

Simopt-Power: Leveraging Simulation Metadata for Low-Power Design Synthesis

Eashan Wadhwa & Shanker Shreejith

Department of Electronic and Electrical Engineering, Trinity College Dublin

Dublin, Ireland

{wadhwa, shreejith.shanker}@tcd.ie

Abstract—Excessive switching activity is a primary contributor to dynamic power dissipation in modern FPGAs, where fine-grained configurability amplifies signal toggling and associated capacitance. Conventional low-power techniques – gating, clock-domain partitioning, and placement-aware netlist rewrites – either require intrusive design changes or offer diminishing returns as device densities grow. In this work, we present Simopt-power, a simulator-driven optimisation framework that leverages simulation analysis to identify and selectively reconfigure high-toggle paths. By feeding activity profiles back into a lightweight transformation pass, Simopt-power judiciously inserts duplicate truth table logic using Shannon Decomposition principle and relocates critical nets, thereby attenuating unnecessary transitions without perturbing functional behaviour. We evaluated this framework on open-source RTLLM benchmark, with Simopt-power achieves an average switching-induced power reduction of $\approx 9\%$ while incurring only $\approx 19\%$ additional LUT-equivalent resources for arithmetic designs. These results demonstrate that coupling simulation insights with targeted optimisations can yield a reduced dynamic power, offering a practical path toward using simulation metadata in the FPGA-CAD flow.

Index Terms—Simulators, Power, FPGA, ASIC, Synthesis, Compilers

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) have emerged as critical components in modern digital systems, offering flexible hardware reconfigurability, rapid prototyping capabilities, and optimised parallel execution. As these devices increasingly permeate various application domains – from embedded systems to high-performance computing – power efficiency and reduction in switching activity have become paramount design considerations. Switching activity, closely correlated with dynamic power dissipation, significantly impacts FPGA reliability, performance, and overall system longevity. FPGA design tools play a pivotal role in managing and minimising this switching activity through optimisation algorithms and resource allocation strategies during synthesis, mapping, placement, and routing phases. Dynamic power in FPGAs arises predominantly from switching activity, with factors such as logic transitions, clock distribution networks, and signal routing contributing significantly.

Despite the growing sophistication of FPGA tools, effectively managing switching activity remains a substantial challenge, intricately tied to the increasing complexity of devices, shrinking technology nodes, and demanding performance constraints. For decades, efforts to curb dynamic power

in FPGAs have centred on fine-grained techniques such as dynamic voltage and frequency scaling (DVFS) [1], power gating of idle regions [2], and clock gating based on toggling statistics—strategies [3] now embedded in most modern FPGA-CAD tools. Yet these approaches focus exclusively on controlling activity *after* a design is committed to hardware; they do not exploit insights available during simulation. In every hardware designer’s workflow, functional simulation serves solely to validate correctness and is subsequently discarded, leaving a rich trove of toggle-rate information untapped for power optimisation.

This paper addresses this gap by analysing the efficacy of modern FPGA synthesis and placement-and-routing tools in minimising switching activity through simulations, thereby optimising dynamic power usage. The specific contributions of this work are outlined below:

- We extend an existing simulation infrastructure to harvest fine-grained switching-activity metadata, yielding per-signal toggle estimates across the entire design.
- We feed these activity profiles into a new truth-table decomposition engine that explicitly balances power reduction against the accompanying area overhead.
- We validate the approach on cutting-edge, large-language-model-generated Verilog suites (RTLLM [4]) alongside the standard VTR benchmarks [5], confirming significant power savings with acceptable area trade-offs.

We present a modest switching-induced power reduction of $\approx 9\%$ while incurring only $\approx 19\%$ additional LUT-equivalent resources for arithmetic designs. While providing empirical insights into the impact of tool-driven optimisations, evaluate their trade-offs against performance metrics, and suggest future directions for FPGA tool development to achieve enhanced power-aware computing.

II. PRELIMINARIES

A. Verilog-To-Routing and Simulators

The open-source Verilog-To-Routing [6] toolflow has been a key enabler for research into CAD optimisations for FPGAs and ASICs. It benefits from open-source community backing and seamless integration into custom and vendor toolflows, while delivering competitive power, area, and performance results for FPGA/ASIC designs [7]–[9]. Many researchers have expanded the original VTR flow with custom extensions for novel architecture exploration, enhanced routing

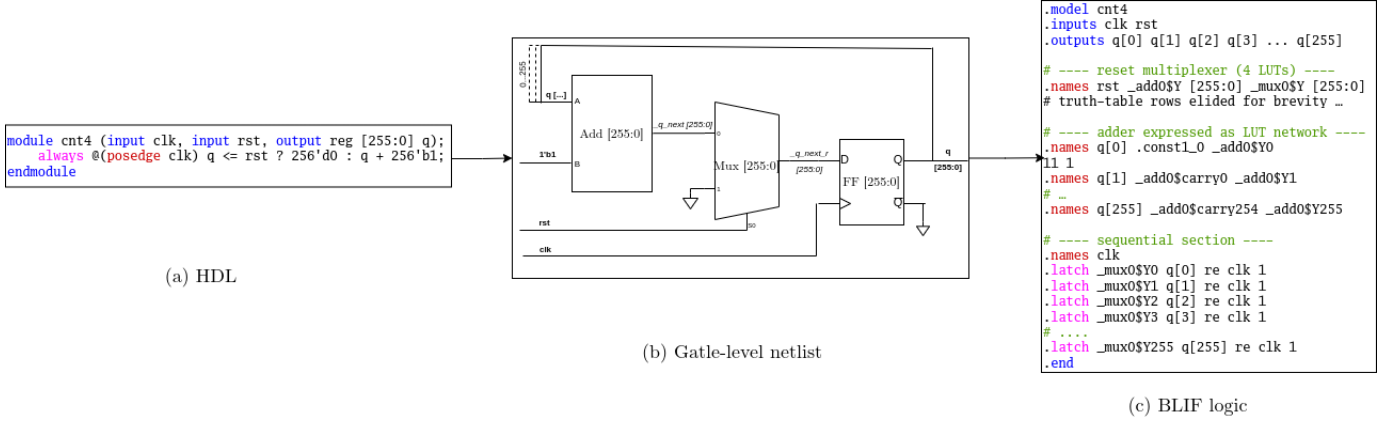


Fig. 1: A standard Yosys synthesis flow, where HDL is synthesised to a gate-level netlist, followed by the generation of a BLIF logic netlist.

algorithms and others. In [7], the authors enhanced the existing VTR flow for supporting a custom architecture which replaces expensive logic (connection, switching) blocks. The authors in [8] implemented a fine-grained optimisation inside Berkley’s ABC [10] on Intermediate representations (IR) graphs to lower the area incurred by designs mapped through VTR. The VTR backend is organised as a modular collection of toolsets, each implementing a distinct stage of the FPGA CAD flow—from logic synthesis through packing, placement, routing, and timing analysis—thereby enabling an end-to-end, open-source evaluation framework [6]. This openness has empowered researchers to prototype and evaluate cutting-edge optimisation algorithms that drive measurable gains across critical performance metrics. A recent example is MapTune [11], which proposes a reinforcement learning framework for guided mapping of cells during the synthesis step, reducing delay and area by a considerable amount. Simopt [9] introduces simulation metadata into the FPGA-CAD synthesis flow, emitting lower latency designs for FPGAs. Using simulations in accelerating efficiency has also been studied in other domains. Combinational equivalence checking of circuit networks is accelerated by studying simulation patterns [12] markedly reducing SAT-solver runtime and verification effort. FPGA-Tidbits [13], a Chisel-based library generates fully synthesisable RTL and employs Verilator for cycle-accurate co-simulation, enabling rapid co-design and prototyping of area-efficient FPGA accelerators. Previous research in simulators however have never addressed the problem of reducing power consumption in generated designs, which our work addresses.

B. FPGA netlist to placed design

Fig. 1 shows canonical front-end flow of how synthesis tools input a Hardware-Description Language (HDL) snippet to emitting a netlist. After parsing the HDL, a boolean network is created which map arithmetic operators, muxes and comparators to logic operations while having no knowledge of platform specific cell libraries. This can then be directly mapped to gate-level netlists, shown as (b) in Fig. 1 while incorporating gate-sizing, and library patterns for k-input Look-up tables (LUTs). The final step of netlist emission, (c) in Fig. 1, includes cell

libraries to map the gate-level netlist to capture the technology-mapped boolean network. In our work we use the standard open-source front-end for FPGA research flows - Yosys, which parses into an internal RTL Intermediate Language (RTLIL) representation, an equivalent of a boolean network. A series of optimisations are performed on this internally represented abstract tree, followed by mapping which replaces the RTLIL primitives with ABC’s gate library to generate out the gate-level netlist. It is then incrementally lowered to a generic BLIF netlist through structural hashing and cut enumeration implemented in Berkeley’s ABC. This netlist containing LUT, FF and latch directives can then be fed into placement tools for *placing* the design onto an FPGA platform. Placement determines the physical coordinates on the FPGA fabric contained in the design netlist. Placement typically is directed by timing closure, dynamic power and congestion factor. Modern placers integrate routability estimators that penalize hot-spots of overlapping demand, while power-aware heuristics redistribute high dynamic power consuming nets to shorten high-activity wires and avoid thermal concentration. However by this stage, there is very little a designer can do to reduce switching activity for designs. Simopt-power mitigates this issue by judiciously reducing the capacitance associated with high-toggling signals in-turn reducing the dynamic power consumption as we explain later in this work. But before the resource mapping is solidified through synthesis and the spatial arrangement finalised by placement, we first turn to simulations to verify that a configured design meets all functional and timing requirements before committing it to hardware.

C. Simulation in Simopt

Simulations offer a virtual environment where designs are exercised under stimuli, enabling early verification and correctness of functionality before any hardware resources are committed. Although designers can choose from a range of commercial and open-source simulators, these tools typically fail to expose the internal metrics necessary to inform subsequent synthesis and placement stages.

In [9] however, authors introduced a Simopt framework which have modified Verilator to extract simulation metadata from a design to generate *simopt-counters* (shown in Fig. 2, as the *Protobuf-encoded Simopt dump*). *Simopt-counters* tracks number of times a simulated net is toggled for a given set of input stimuli, which was later used by various *Simopt-backends*. Authors demonstrated this framework by integrating these counters into one such open-source framework, Yosys to generate lower latency designs. However, their findings only focused on one such backend and only combinatorial circuits only. In our work, we re-use these *simopt-counters* as *switching activity* estimates since they essentially give similar counts to toggling activity of a net. We explain how we re-used this framework in the following section III.

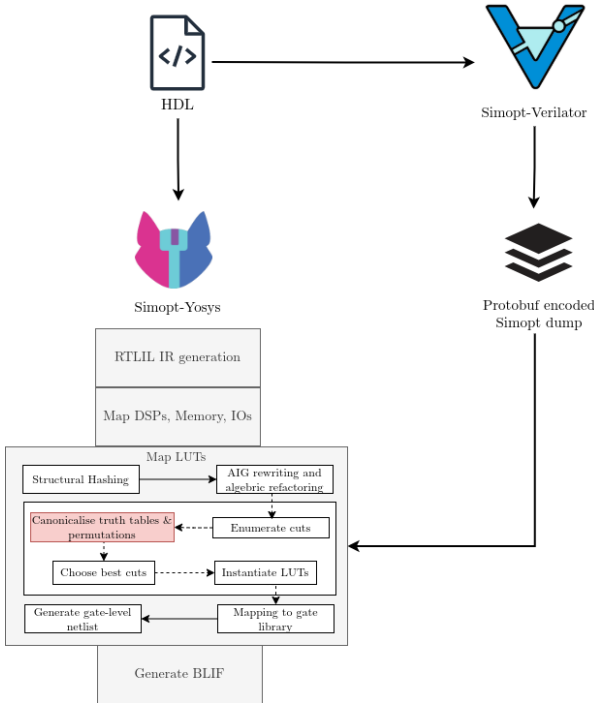


Fig. 2: How Simopt-Power integrates into the Simopt framework. The red box indicates the step where the proposed truth-table decomposition step is integrated.

III. SIMOPT-POWER

Power consumption on modern FPGAs is primarily composed of static power and dynamic power, with the latter contributed by the dynamic switching activity of the active logic elements. Dynamic power on an FPGA can be captured through the switching power equation

$$\approx \alpha \cdot C \cdot V^2 \cdot f$$

where C , V , f and α represent effective switch capacitance, supply voltage, clock frequency and switching factor (i.e. the average number of signal transitions per node of a circuit), respectively. Dynamic power consumption optimisation thus requires the tools to minimise the impact of each of these parameters where possible. V is a device-level parameter which can have a significant impact on signal integrity, whereas in

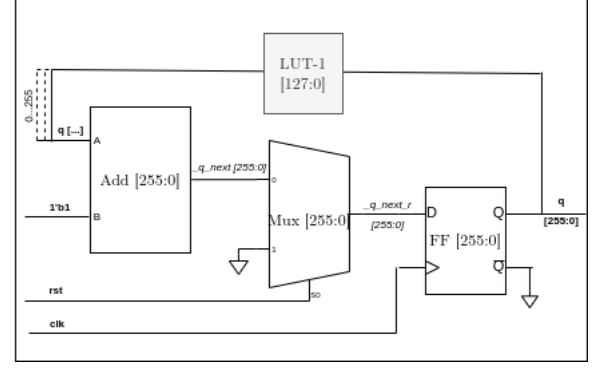


Fig. 3: Simopt-power truth-table decomposed synthesis netlist of the counter from Fig.1(b). The grey box indicates the additional LUT logic added by the Simopt-Power framework to reduce power.

high-performance designs, f should be as high as possible. Prior research has explored the possibility of dynamic voltage frequency scaling (DVFS) [14] and clock gating to optimise power consumption at runtime using offline characterisation of designs [14]. Our approach is to optimise the design-time parameters α and C to minimise the power consumption on the logic when deployed on the FPGA. By duplicating the driving logic, each copy of the driving logic can be placed physically close to a subset of the loads, reducing the wire length to drive the logic to result in a smaller overall C , and/or reducing the number of programmable interconnect switches that the logic needs to pass through, resulting also in lower overall C . The duplication of driving logic, as we also observe from our results, typically results in an additional few μW 's, which is outweighed by the drop in dynamic power [1]–[3].

To determine the signal toggle levels under actual input conditions, the design is initially simulated with Simopt-Verilator with a test fixture that replicates real-world data that the system could observe. Simopt-Verilator records per-cycle toggle counts, referred to as *simopt-counters*, for every signal and register (entity). These activity profiles are serialised as Protobuf-encoded metadata files and later parsed by Simopt-Power, which uses the counters to drive its power-reduction transformations. The *simopt-counters* tags each signal selected in the design, and its value is incremented each time the signal state toggles when the design is simulated using the Simopt-Verilator framework. To illustrate this flow, consider the simple example in Fig. 1, where a free-running clock is fed as an input clock clk for 512 ticks (Fig. 1(a)). The gate-level netlist of the counter, shown in Fig. 1(b), uses a multi-bit vector q to store the counter's state whose bit position toggles at the clock edge depending on the current state, and is individually tracked by the corresponding *simopt-counter*. The internal next-state wires, q_next , q_next_r shown in Fig. 1(b) are also tracked by separate *simopt-counters*. During the synthesis flow, the relation between *simopt-counters*, Verilator's intermediate signal representations, and Yosys's synthesised net names are maintained by the Simopt-Verilator flow.

Once the *simopt-counters* are determined for each signal in the design, this can be forwarded to the downstream processing

Algorithm 1: TRUTHTABLEDECOMPOSE – Shannon-Decomposes a truth-table

Input: T – pointer to the truth table on the *right* cut of size n_{right} variables;
 C_{left} – indices of the n_{left} variables in the cut;
 C_{right} – indices of the n_{right} variables in the cut;
Result: T is replaced by the Shannon-decomposed truth-table with positive and negative co-factors.

```

1 if not  $C_{\text{simopt\_counter}} < \text{simopt\_counters\_median}$  then
2   return
3 end
4 for  $i \leftarrow 0$  to  $n_{\text{right}} - 1$  do
5    $\text{splitVar} \leftarrow -1$ ;
6   if  $C_{\text{right}}[i] \notin C_{\text{left}}$  then
7      $\text{splitVar} \leftarrow C_{\text{right}}[i]$ ; break;
8   end
9   if  $\text{splitVar} = -1$  then
10    continue; // nothing to split
11  end
12   $\text{pos} \leftarrow \text{index of } \text{splitVar} \text{ in } C_{\text{right}}$ ;
13   $n_{\text{words}} \leftarrow 2^{n_{\text{right}}} / (8 \cdot \text{sizeof}(\text{word}))$ ;
14  allocate arrays  $T_0, T_1$  of  $n_{\text{words}}$  words;
15   $T_0 \leftarrow \text{SHANNONCOFACTOR}(T, n_{\text{right}}, \text{pos}, 0)$ ;
16   $T_1 \leftarrow \text{SHANNONCOFACTOR}(T, n_{\text{right}}, \text{pos}, 1)$ ;
17  copy  $T_1, T_0$  into  $T$ ;
18  free  $T_0, T_1$ ;
19 end
20 for  $i \leftarrow 0$  to  $n_{\text{left}} - 1$  do
21    $\text{splitVar} \leftarrow -1$ ;
22   if  $C_{\text{left}}[i] \notin C_{\text{right}}$  then
23      $\text{splitVar} \leftarrow C_{\text{left}}[i]$ ; break;
24   end
25   if  $\text{splitVar} = -1$  then
26    continue; // nothing to split
27  end
28   $\text{pos} \leftarrow \text{index of } \text{splitVar} \text{ in } C_{\text{left}}$ ;
29   $n_{\text{words}} \leftarrow 2^{n_{\text{left}}} / (8 \cdot \text{sizeof}(\text{word}))$ ;
30  allocate arrays  $T_0, T_1$  of  $n_{\text{words}}$  words;
31   $T_0 \leftarrow \text{SHANNONCOFACTOR}(T, n_{\text{left}}, \text{pos}, 0)$ ;
32   $T_1 \leftarrow \text{SHANNONCOFACTOR}(T, n_{\text{left}}, \text{pos}, 1)$ ;
33  copy  $T_1, T_0$  into  $T$ ;
34  free  $T_0, T_1$ ;
35 end

```

logic within the mapping-placement-routing flow in Yosys, captured in Fig. 2. The highlighted block in red in the flow is the Simopt-power extension, which implements the truth-table decomposition algorithm to determine logic duplication based on the simopt dump from the simulation. In the traditional Yosys flow, the RTL graph IR is structurally cleaned before synthesising the logic using ABC to generate a final netlist shown in Fig. 1(c). ABC performs iterative logic restructuring, enumerating small graphs (cuts) and computes canonical truth tables using Boolean-function hashing. These truth tables are utilised during the technology mapping phase, where the

Algorithm 2: SHANNONCOFACTOR – Calculates Shannon cofactor of truth-table

Input: T_{src} – pointer to the original truth table on n variables;
 n – number of variables;
 p – position of variable to fix;
 v – value to cofactor on ($v \in \{0, 1\}$)
Output: T_{dst} – cofactor of T_{src} with $x_p = v$

```

1  $n_{\text{words}} \leftarrow 2^n / (8 \cdot \text{sizeof}(\text{word}))$ ;
2 for  $i \leftarrow 0$  to  $n_{\text{words}} - 1$  do
3    $T_{\text{dst}}[i] \leftarrow 0$ ;
4 end
// Scan every minterm and check if it matches  $x_p = v$ 
5 for  $m \leftarrow 0$  to  $2^n - 1$  do
6   if bit  $m$  is set in  $T_{\text{src}}$  then
7     if  $(m \gg p) \& 1 = v$  then
8        $m' \leftarrow \text{remove bit } p \text{ from } m \text{ (collapse to } n-1 \text{ vars)}$ ;
9       set bit  $m'$  in  $T_{\text{dst}}$ ;
10    end
11  end
12 end

```

functions are packed into k-input LUTs or specific library functions. Simopt-power introduces an additional decomposition pass, described further in Alg. 1, to replace the existing truth table pointers for nets that have high toggle rates with an alternate one that further decomposes the logic to minimise the logic loading. The net-level *simopt-counter* data extracted from the simulation is thresholded using the median *simopt-counter* value across the design to identify the logic signals that need further decomposition. For signals with high activity, the Shannon cofactors are calculated for the corresponding truth tables using the Shannon decomposition function

$$f(x) = x f(1) + \bar{x} f(0)$$

where the boolean function $f(x)$ is decomposed into its positive and negative cofactors with respect to x .

In Alg. 2, the Shannon cofactor of a Boolean function with respect to a variable x_p is computed by scanning all 2^n minterms of the original truth table (line 5) and selecting only those for which $x_p = v$ holds (line 6-7). For each selected minterm, the index is compressed by removing bit position p to obtain a reduced $(n - 1)$ variable representation (line 8), and the corresponding output bit is set in the destination truth table $T * \text{dst}$ (line 9). This process yields the restricted function $f * x_p = v$ as a new truth table (lines 5–9), enabling decomposition in truth-table-based logic optimisation. Cuts introducing such unique variables are heuristically strong candidates for splitting through Shannon decomposition as they represent functional expansions that were not previously factorised in the logic graph. These new variables correspond to signal lines whose influence has not yet been resolved in either subtree. As such, they typically exhibit higher switching activity, since they inject additional functional control into the

resulting cut function. In Alg. 1, these variables are called *splitVar*, and are explained in lines 4-11 for the right cut graph, and lines 21-27 for the left cut graph.

The copy statement for truth-tables T_1, T_0 in line 17,33, implements a *contiguous-memory concatenation* that rebuilds a single, 2^{n-1} -bit truth table from its Shannon cofactors. Each cofactor, T_1 and T_0 , already holds the function’s values under fixed assignments of 0 and 1, respectively, and thus contains 2^{n-1} bits (stored here as an array of n_{words} elements). Since cofactors were computed with identical variable ordering for the remaining $n-1$ variables, a simple block copy concatenates the table without requiring further bit shuffling. The operation merges the two half-tables into the full table, setting the stage for subsequent synthesis steps without changing the function’s semantics.

With the updated tables, the standard ABC flow resumes by mapping the LUTs to a gate-level netlist for Yosys to generate a BLIF netlist. With reference to Fig.2(b) the new netlist would like Fig.3, with the additional decomposed netlists. For implementing these designs on a commercial FPGA, these netlists are imported as a black box into AMD’s Vivado tool as an IP and subsequently, to generate the power reports for estimating the dynamic power consumption of the toggle-optimised design.

IV. RESULTS

In order to evaluate Simopt-power, we picked datasets which contained sufficient sequential logic to measure power benefits. This is because it is more complex to instrument for switching-activity measurements and for formal reasoning about state traversal (increasing toggle activity concentration), whereas a purely combinational circuit toggles nearly every gate on every evaluation deterministically and may not reap any benefits from this framework. Since designs have to be Simopt-Verilator and Simopt-Yosys compatible and need to be sufficiently complex to have large sequential circuits, we chose

- *RTLLM* dataset [4]: an open-source benchmark that poses progressively complex hardware design tasks and scores large-language-model outputs on syntax, functional correctness and implementation quality. Some of the RTL circuits generated by the large language model are complex and are syntactically valid for evaluating the Simopt-Power framework
- *Koios* dataset [5]: Koios is an open-source benchmark suite comprising of deep-learning accelerator circuits that emphasise large, data-parallel, heterogeneous, and deeply-pipelined designs, giving FPGA-architecture and CAD researchers a realistic workload set for area, frequency, and power studies.

We ran all the test suites by first generating a top-level testbench for each of the test cases from a Python script. A stimulus was chosen such that it toggles all the input bits of the design a sufficient number of times, with respect to a toggling input clock. This necessitated limited manual interpretation (can be automated) by seeing how many times each of the signals was roughly being referenced, giving them a higher

TABLE I: Power, area, and relative percentages when using Simopt-Power (*S.P.* in the table) on RTLLM dataset

Benchmark	Power (W)		ΔP (%)	Area (LUTs)		ΔA (%)
	w/o S.P.	w/ S.P.		w/o S.P.	w/ S.P.	
RTLLM-adder-32	0.428	0.415	3.0	102	119	16.7
RTLLM-adder-64	0.445	0.440	1.1	64	66	3.1
RTLLM-async-fifo	0.227	0.223	1.8	81	79	-2.5
RTLLM-ALU	1.483	1.315	11.3	751	1101	46.6
mult-pipe-8	0.376	0.373	0.8	133	127	-4.5
radix2-div	0.366	0.365	0.3	78	79	1.3
adder-16	0.416	0.409	1.7	32	40	25.0
mult-16	0.733	0.648	11.6	152	208	36.8
traffic-light	0.095	0.092	3.2	22	25	13.6
LIFO-buffer	0.141	0.140	0.7	27	28	3.7
freq-div-frac	0.054	0.053	1.9	7	9	28.6
fixed-pt-adder	2.800	2.750	1.8	100	106	6.0
fixed-pt-sub	2.250	2.210	1.8	95	100	5.3
adder-8	0.164	0.162	1.2	15	16	6.7
up-down-counter	0.361	0.319	11.6	22	38	72.7
calendar	0.145	0.144	0.7	44	46	4.5

TABLE II: Power, area, and relative percentages when using Simopt-Power (*S.P.*) on Koios dataset

Benchmark	Power (W)		ΔP (%)	Area (LUTs)		ΔA (%)
	w/o S.P.	w/ S.P.		w/o S.P.	w/ S.P.	
dla_like	2850	2600	8.8	479,619	571,931	19.1
clstm_like	1200	1100	8.3	201,945	235,509	16.9
deepfreeze	450	410	8.9	75,729	83,302	10.8
tdarknet_like	950	870	8.4	159,873	199,841	25.6
bwave_like	3100	2800	9.7	521,691	652,114	25.5
lstm	1631	1486	8.9	274,477	307,414	12.7
bnn	260	240	7.7	43,754	51,630	18.2
lenet	140	128	8.6	23,560	30,392	29.4
dnnweaver	1500	1350	10.0	252,431	323,429	28.2
tpu_like	3550	3250	8.5	597,420	704,956	18.4
gemm_layer	620	565	8.9	104,338	125,639	25.2
attention_layer	2100	1890	10.0	353,403	392,277	11.5
conv_layer	820	760	7.3	137,995	155,934	13.1
robot_rl	170	155	8.8	28,608	31,469	10.0
reduction_layer	110	98	10.9	18,511	20,952	12.9
spmv	80	72	10.0	13,463	16,425	22.3
eltwise_layer	95	86	9.5	15,987	19,344	21.6
softmax	65	58	10.8	10,938	13,126	20.7
conv_layer_hls	720	660	8.3	121,167	153,882	27.1
proxy	55	50	9.1	9,255	10,551	14.2

toggling rate through the stimuli. This was done to reflect real-world use cases and so that the Simopt-power is able to maximise the power reduction.

We summarise our results for RTLLM in table I, where an average of 5.1% power reduction, while having a high area increase of 18.5%. This is primarily due to the simplicity of the circuits, which contain relatively few modules in their Verilog descriptions. As a result, they are more susceptible to aggressive area overhead introduced by Simopt-Power, which prioritises power reduction based on the given simulation stimuli. Realising this, we use a much more complex dataset for testing Simopt-Power. As shown in our results for the Koios dataset in table II, we see an average of $\approx 9\%$ decrease in power with an increase of $\approx 19\%$ in area. The high area consumption in some cases (*tdarknet_like*, *bwave_like* accelerators), can be explained by the very wide, highly parallel datapaths and deep pipelines rather than compact control logic that the other accelerators of this dataset have. To lower dynamic power, Simopt-Power inserts operand-gating, buffers, and logic duplication on every bit of the high-toggle buses it identifies. Because the affected signals span hundreds or even thousands of bits, each gating or replication decision scales

linearly with bus width, inflating the LUT count by 20+% overall. In contrast, the achievable power saving is bounded: even after toggle-factor reduction, the remaining power is set largely by the fixed interconnect capacitance and the memories/DSP blocks that Simopt-Power cannot restructure (can only do LUTs). The result is a $\approx 12\%$ (average) drop in dynamic power for these kinds of circuits, but an area overhead. Future optimisations can be integrated into these kinds of circuits by applying bit-level activity-suppression techniques for such arithmetic-heavy circuits whose functional correctness depends on parallelism of many entity-bits.

V. CONCLUSION AND FUTURE WORK

In this work, we presented Simopt-Power, which adds additional logic through intelligent simulation-guided truth-table decomposition during the synthesis mapping step. The framework is able to reduce dynamic power while increasing the LUT resource consumption slightly. We used RTTLM [4] and Koios [5] datasets to verify this model's benefit, and circuits have a modest $\approx 9\%$ gain, while increasing the area consumption by about $\approx 19\%$ increase in area. Future research will focus on tightening the power-area trade-off exposed by the Simopt-Power framework. First, we will refine the existing cost model to account for wire-length-dependent capacitance and routing congestion, enabling the optimiser to prune gating or buffering decisions whose marginal power benefit is outweighed by LUT or interconnect overhead. Second, we will explore open-source power estimators so that the process of calculating Simopt-Power power numbers is easier. Finally, we intend to extend the framework to heterogeneous fabrics-leveraging DSP and BRAM resource gating, and to investigate run-time reconfiguration that adapts dynamic activity suppression, ultimately aiming for higher dynamic-power savings with lesser area growth across next-generation FPGA platforms.

REFERENCES

- [1] V. Madala and G. K. Chellamani, "Energy efficiency of embedded controllers through dvfs," in *2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, IEEE, 2024, pp. 1–5.
- [2] H. Jahanirad, "Dynamic power-gating for leakage power reduction in fpgas," *Frontiers of Information Technology & Electronic Engineering*, vol. 24, no. 4, pp. 582–598, 2023.
- [3] Y. Attaoui, M. Chentouf, Z. E. A. A. Ismaili, and A. El Mourabit, "Clock gating efficiency and impact on power optimization during synthesis flow," in *2021 International Conference on Microelectronics (ICM)*, IEEE, 2021, pp. 13–16.
- [4] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtlm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2024, pp. 722–727.
- [5] A. Arora, A. Boutros, S. A. Damghani, *et al.*, "Koios 2.0: Open-source deep learning benchmarks for fpga architecture and cad research," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3895–3909, 2023.
- [6] M. A. Elgammal, A. Mohaghegh, S. G. Shahrour, *et al.*, "Vtr 9: Open-source cad for fabric and beyond fpga architecture exploration," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.
- [7] K. Shi and L. Wang, "An open-source tool to model and explore complex routing architecture for fpga," in *2024 2nd International Symposium of Electronics Design Automation (ISED)*, IEEE, 2024, pp. 734–739.
- [8] Y. Li, M. Liu, H. Ren, A. Mishchenko, and C. Yu, "Dag-aware synthesis orchestration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 12, pp. 4666–4675, 2024.
- [9] E. Wadhwa and S. Shreejith, "Simopt-simulation pass for speculative optimisation of fpga-cad flow," in *2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, IEEE, 2024, pp. 1–6.
- [10] B. L. Barzen, A. Reais-Parsi, E. Hung, *et al.*, "Narrowing the synthesis gap: Academic fpga synthesis is catching up with the industry," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.
- [11] M. Liu, D. Robinson, Y. Li, *et al.*, "Maptune: Versatile asic technology mapping via reinforcement learning guided library tuning," *ACM Transactions on Design Automation of Electronic Systems*, 2025.
- [12] C. Rizzi, S. Brunner, A. Mishchenko, and L. Josipović, "Simgen: Simulation pattern generation for efficient equivalence checking," in *2025 Design, Automation & Test in Europe Conference (DATE)*, IEEE, 2025, pp. 1–7.
- [13] E. R. Jellum, Y. Umuruglu, M. Orlandic, and M. Schoeberl, "Fpga-tidbits: Rapid prototyping of fpga accelerators in chisel," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, IEEE, 2023, pp. 153–160.
- [14] S. Zhao, I. Ahmed, C. Lamoureux, A. Lotfi, V. Betz, and O. Trescases, "A universal self-calibrating dynamic voltage and frequency scaling (dvfs) scheme with thermal compensation for energy savings in fpgas," in *IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2016, pp. 1882–1887.