PONO: Power Optimization with Near Optimal SMT-based Sub-circuit Generation

Sunan Zou and Guojie Luo
School of Computer Science, Peking University, Beijing, China
Center for Energy-efficient Computing and Applications, Peking University, Beijing, China
{zousunan,gluo}@pku.edu.cn

ABSTRACT

Generating high-quality sub-circuits for local substitution is an effective optimization technique in logic synthesis. There have been abundant works on generating area- and delay-optimal sub-circuits, greatly enhancing the logic optimization quality. However, power-oriented sub-circuit generation is rarely discussed, while optimizing power consumption in this sub-15 nm era is of paramount interest. We propose PONO, an SMT-based near optimal sub-circuit generation flow for power optimization. PONO enables power-oriented circuit library building and fills the gap in generating circuits near the Pareto frontier in PPA (Power, Performance, and Area). It manifests superiority in power reduction over traditional one in rewrite, a key logic optimization algorithm. We test PONO on EFPL benchmarks, and it shows 8.7% less power consumption without degrading the post-place-and-route performance and area.

CCS CONCEPTS

• **Hardware → Circuit optimization**; *Circuits power issues.*

KEYWORDS

logic synthesis, SMT problem, synthesis for low power

ACM Reference Format:
Sunan Zou and Guojie Luo. 2024. PONO: Power Optimization with Near Optimal SMT-based Sub-circuit Generation . In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655904

1 INTRODUCTION

Reducing power consumption is vital in Electronic Design Automation (EDA). Power-oriented logic synthesis has become prevalent in the early stage of logic optimization [21]. As a critical technique in logic synthesis, logic rewrite [13] substitutes sub-circuits with high-quality pre-computed ones iteratively to optimize the circuit quality. The pre-computed circuits have a vital influence on the rewrite quality, and various works have focused on generating area and delay optimal sub-circuits. However, power-oriented sub-circuit generation is still missing. In this paper, we apply SMT formulas to encode power metrics to generate power-oriented sub-circuits. We can optimize circuit power consumption during logic optimization by building a power-oriented, nearly optimal circuit library.

Previous works leverage circuit enumeration [12, 16], function decomposition [8, 11] to find high-quality circuits with different optimization targets and constraints. Recent progress in SAT/SMT solvers enables exact synthesis to find optimal circuits. [6]. This technique translates the synthesis problem into a SAT/SMT formula with different optimization goals like area [17], delay [2], and fanout bound [9]. However, this promising technique fails to consider power due to the challenges below: **Difficulty in selecting power metrics:** Power consumption relates to variables like switching probability, leakage, and capacitance. Designing and modeling them in the synthesis stage is hard. Furthermore, propagating these variables through proper formulas remains a problem. **Infinite search space:** Power consumption is related to input signal probability patterns or workload vectors, leading to infinite search space. The power-oriented circuit library needs to store $|F| \times |I_p|$ logic circuits, an infeasible size, where $|F|$ and $|I_p|$ are the number of functions and input patterns, respectively.

To tackle the challenges above, we select effective metrics to evaluate local dynamic power consumption and design clauses to propagate variables across the circuit. By designing a stratification technique for signal probability, search space can be reduced to a finite one, and we prove that a tolerable value bounds the induced error. The proposed equioptimizable input patterns further shrink the circuit library size. Combining these techniques, we propose PONO to enhance logic synthesis by generating power-oriented sub-circuits. PONO includes SMT formulas to encode power consumption and apply a similar flow of SAT-based exact synthesis to generate sub-circuits. Since obtaining accurate power estimation during synthesis is impossible, the generated circuits cannot guarantee optimality. However, PONO gradually releases the total power consumption constraint to generate nearly optimal sub-circuits. To summarize, our contributions are as follows:

- To the best of our knowledge, we are the first to apply SMT to generate high-quality sub-circuits with power consideration.
- We generate a power-oriented library and fill the gap of generating circuits near the Pareto frontier in PPA.
- PONO reduces 8.7% power consumption using LUT-based rewrite compared to the state-of-the-art method.

We evaluate PONO on EPFL arithmetic and random control benchmarks. The generated circuits push forward the Pareto front in PPA, and the library building process finishes within seven days. By applying the power-oriented library in rewrite transformation, we achieve 8.7% more power reduction than a state-of-the-art method with comparable area and latency performance.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655904

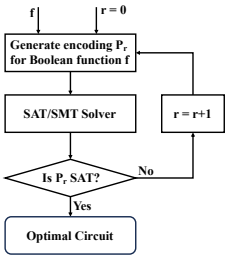

Figure 1: Exact synthesis flow

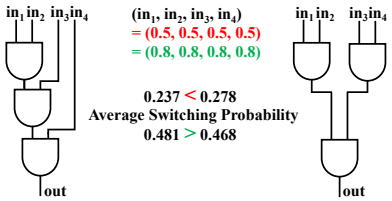

Figure 2: Average switching probability of logically equivalent circuits with different topology.

2 BACKGROUND

2.1 SAT/SMT-Based Exact Synthesis

Exact synthesis can find optimal circuits implementing a Boolean function with certain resource constraints. Figure 1 shows the general flow of SAT/SMT-based exact synthesis [7]. First, it initializes resource constraint r like the number of gates, depth, and power-related resources to a lower bound. Then, it encodes the given Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m = (f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$ and constraint into a SAT/SMT problem P_r and send it to a solver. Resource constraint r increments by one unit every loop until the formulated problem P_r can be solved. Finally, a corresponding optimal circuit represented by the Boolean chain is gained by decoding P_r. Boolean chain for f is a sequence of connected gates $(x_{n+1}, ..., x_{n+r})$. Each x_i can only refer to previous steps in the sequence as $x_i = x_{j(i)} o_i x_{k(i)}$ for all $n + 1 \leq i \leq n + r$, where $1 \leq j(i) < k(i) < i$ and operator o_i can implement all sixteen 2-input Boolean functions. Each f_i can be represented by either $x_{l(i)}$ or $\bar{x}_{l(i)}$ where $0 \leq l(i) \leq n + r$ and $x_0 = 0$. To reduce search space, we constrain f_i to normal function ($f_i(0, ..., 0) = 0$) without loss of generality [7].

There are multiple encoding ways like Single Selection Variables (SSV), Multiple Selection Variables (MSV), and Distinct Input Truth Tables (DITT) [6]. This paper extends SSV to power optimization, and we introduce its formulation below. Encoding P_r contains the following variables for $n < i \leq n + r$, $1 \leq h \leq m$, and $0 < t < 2^n$:

$$x_{it} : the\ t^{th}\ bit\ of\ x_i's\ truth\ table.$$
$$g_{hi} : f_h(x_1, ..., x_n) = x_i.$$
$$s_{ijk} : x_i = x_j\ o_i\ x_k,\ 1 \leq j < k < i.$$
$$f_{iuv} : u\ o_i\ v,\ 0 \leq u,\ v \leq 1,\ u + v > 0.$$

A set of clauses can constrain these variables to generate Boolean chains implementing all given functions. First, we add the main clauses $\bar{s}_{ijk} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c) \vee (f_{ibc} \oplus \bar{a}))$. It means step x_i has inputs x_j and x_k implement the function $f(b, c) = a$ on t^{th} bit of x_i, where a, b, c are constants. Then we add clauses $\bar{g}_{hi} \vee \bar{x}_{it}$ or $\bar{g}_{hi} \vee x_{it}$ based on the truth table of f_h. Besides, clauses $\bigvee_{i=n+1}^{i=n+r} g_{hi}$ make every output h points to a certain step x_i and clauses $\bigvee_{1 \leq j < k < i} s_{ijk}$ constrain that every step x_i has two inputs. We extend SSV formulas to an SMT one by introducing power-related variables and clauses. This modification allows us to generate power-oriented netlists that are nearly optimal.

2.2 Logic Rewrite

High-quality pre-computed circuits can help optimize the netlist by a logic rewrite transformation. A netlist is a circuit representation consisting of logic nodes and edges connecting them. A node's cut is a set of nodes that can block all of the paths from the primary inputs to it. Rewrite tries replacing cuts with pre-computed equivalent sub-circuits for each node iteratively [13]. Such local optimization can accumulate and result in final quality improvement. Rewrite transforms a netlist into a functionally equivalent one with a better PPA. The effect of the rewrite lies in the cut selection strategy and subgraph quality. We focus on improving the latter one in this paper. PONO can generate power-oriented sub-circuits and thus enhance rewrite in power consumption. For simplicity, in this paper, we apply LUT mapping [15] to generate cuts and replace them with pre-computed ones.

2.3 Power Optimization

Power consumption in CMOS circuits mainly comprises dynamic and static ones. Dynamic power (P_D) comes from load capacitance charge and discharge, which can be calculated through

$$P_D = \frac{1}{2} \times \sum_{n \in nodes} V^2 \times C_n \times D_n \tag{1}$$

V is the working voltage, D_n is the transition density (average number of logic transitions per second), and C_n is the load capacitance. Since load capacitance is unavailable during logic synthesis, transition density is the typical power metric in this stage [19]. Static power (P_S) is the summation of the leakage power of all standard cells in the network. Optimizing these variables can help reduce power consumption. We choose a representative variable D_n to represent circuit power consumption and help generate power-oriented near optimal circuits.

3 OBSERVATION AND MOTIVATION

Previous sub-circuit generation works using exact synthesis have offered new opportunities for the area and delay optimization in logic synthesis [2, 17]. The optimal circuit libraries they built outperformed ABC [14]. However, these optimal libraries have not considered power reduction. It is intuitive to generate a power-oriented sub-circuit for logic optimization.

Though topology information is used in the original exact synthesis to accelerate the flow [4, 5], the internal structure of generated circuits is not constrained. Thus, the flow can generate circuits of the same size or depth but with different topologies for the same function. Figure 2 illustrates the logically equivalent 4-input *AND* gate implementations. The internal structure influences the switching power, and different input signal probability prefers different structures. For example, the left circuit in Figure 2 has a lower average and total switching probability when the input probability

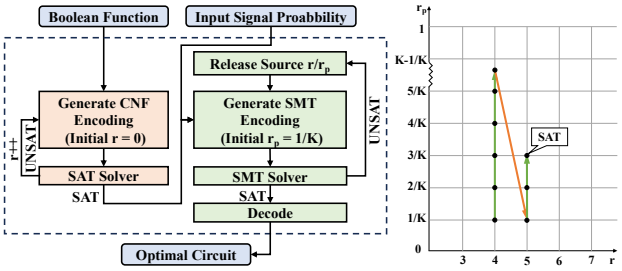

Figure 3: (a) Sub-circuit generation flow of PONO; (b) Illustration of increasing r_p and r in turn.

is $(0.5, 0.5, 0.5, 0.5)$. In contrast, the right circuit performs better in power when the input probability is $(0.8, 0.8, 0.8, 0.8)$. This highlights the need to consider power during sub-circuit generation.

Furthermore, generating sub-circuits to optimize power in a local scenario is promising. Optimizing local size reduces global area [17], and optimizing depth can ensure no worse global results [2]. Optimizing dynamic power locally applies the same. Reducing the switching probability inside a sub-circuit by constraining the internal structure does not affect the output switching probability. Local optimization has no risk of global quality degradation. This technique can combine with power-oriented technology mapping to hide signals with high switching probability. Its potential application in flow-level power optimization [21] and power-oriented standard cell library construction [18] are also promising.

4 POWER-ORIENTED SUB-CIRCUIT GENERATION AND APPLICATIONS

4.1 Metric Selection and Overview

The first step in generating near-optimal sub-circuits considering power is to model the power consumption. The number of gates can model static power in logic synthesis as technology is not induced in this stage. Thus, we do not induce new variables or clauses and use r to represent them. For dynamic power, we need to select variables in Equation (1) to model the power consumption of the generated circuit. They must be representative and feasible to propagate among used gates during encoding. We first rule out working voltage and frequency as we assume they are constant and remain the same as device specifications for all sub-circuits.

We also discard load capacitance C_n, which is hard to approximate during logic synthesis since necessary technology information is missing. Indeed, this information becomes available only during physical design. Furthermore, load capacitance does not impact local power consumption significantly inside a sub-circuit. This is because the fanouts of a node decide its load capacitance, and the number of fanouts is generally small in a sub-circuit. There are only dozens of gates in a sub-circuit, which induces small load capacitance compared to counterparts between sub-circuits. Therefore, we use D_n to model the local power behavior of a sub-circuit. Transition density and switching probability have a linear relationship in a rough model. Timing and glitches may cause inaccuracy of this model, but it succeeds in working as a high-level heuristic, as validated in Section 5. We can calculate the switching probability from the input signal probability of being 1 and the nodes' function. We extend SSV encoding by adding the variable p_i to represent x_i's signal probability of being 1 (shortened to signal probability in the rest of this paper). PONO also constrains a new type of resource: average switching probability r_p to signify the power consumption of a circuit. By inducing clauses related to these variables, our extension can model power consumption, and we formulate this extension as an SMT problem.

Figure 3a elaborates on the framework of PONO. It takes targeted Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ and input signal probability $P = (p_1, p_2, ..., p_n)$ as inputs. Then, an exact synthesis like trial-and-error flow starts, containing two stages: find a feasible solution and a solution with estimated "minimal" power consumption. The first stage (in red) does not consider power like the original exact synthesis and uses SAT solver ParKissat [20] for faster solving. After obtaining a feasible solution, the second stage (in green) tries to optimize power by constraining the number of gates r and power resource r_p, starting another round of trial-and-error flow. We use SMT solver Z3 (NRA logic) [3] to solve formulated encoding. As r and r_p constrain the power consumption by $r \times r_p$, we may release the r constraint for significant r_p reduction, enlarging the power optimization search space. Therefore, r and r_p may increment alternatingly, as in Figure 3b, where the green line denotes r_p increment, and the red line denotes r increment.

For power-oriented library construction, we discretize r_p in real to reduce infinite search space into finite one. This approximation can be reduced to the original problem when the number of discretized classes K is large enough. Since the size of the library will be $|F| \times K^n$, where $|F|$ is the number of functions in a library. Such exponential increase speed makes it infeasible to store such a library as K increases. Therefore, we propose equioptimzable input signal probability patterns to shrink the library size.

4.2 Encoding Method

We encode power-oriented near optimal sub-circuit generation into an SMT problem based on SSV encoding. SMT consists of two parts: SSV transformed one and power-related one. The former directly transforms the original SSV encoding in SAT formulas into SMT formulas. The latter adopts a new set of variables to represent the power-related resource:

$$p_i : signal\ probability\ of\ x_i\ being\ 1,\ 1 \leq i \leq n + r.$$
$$r_p : average\ switching\ probability.$$

We propagate p_i through the whole circuit and calculate the average switching probability via $r_p = \frac{\sum_{n+1}^{n+r} 2p_i \times (1-p_i)}{r}$ to serve as a constrained resource for power consumption.

Both signal probability p_i and average switching capacitance r_p are real numbers in $[0, 1]$, causing infinite search space and an intractable library. Therefore, we leverage a stratification technique to reduce search space from infinite to finite. This technique shrinks the switching and input probability to a finite number of classes and makes it possible to generate power-oriented circuits through SMT formulas. We create K classes to separate interval $[0, 1]$ into K intervals $[0, 1/K), [1/K, 2/K), ..., [K - 1/K, 1]$. We classify the input signal probability into k^{th} subinterval into the k^{th} class. During PONO flow, we gradually loose r_p constraints in units

of classes. By constraining the r_p of each step to a specified value, we can generate sub-circuits for power-oriented logic optimization. This stratification technique, as a simplified model, induces error. However, we can reduce this model to the original problem with a sufficiently large number of classes. The induced error for switching probability and input signal probability are bounded by $\frac{CD}{K}$ when considering a simplified independent input scenario and constraining $K > D + 2D^2$. C is a constant and D is the maximum depth of the generated circuit. Though the reconvergence path makes this simplification less accurate than the simulation result, it still indicates the power consumption as a heuristic. Using large K can also reduce the induced error. We offer a brief proof below:

Consider a circuit with an input probability pattern $(p_1, p_2, ..., p_n)$. We define the node's depth (level) as the maximum number of nodes (including itself) connecting it to any primary inputs. We propose that output signal probability error at d-level nodes is bounded by $\frac{d+1}{k}$ and thus by $\frac{CD}{K}$. We prove this proposition by mathematical induction. For 0-level nodes (primary inputs), the signal error is bounded by $\frac{1}{K}$, the same as the subinterval length. Assume the proposition holds for (d-1)-level nodes, and we check the output signal error for d-level nodes. Output signal probability S_p is a function of input signal probability. We denote the input signal probability error as ϵ. We calculate the output signal probability error function $f_e = \max(|S_p(p_1 \pm \epsilon_1, p_2 \pm \epsilon_2) - S_p(p_1, p_2)|)$ of all sixteen two-input Boolean functions as in Table 1, where p_1, p_2 are actual input signal probability of d-level nodes. Note that p_1, p_2 are in $[0, 1]$, and ϵ_1, ϵ_2 are bounded by $\frac{d}{k}$ for $(k-1)$-level nodes. Therefore, error functions are bounded by $\frac{d+1}{K}$ when $K > D + 2D^2$ for k-level nodes, and thus, the proposition holds. As for switching probability, the error function for d-level nodes is smaller than $2|(p + \frac{d}{k})(1 - p - \frac{d}{k}) - p(1-p))|$, bounded by $\frac{CD}{K}$.

Boolean Function	f_e
0x0, 0xf	0
0x1, 0xe	$p_1\epsilon_2 + p_2\epsilon_1 + \epsilon_1\epsilon_2$
0x2, 0x4, 0xb, 0xd	$(1 - p_{1/2})\epsilon_2 + p_{2/1}\epsilon_1 + \epsilon_1\epsilon_2$
0x3, 0x5, 0xa, 0xc	$\epsilon_{1/2}$
0x7, 0x8	$(1 - p_1)\epsilon_2 + (1 - p_2)\epsilon_1 + \epsilon_1\epsilon_2$
0x6, 0x9	$\lvert\epsilon_1(2p_2 - 1) \pm \epsilon_2(2p_1 - 1) \pm 2\epsilon_1\epsilon_2\rvert$

Table 1: Output signal probability error functions.

Since a node's input signal probability and Boolean function decide its output signal probability, signal probability p_i can propagate inside a circuit with variables s_{ijk} and f_{iuv}. Selection variables s_{ijk} specify the relationship $x_i = x_j o_i x_k$ while function variables f_{iuv} specify the Boolean function o_i. The truth table of x_i's Boolean function can be seen as a sum-of-product formula. The output signal becomes 1 only when input node signals make one product term 1. We can calculate the signal probability p_i by summing up the product of the correspondent signal probability for each term related to p_j and p_k. Therefore, we propagate p_i through the whole circuit by adding a set of SMT clauses:

$$p_i = \sum (s_{ijk} \times \sum_{uv} f_{iuv} \times p_{ju} \times p_{kv}) \tag{2}$$

p_{ju} equals p_j when constant value $u = 1$, and p_{ju} is $1 - p_j$ when $u = 0$. Similarly, p_{kv} depends on the value of v. Then, we calculate

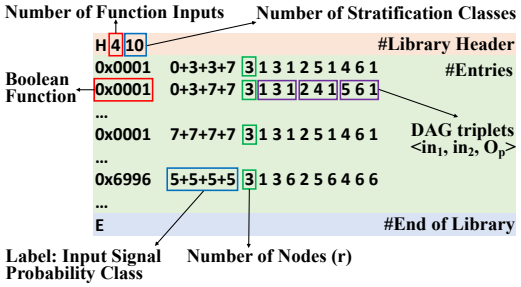

Figure 4: Illustration of 4-input power-oriented library with 10 stratification classes.

the average switching probability based on the signal probability of each node through the SMT clauses:

$$r_p = \frac{2\sum_{n+1}^{n+r} p_i \times (1 - p_i)}{r} \tag{3}$$

Finally, we constrain the allowed power resource via the clause:

$$r_p < 1/K \tag{4}$$

K increments by unit at each step until the encoded SMT problem has a satisfiable assignment.

4.3 Library Building

The previous section elaborates on how we shrink the infinite search space to a finite one. However, building a power-oriented library with exponentially increasing sub-circuit entries as K increases is still challenging. Therefore, we propose an equioptimizable pattern that further reduces search space by merging input signal probability patterns corresponding to the same circuit. The flow of PONO finds the circuit with minimum $r_p \times r$ to implement a specific function with a given input signal probability P. Different input signal probability patterns may apply to the circuit with the same internal structure, and we call these input patterns equioptimizable.

A circuit's internal structure depends on allowed resource r and variables s_{ijk} and f_{iuv} assignments. When internal structures are specified, r_p is a function of input signal probability P as in Equations (2) and (3). Therefore, $r_p \times r$ defined on $[0, 1]^n$ is a piecewise function. Subfunctions of $r_p \times r$ are differentiable elementary functions that alternate only when the generated circuit changes. The subdomain of each subfunction corresponds to a set of input signal probability patterns. Since equioptimizable patterns are continuous in value, we only need to record the changing-point value of input patterns causing circuit change.

The designed power-oriented library contains a header and a list of entries (label-circuit pairs), as shown in Figure 4. It only records P classes of functions to compact the library size. The label of a circuit is a combination of a targeted Boolean function truth table and a changing-point input signal pattern. Header records basic information like the number of inputs and stratification classes K. We design a sorting rule for n-input signal probability patterns $P_1 = (p_1^1, p_2^1, ..., p_n^1)$ and $P_2 = (p_1^2, p_2^2, ..., p_n^2)$:

$$P_1 < P_2 \iff \exists\, i\ (p_i^1 < p_i^2\ \ \&\ \ \forall\, j < i\ (p_j^1 = p_j^2)) \tag{5}$$

The label of each entry records the P that input signal probability $P_i < P$ corresponds to a different circuit from the counterpart corresponding to P. Entries having the same truth table list in

Algorithm 1 Power-oriented Logic Rewrite

Input: Netlist N, Library L, Input Signal Probability P,

```
LUT_Mapping(N);
Initialize_Signal_Probability(N, P);
for node ∈ N do
    Cuts = node.inputs;
    P_cut = [ ]
    for input ∈ Cuts do
        P_cut.insert(input.signal_probability);
    end for
    Sub_Circuit = L.find(node.function, P_cut);
    Substitute(N, node, Cuts, Sub_Circuit);
end for
```

ascending order of P. The circuit is stored as a direct acrylic graph (DAG) represented by r triplets $\langle in_1, in_2, O_p \rangle$. in_1 and in_2 are the node input indices, and o_p is the opcode of the node's function. Allowed operations of o_p are sixteen 2-input Boolean functions. Since the number of nodes in a generated sub-circuit is less than twenty for 4-input functions, we compact the triplet into an INT16 type for further library size reduction. The library building can run in parallel since generation for different Boolean functions and input signal patterns are independent.

4.4 Application in Logic Synthesis

Logic rewrite transformation replaces sub-circuits with equivalent pre-computed subgraphs for each node [13]. The sub-circuit library plays a vital role in the quality of the rewrite. We design a power-oriented rewrite flow by applying the generated power-oriented circuit library. Unlike the original rewrite flow, PONO needs to calculate the initial signal probability of all nodes to look up corresponding pre-computed circuits. Users can decide how to gain signal probability by either probability propagation or workload simulation in the trade-off between accuracy and execution time.

Algorithm 1 elaborates on the flow of power-oriented rewrite in PONO. It inputs circuit netlist N and finishes LUT mapping to gain 4-cuts for rewrite transformation. Then, it calculates the signal probability of all nodes based on the input signal probability or workload vector. Probability propagation can finish in linear time, while workload simulation needs more time in exchange for higher accuracy. After that, it traverses each 4-cut in the netlist, looks up an equivalent sub-circuit from the power-oriented library, and replaces the cut with the pre-computed sub-netlist. The sub-circuit searching process first locates a set of entries based on the Boolean function and then finds the power-oriented circuit by binary search according to the input signal probability pattern. Experiments show that this power-oriented rewrite flow can reduce power consumption with comparable area and latency.

5 EVALUATION OF PONO

In this section, we evaluate the efficacy of PONO. We implement PONO in C++ and test it on a server with 2-way Intel Xeon Gold 6248R CPUs and 512GB DDR4 memory. We compare PONO with LUT-based rewrite using state-of-the-art exact synthesis flow [6] (baseline) and ABC rewrite [14], which do not consider power

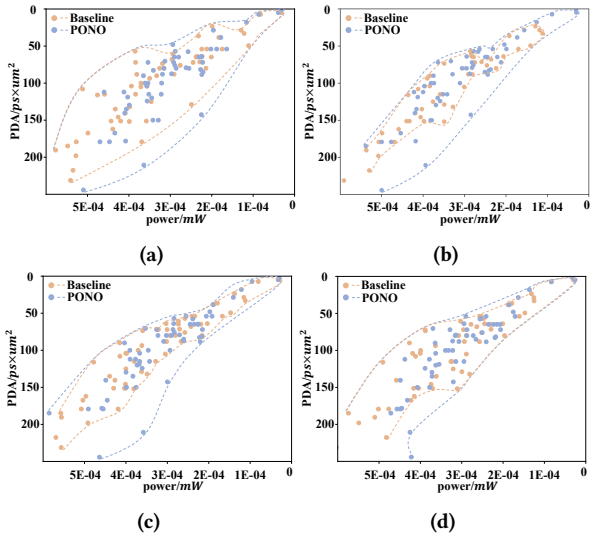

Figure 5: Quality of generated sub-circuits under input signal probability patterns: (a): (0.2, 0.2, 0.2, 0.2); (b): (0.4, 0.4, 0.4, 0.4); (c): (0.6, 0.6, 0.6, 0.6); (d): (0.8, 0.8, 0.8, 0.8).

consumption. Our experiments verify that PONO can effectively generate a power-oriented library and help rewrite transformation to reduce power consumption. We limit SAT/SMT solving time to 18 hours, and 157 out of 3984 cases exceed the time limitation. The library generation process for 4-input functions takes about seven days. Number of stratification classes K is set to 10. We evaluate our framework on EPFL arithmetic and random control benchmarks [1] with NanGate 15nm [10] technology. When evaluating the results, we use ABC for technology mapping and the commercial tool Innovus (v18.12-s106 1) for placement and routing to get accurate power, latency, and area value.

We compare the library generated by PONO and baseline flow [6] to verify the quality of generated power-oriented near optimal circuits. We conduct experiments on P classes of 4-input Boolean functions under four different input signal probability patterns: (0.2, 0.2, 0.2, 0.2), (0.4, 0.4, 0.4, 0.4), (0.6, 0.6, 0.6, 0.6), and (0.8, 0.8, 0.8, 0.8). Figure 5 presents the comparison result of the generated circuits' quality, where PDA is the product of delay (latency) and area. PONO can push the Pareto frontier (PDA and power) forward under tested input signal probability patterns. This manifests the circuit quality superiority over the original exact synthesis flow. Therefore, the generated power-oriented library can optimize the circuit in power, performance (latency), and area simultaneously rather than just considering the latter two metrics.

We then verify the optimizing effect when applying a power-oriented library in LUT-based rewrite transformation on EPFL arithmetic and random control benchmarks. We randomly generate ten sets of workload vectors to obtain input signal probability and calculate the average power result for evaluation. We compare the power-oriented rewrite flow in Section 4.4 with LUT-based rewrite using the library generated by the original exact synthesis flow [6] and ABC rewrite [14]. Table 2 shows the tested circuits' power, performance, and area. PONO can effectively optimize 8.7% more power consumption than the baseline. It also outperforms ABC

Case	Original Circuits			ABC rewrite			Baseline			LUT-rewrite using PONO		
	Power (mW)	Latency (ps)	Area (um^2)	Power Saving	Latency Saving	Area Saving	Power Saving	Latency Saving	Area Saving	Power Saving	Latency Saving	Area Saving
adder	2.70E+02	893.90	197.35	-34.06%	-5.21%	-37.04%	-47.33%	51.15%	-55.13%	-33.69%	46.08%	-66.10%
bar	1.63E+03	394.80	652.05	14.26%	5.45%	12.41%	21.27%	-21.05%	27.38%	37.31%	-2.81%	6.59%
div	3.61E+04	23106.95	14922.20	27.61%	0.36%	18.43%	-23.79%	-10.45%	20.50%	15.29%	11.81%	3.02%
hyp	9.66E+05	273878.59	48562.91	1.75%	39.76%	-7.66%	2.47%	32.29%	-1.52%	6.03%	39.48%	-6.94%
log2	5.86E+04	3117.10	6142.03	-5.28%	-0.63%	-1.74%	-5.58%	1.80%	-19.74%	12.05%	11.40%	21.93%
max	3.51E+03	1317.20	687.78	-13.38%	-23.15%	-2.84%	6.87%	2.44%	14.41%	21.76%	9.54%	8.42%
multiplier	4.87E+04	3370.91	6362.68	9.31%	51.84%	10.03%	21.49%	54.07%	11.35%	-0.12%	56.78%	-4.68%
sin	8.58E+03	1399.60	1323.07	-9.12%	9.83%	-6.68%	0.17%	2.49%	-3.27%	9.58%	-0.69%	-2.76%
sqrt	1.65E+05	68334.73	6345.87	2.66%	12.27%	15.65%	54.16%	-18.91%	14.36%	24.25%	-5.65%	12.66%
square	1.34E+04	1224.50	3602.74	-9.00%	-17.64%	-0.63%	-29.32%	16.35%	-12.36%	-4.99%	11.50%	-13.75%
arbiter	3.78E+03	429.20	1888.96	0.50%	-0.05%	0.00%	72.91%	27.77%	29.45%	77.83%	22.65%	33.12%
cavlc	1.88E+02	127.60	120.18	4.68%	13.48%	3.31%	-0.85%	0.47%	0.67%	4.09%	7.92%	-2.00%
ctrl	3.29E+01	39.10	26.64	26.45%	-21.23%	18.27%	16.97%	-11.76%	12.61%	26.54%	-5.88%	8.56%
dec	4.29E+01	61.30	62.62	-2.80%	-20.72%	-0.16%	-9.31%	-4.73%	-1.10%	4.22%	-1.35%	1.46%
i2c	2.35E+02	89.30	227.38	6.30%	-4.03%	3.96%	4.39%	-6.72%	12.30%	6.90%	-4.14%	7.37%
int2float	4.92E+01	67.30	39.08	4.71%	0.45%	4.28%	5.34%	9.96%	-1.38%	2.64%	2.82%	2.23%
mem_ctrl	2.43E+04	2482.80	8217.13	-2.63%	17.63%	-0.37%	9.82%	15.70%	6.01%	9.98%	19.69%	-1.26%
priority	5.28E+02	778.50	273.83	33.64%	46.29%	24.77%	-4.17%	35.35%	12.85%	35.00%	-10.08%	10.57%
router	6.47E+01	133.30	60.95	14.39%	7.28%	11.30%	14.58%	10.05%	13.31%	17.79%	3.53%	5.41%
voter	2.12E+04	417.70	4592.22	19.83%	13.57%	19.76%	32.81%	4.48%	12.15%	44.90%	3.33%	11.38%
average	-	-	-	4.49%	6.28%	4.25%	7.15%	9.54%	4.64%	**15.87%**	**10.79%**	**1.76%**

Table 2: Power, performance (latency), and area of circuits before and after LUT-based rewrite using PONO library.

rewrite with an 11.4% higher power reduction. Meanwhile, it has comparable performance and area optimization results with tolerable overhead in rare cases. Such results verify the efficacy of PONO in optimizing power consumption.

6 CONCLUSION AND FUTURE WORK

This paper presents a power optimization method by generating near-optimal power-oriented sub-circuits using SMT. We enable the creation of power-oriented libraries by introducing probability stratification and equioptimizable input patterns. These techniques shrink the infinite search space into a finite one. We formulate the circuit generation for power optimization as an SMT problem by extending SSV encoding. Employing the generated power-oriented library in LUT-based rewrite manifests superiority in power reduction over the power-oblivious rewrite library by ABC and exact synthesis. PONO reduces 8.7% more power consumption compared to the baseline on the EPFL benchmarks, with comparable area and latency performance, and it also outperforms ABC rewrite regarding power reduction. In the future, we will explore PONO's application in complete logic synthesis flow and standard cell library construction for power optimization.

ACKNOWLEDGMENT

This work was partly supported by the National Key R&D Program of China (Grant No. 2022YFB4500500) and the National Natural Science Foundation of China (Grant No. 62090021).

REFERENCES

[1] Luca Amarú et al. 2015. The EPFL combinational benchmark suite. In *IWLS*.
[2] Luca Amarú et al. 2017. Enabling exact delay synthesis. In *ICCAD*.
[3] Leonardo De Moura et al. 2008. Z3: An efficient SMT solver. In *TACAS*.
[4] Xianliang Ge et al. 2022. Topology-Based Exact Synthesis for Majority Inverter Graph. In *ISCAS*.
[5] Winston Haaswijk et al. 2018. SAT Based Exact Synthesis using DAG Topology Families. In *DAC*.
[6] Winston Haaswijk et al. 2019. SAT-based exact synthesis: Encodings, topology families, and parallelism. *IEEE TCAD* (2019).
[7] DE Knuth. 2022. The art of computer programming: Volume 4B.
[8] Eugene L Lawler. 1964. An approach to multilevel Boolean minimization. *JACM* (1964).
[9] Dewmini Sudara Marakkalage et al. 2023. Fanout-Bounded Logic Synthesis for Emerging Technologies-A Top-Down Approach. In *DATE*.
[10] Mayler Martins et al. 2015. Open cell library in 15nm FreePDK technology. In *ISPD*.
[11] Alan Mishchenko. 2001. An approach to disjoint-support decomposition of logic functions. *Tech. Report* (2001).
[12] Alan Mishchenko. 2014. Enumeration of irredundant circuit structures. In *IWLS*.
[13] Alan Mishchenko et al. 2006. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *DAC*.
[14] Alan Mishchenko et al. 2007. ABC: A system for sequential synthesis and verification. *URL http://www.eecs.berkeley.edu/alanmi/abc* (2007).
[15] Sayak Ray et al. 2012. Mapping into LUT structures. In *DATE*.
[16] Robert Allan Smith. 1965. Minimal three-variable NOR and NAND logic circuits. *IEEE Transactions on Electronic Computers* (1965).
[17] Mathias Soeken et al. 2018. Practical exact synthesis. In *DATE*.
[18] Weihua Xiao et al. 2023. MiniTNtk: An Exact Synthesis-based Method for Minimizing Transistor Network. In *ICCAD*.
[19] Zhiyao Xie et al. 2022. DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters. In *ICCAD*.
[20] Xindi Zhang et al. 2022. ParKissat: Random Shuffle Based and Pre-processing Extended Parallel Solvers with Clause Sharing. *SAT Competition*.
[21] Sunan Zou et al. 2023. PowerSyn: A Logic Synthesis Framework with Early Power Optimization. *IEEE TCAD* (2023).