmarks demonstrate meaningful switching activity reductions achievable with minimal, often zero, area overhead compared to *twoexact*. Runtime remains the primary limitation, especially for high-activity functions, which confine practical use to smaller inputs (such as the evaluated $k = 4$) without extensive computational resources. The fact that 11 of the 222 NPN classes timed out underscores this challenge and suggests that these functions have particularly large and complex solution spaces. Nonetheless, this work confirms the feasibility and value of targeting switching activity directly within exact synthesis.

²Since we propose an *exact* synthesis scheme, comparisons against heuristics are out of the scope of this work.

³The 11 classes that timed out were 0x179A, 0x177E, 0x16AD, 0x16AC, 0x169B, 0x1698, 0x1697, 0x168E, 0x168B, 0x1689, and 0x1681, which correspond to functions with either complex structures or inherently high minimum switching activity.



(a) Minimum switching activity as a function of runtime; functions on the x-axis are ordered by S .



(b) Normalized switching activity as a function of normalized runtime.

Figure 4: Runtime comparison of the three proposed search algorithms (Section III-D) for 4-input NPN functions. Binary Search has been parameterized with $\Delta S = 75$.

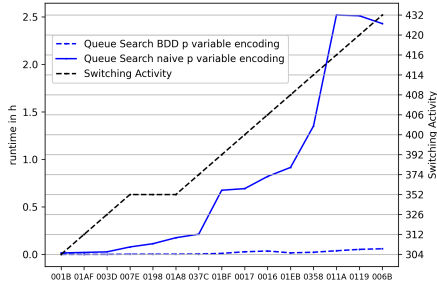
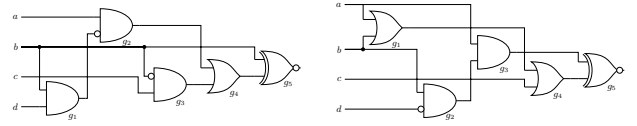


Figure 5: Runtime comparison of the p variable encoding strategy on Queue Search.



(a) The `twoexact` result with switching activity $S = 440$. (b) The `pexact` result with switching activity $S = 304$.

Figure 7: Comparison of the exact networks yielded by `twoexact` and `pexact` for the truth table $0x189$, respectively.

V. CONCLUSION AND FUTURE WORK

This paper introduced a SAT-based exact synthesis framework called `pexact` targeting the switching activity of logic networks. By extending the DITT encoding with novel constraints to model and limit gate switching, we developed three search algorithms that find networks that are provably optimal for this metric.

Experimental results demonstrated the approaches' effectiveness, achieving an average switching activity reduction of 6.7% on 4-input NPN class benchmarks compared to ABC's `twoexact` method, with best-case gains up to 30.9%. Notably, these power savings were often achieved with no area penalty (153 of 211 cases), highlighting the potential for direct, power-efficient replacement of existing circuits.

While the proposed methods offer a promising path towards power-aware exact synthesis, runtime remains a significant challenge. Future work will prioritize improving scalability and practical relevance in several key areas. Methodologically, we plan to explore hybrid exact/heuristic algorithms to better manage the search space. Furthermore, the underlying switching activity model can be extended to incorporate signal probabilities, enabling the synthesis of circuits optimized for specific, non-random input distributions commonly encountered in real-world applications.

A key use case of exact synthesis is generating cell libraries for logic rewriting. While this work focuses on the synthesis of individual functions, a critical next step is to evaluate the impact of a `pexact`-generated library on network-level optimization. We hypothesize that local substitutions with power-optimized cells will yield global power reductions, but this remains an important avenue for future validation.

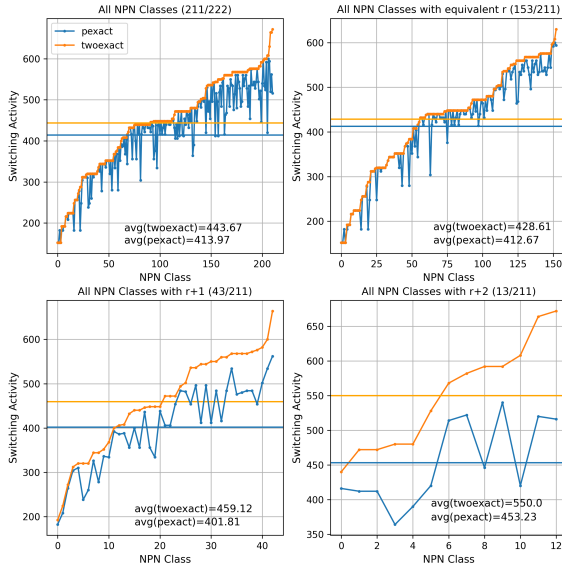


Figure 6: Quality of result comparison of `pexact` against `twoexact` shows an average reduction of S by 6.7% in our favor.

REFERENCES

- [1] X. Xie, “Low-power technologies in high-performance computer: trends and perspectives,” *National Science Review*, vol. 3, no. 1, pp. 23–25, 2016.
- [2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-Power CMOS Digital Design,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.
- [3] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Muroga, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, “SIS: A System for Sequential Circuit Synthesis,” in *Design Automation Conference (DAC)*, 1992, pp. 443–448, describes the influential SIS system, which implemented many heuristic optimization algorithms.
- [4] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. M. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
- [5] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.
- [6] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.
- [7] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*, ser. The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 2015, vol. 4A.
- [8] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, “SAT-based Exact Synthesis: Encodings, Topology Families, and Parallelism,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 871–884, 2020.
- [9] M. Soeken, W. Haaswijk, E. Testa, A. Mishchenko, L. G. Amarù, R. K. Brayton, and G. De Micheli, “Practical Exact Synthesis,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 309–314.
- [10] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, “SAT based Exact Synthesis using DAG Topology Families,” in *Design Automation Conference (DAC)*, 2018, pp. 1–6.
- [11] L. G. Amarù, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P.-E. Gaillardon, J. Olson, R. K. Brayton, and G. De Micheli, “Enabling Exact Delay Synthesis,” in *International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, 2017, pp. 352–359.
- [12] M. Soeken, G. De Micheli, and A. Mishchenko, “Busy Man’s Synthesis: Combinational Delay Optimization with SAT,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 830–835.
- [13] R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
- [14] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*. IOS Press, 2009.
- [15] M. Walter, R. Wille, D. Große, F. Sill Torres, and R. Drechsler, “An Exact Method for Design Exploration of Quantum-dot Cellular Automata,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 503–508.
- [16] M. Walter, W. Haaswijk, R. Wille, F. Sill Torres, and R. Drechsler, “One-pass Synthesis for Field-coupled Nanocomputing Technologies,” in *ASP-DAC*. ACM New York, NY, USA, 2021, pp. 574–580.
- [17] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, “Using Boolean Satisfiability for Exact Shuttling in Trapped-Ion Quantum Computers,” in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 127–133.
- [18] R. Wille, A. Lye, and R. Drechsler, “Exact Reordering of Circuit Lines for Nearest Neighbor Quantum Architectures,” *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 12, pp. 1818–1831, 2014.
- [19] A. Zulehner and R. Wille, “Taking One-to-one Mappings for Granted: Advanced Logic Design of Encoder Circuits,” in *Design, Automation and Test in Europe (DATE)*, 2017, p. 818–823.
- [20] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, “Exact Multiple-Control Toffoli Network Synthesis With SAT Techniques,” *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 5, pp. 703–715, 2009.
- [21] P. Ebner, M. Emmerich, E. R. Safai, A. Paul, M. Odijk, J. Loessberg-Zahl, and R. Wille, “Automatic Design for Modular Microfluidic Routing Blocks,” *International Conference On Computer Aided Design (ICCAD)*, pp. 1–7, 2025.
- [22] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille, “Close-to-Optimal Placement and Routing for Continuous-Flow Microfluidic Biochips,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE Press, 2017, p. 530–535.
- [23] R. Brayton and A. Mishchenko, “ABC: An Academic Industrial-Strength Verification Tool,” in *International Conference on Computer-Aided Verification (CAV)*, 2010, pp. 24–40, available: <https://github.com/berkeley-abc/abc>.
- [24] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White, “Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 1, pp. 121–127, 1997.
- [25] L. Amarù, P.-E. Gaillardon, and G. De Micheli, “The EPFL Combinational Benchmark Suite,” in *International Workshop on Logic and Synthesis (IWLS)*, 2015.
- [26] T. Ajayi, D. Blaauw, V. Zolotov, D. Papa *et al.*, “OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain,” in *Government Microcircuit Applications and Critical Technology Conference*, 2019, pp. 1105–1110.
- [27] C. Sinz, “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints,” in *International Conference on Principles and Practice of Constraint Programming (CP)*, ser. Lecture Notes in Computer Science, vol. 3709. Springer, 2005, pp. 827–831.
- [28] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, “Fast Boolean Matching based on NPN Classification,” in *International Conference on Field-Programmable Technology (FPT)*. Kyoto, Japan: IEEE, 2013, pp. 310–313.