Auto-OPS: A Framework For Automated Optical Probing Simulation on GDS-II

Paul Flammarion, Sajjad Parvin, Frank Sill Torres, and Rolf Drechsler

***Abstract*—In this work, for the first time, we propose a security evaluation framework, namely Auto-OPS, that automates performing the Optical Probing (OP) attack in simulation on a full GDS-II design file. Auto-OPS empowers designers by automatically extracting the active regions geometry model of each logic cell in the standard cell library or custom-designed logic cells to evaluate the security robustness of a design. Auto-OPS enables scaling up of the current OP evaluation environments which rely on manual extraction of active regions which is an error-prone and cumbersome procedure. Additionally, we evaluated and demonstrated the performance of our framework on several benchmark circuits GDS-II files designed using an open-source 45 nm standard cell library.**

I. Introduction

CHIPS have been shown in the literature to be vulnerable to various types of Side-Channel Attacks (SCA) that can lead to leakage of chips' sensitive information, such as Spectre [1], Meltdown [2], etc. The mentioned SCA resulted in the leaking of sensitive information from the chips. These attacks could have been mitigated in the design phase. Hence, there is a need for security evaluation frameworks to evaluate the robustness of circuits against such attacks pre-silicon to reduce the final cost of secure-end products.

Recently, a powerful non-invasive laser-based SCA called Optical Probing (OP) attack was proposed to extract sensitive information from chips [3], [4]. OP attack utilizes a Near-InfraRed (NIR) laser light to extract the information through the backside of chips. Various mitigation techniques have been proposed in the literature to robustify a design against OP attack in the design phase [5], [6]. These proposed mitigation techniques against OP attack are at the layout level [5], [6], [7] and the state-of-the-art OP evaluation environments [8], [9] only consider a handful of logic cells for performing OP and security evaluation of circuits in simulation. This is due to the fact that the existing OP evaluation environments rely on a manual geometry modeling of the active regions of each logic cell to perform OP in simulation [8], [9]. As a result, whenever designers are required to perform OP in the simulation, they must create a geometry-based model of each logic cell for all possible input combinations. This process can be time-consuming, especially when the layout of a logic cell is modified in the design phase in several design iterations, the designers are required to migrate to a new technology with a new standard logic cell library, or a custom logic cell design is used to design a circuit.

To bridge this gap, we propose an approach to automatically extract the active regions of logic cells from a design's layout file. Auto-OPS is the first framework that allows designers and engineers to extract the active regions of the layout automatically. Consequently, designers have the freedom to choose from any logic cells, or even use custom logic cells to design a layout, and yet be able to automatically evaluate the robustness of their design against OP attack in simulation in a few seconds. Auto-OPS provides a cost and time-effective approach to evaluate the OP robustness of designed chips during design time.

II. Auto-OPS Framework

In this section, we describe the Auto-OPS framework and outline our approach, which automates the process of performing OP in simulation on a large design's GDS-II file. The complete flow of the Auto-OPS framework can be partitioned into six stages and is shown in Fig. 1. Fig. 1 utilizes an INVERTER logic cell to explain the flow of Auto-OPS. We will refer to Fig. 1 in subsequent subsections to explain each stage of Auto-OPS in detail using the used INVERTER logic cell example.

A. Design Entry

After parsing the GDS-II file, Auto-OPS identifies the distinct logic cells used in the design and extracts their relevant geometries. To this end, Auto-OPS filters relevant layers such as metal, vias, diffusion, poly-silicon, pin, and N-well. Additionally, with the aid of the liberty timing file (*.lib) which contains information about the functionality of each logic cell, Auto-OPS determines each cell's functionality, which will be used to extract the active regions of each logic cell's layout based on the applied input value to the logic cell. For example, in the INVERTER cell shown in **Stage #1** of the Fig. 1, only diffusion, poly-silicon, metal, and pin layers (**A**: input, **ZN**: output, **Vdd**: power, **Vss**: ground) are extracted. These layers help define the cell's active regions. **Stage #1** has a complexity of $\mathcal{O}(n)$ as it parses the logic cells once.

B. Model Initialization

For each logic cell, each layout's geometric region is converted to a node representation that is defined similarly

P. Flammarion was with the Cyber-Physical Systems group at the German Research Centre for Artificial Intelligence (DFKI GmbH), Bremen, Germany (e-mail: paul.flammarion@eleve.isep.fr).

S. Parvin is with the Institute of Computer Science, University of Bremen, 28359 Bremen, Germany (e-mail: parvin@uni-bremen.de).

F.S. Torres is with the Institute for the Protection of Maritime Infrastructures, German Aerospace Center, Bremerhaven, Germany (e-mail: frank.silltorres@dlr.de)

R. Drechsler is with the Institute of Computer Science, University of Bremen, 28359 Bremen, Germany, and also with the Department of Cyber-Physical Systems, DFKI GmbH, Bremen, Germany (drechsler@uni-bremen.de).

The work described in this paper has been supported by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation) under the priority programme SPP 2253 – 439918011 in project DR 287/38-1 and SPP 2253 - 535696594 in project DR 287/43-1.

Manuscript received XX XX, XXXX; revised XX XX, XXXX.

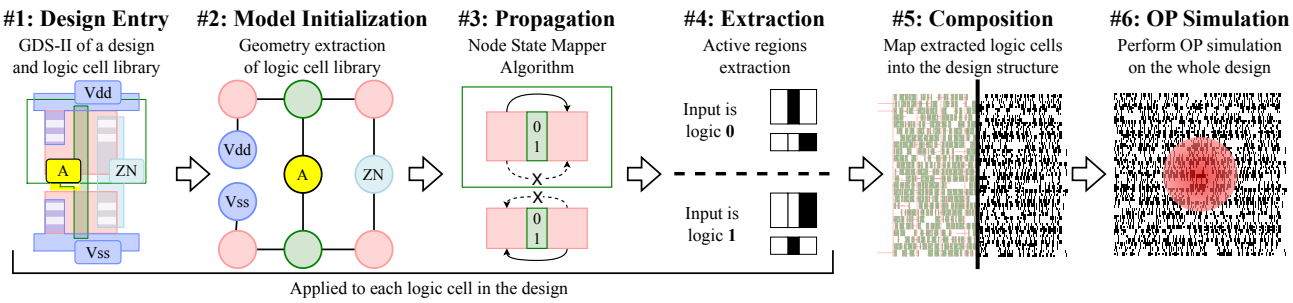


Fig. 1: Flow of Auto-OPS

to the connection at the transistor-level schematic as shown in **Stage #2** of Fig. 1. The coordinates, layers, and links to other nodes define the necessary information of the extracted geometry model to later compute the active regions from the logic cell's layout. For example, if the information between the via and metal layers overlap, the algorithm creates a connection between those two nodes as a link.

After processing all shapes and geometries, Auto-OPS performs three steps across the generated nodes to reconstruct the circuit structure from the layout. The first step is to locate and initialize the diffusion layers and define them as nodes. The second step consists of locating the overlapping coordinates between the poly-silicon and diffusion layers and defining them as poly-silicon nodes. The final step is to create edges between diffusion, poly-silicon, and generated pin nodes that are each connected as in the transistor-level schematic of the layout. Automating this step prevents errors that were previously introduced by manually determining active regions [8]. The complexity of **Stage #2** is $\mathcal{O}(n)$. This is due to the one-time identification of all required layers, such as the diffusion layer, via layer, and poly-silicon layer, from each logic cell's layout.

For example, in the Fig. 1, the **Stage #2** of the Auto-OPS's flow shows the extraction of each processed node of an INVERTER logic cell's layout. All these nodes that are defined as either diffusion or poly-silicon types in Auto-OPS are represented by the color green and red in the **Stage #2** of the Fig. 1, respectively. Green (poly-silicon layer) and red (diffusion layer) colored nodes indicate transistor regions (Source, Gate, and Drain terminal) [10] that can contribute to the reflection of light under OP based on the present voltage at these terminals. After the **Stage #2** of the flow, we reconstruct circuit information from the layout using a graph representation and then all nodes are ready to propagate the applied input voltage to various regions of the logic cell's layout in the next step.

C. Propagation

Determining the propagation of the voltages based on the applied input pattern through the processed nodes which is shown in Fig. 1 is elaborated in Algorithm 1. The algorithm focuses on the propagation of the voltages through all the nodes within a logic cell that can contribute to light reflection, namely diffusion and poly-silicon types nodes. Algorithm 1 utilizes the internal state and graph structure, as shown in the

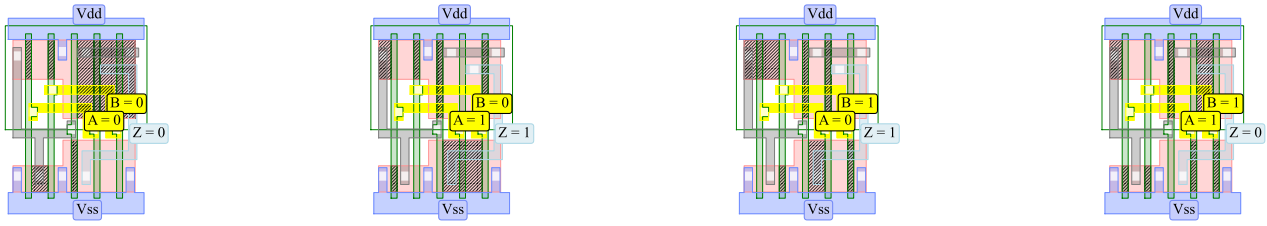
Algorithm 1: Node State Mapper Algorithm

Input : List of relevant diffusion type nodes
Output: Stateful representation of nodes for extraction

forall *node* ***in*** $list_{nodes}$ **do**
  if *node has a propagated voltage* **then**
    $node.voltage \leftarrow propagated.voltage$
  else
    $node.voltage \leftarrow neighborAlgorithm(node)$
  end
end
Function *neighborAlgorithm*$(node)$
  if $left_{poly\text{-}silicon}$ *propagates* **then**
    $node.voltage \leftarrow left_{diffusion}.voltage$
  else if $right_{poly\text{-}silicon}$ *propagates* **then**
    $node.voltage \leftarrow right_{diffusion}.voltage$
  else
    $node.voltage = None$
  end
  return $node.voltage$
end Function

Stage #2 of Fig. 1 to propagate the input voltages. For the INVERTER logic cell shown in Fig. 1, there are various possible connections between nodes that could contribute to the voltage propagation across the logic cell's nodes representation based on the applied input values. When an edge connects two nodes and one of them has a known voltage, such as *Vss*, *Vdd*, or inputs, the voltage is propagated to the other connected node.

The propagation behavior changes based on the voltage present at a node of poly-silicon type. This is based on MOSFET's operating principle, where it conducts depending on the voltage present at the gate terminal. The poly-silicon node is the MOSFET's gate terminal in the node-based representation of Auto-OPS flow. The poly-silicon node conducts the voltage only when the applied input voltage to the gate terminal results in $|V_{gate} - V_{body}| > 0$ with respect to the MOSFET's body voltage. Otherwise, it does not conduct the voltage from the neighboring node. The neighborAlgorithm function in Algorithm 1, is assigning the neighboring diffusion node voltage to the current diffusion node when the respective poly-silicon is conducting. Moreover, if a diffusion node is trapped between two non-conducting polysilicon nodes due to the applied input combination, the diffusion node will not contribute to the light reflection. After the execution of Algorithm 1, all nodes that contribute to the light reflection based on the applied input combination are determined and prepared for the next stage of Auto-OPS's flow. The complexity of **Stage #3** is $\mathcal{O}(m \times n)$ where, m, and n are initial zones that can be active, and the number of connected elements to the possible active node,



(a) (b) (c) (d)

Fig. 2: Active regions extraction of a standard CMOS XOR2 logic cell (a) Input A is logic 0 and input B is logic 0, (b) Input A is logic 1 and input B is logic 0, (c) Input A is logic 0 and input B is logic 1, (d) Input A is logic 1 and input B is logic 1.

respectively.

D. Extraction

With the known voltage state of each node, the extraction of active regions is performed. It is important to consider that NMOS and PMOS transistors behave differently for the extraction of the active region. According to [5], PMOS and NMOS transistors reflect light under OP when the voltage present at the diffusion or poly-silicon nodes are set to 0 and 1, respectively. Moreover, according to [5], the reflection amplification of NMOS, and PMOS transistors are different ($K_{PMOS} = 1.3K_{NMOS}$). Consider the INVERTER logic cell's active regions, as shown in the **Stage #4** of the Fig. 1. When the applied input is set to logic 0, active regions of the PMOS and NMOS are only the diffusion region overlapped by poly-silicon and the diffusion region connected to the output signal, respectively. On the other hand, for the INVERTER logic cell, when the applied input is set to logic 1, the active regions of the PMOS and NMOS are only the diffusion region connected to the output signal, and the diffusion region overlapped by poly-silicon, respectively. This is due to having $|\Delta V| > 0$ for these regions with respect to their body voltage based on the applied input.

To describe Auto-OPS's flow up until the **Stage #4** using a more complex logic cell, we showcase active regions extraction on a XOR2 logic cell, as shown in the Fig. 2. We demonstrate the extraction of the active region on all possible input combinations of the XOR2 logic cell. As shown in Fig. 2, the hachured zones overlay on the XOR2 logic cell's layout represent the active regions of an XOR2 logic cell. The approach discussed through the **Stages #1-4** in the flow of Auto-OPS in the Fig. 1 is applied to all the distinct logic cells present in the GDS-II file of a large design. The complexity of **Stage #4** for each logic cell is given as $\mathcal{O}(2^{k+i} \times n \times m)$, where k is the maximum number of inputs for the logic cell, i is the maximum number of outputs, n is the maximum number of possibly active regions, and m is the maximum number of elements connected to that zone. It must be noted, that we do perform active region extraction on standard logic cells, and they have a limited number of maximum input pins. Hence, the upper bound for the complexity of active region extraction is limited to the logic cell's number of input pins.

E. Composition

A GDS-II file of a large design is composed of multiple logic cells from the cell library. In the first 4 stages of the Auto-OPS's flow, we extracted the active regions of each distinct logic cell in the large design. Auto-OPS extracts the die area definition and information on logic cell placement and orientation from the GDS-II file. As the next stage, based on the applied input to the design, Auto-OPS places the active regions geometry model of each logic cell which is computed in the previous stages according to the coordinates defined in the design file, as shown in the stage #5 in the Fig. 1. In this stage, by composing the information from the GDS-II design file with the extracted logic cells' active regions geometry model, Auto-OPS creates a new representation of the GDS-II file. As a result, the composition stage of Auto-OPS computes an OP simulation-compatible representation of the GDS-II file. The complexity of **Stage #5** of our framework is $\mathcal{O}(n)$, as it runs through all the used cells in the design once.

F. OP Simulation

The final stage in the Auto-OPS flow is to perform OP simulation on the entire design. This is done by placing the center of the laser on any location of the OP composition representation of the GDS-II file to perform Electro-Optical Probing (EOP), and Electro-Optical Frequency Mapping (EOFM) analysis according to the equation described in [8]. Interested readers are encouraged to refer to [11], to learn more about performing OP simulation on the geometry-based representation of the GDS-II file.

III. Experimental Results & Discussion

In this section, we evaluate and discuss the performance of the Auto-OPS in terms of scalability and the required processing time of our framework on various benchmark circuits. For this end, we used benchmark circuits from ISCAS'85 [12], ICCAD Contest 2021 [13], and EPFL benchmarks [14]. All the benchmark circuits were passed through the *Cadence Genus* synthesis tool using a 45 nm open-source standard cell library [15]. Next, all the synthesized benchmark circuits were passed through *Cadence Innovus* for place and route using an area density of 65 %. Finally, the placed and routed GDS-II design files were streamed out to be used in the Auto-OPS framework.

The performance results of the Auto-OPS on the aforementioned benchmarks are shown in Table I. In Table I, PI represents the number of the circuit's primary input, PO is the number of the circuit's primary output, DC is the number of

TABLE I: Performance evaluation of Auto-OPS.

	Circuits	PI	PO	DC	TC	DA	ET
ISCAS'85 [12]	c17	5	2	1	6	10.05	3.03
	c432	36	7	21	79	120.87	5.20
	c499	41	32	9	171	400.38	3.29
	c1355	41	32	9	172	341.17	3.86
	c1908	33	25	19	204	357.26	3.97
	c2670	233	140	23	268	459.75	5.40
	c3540	50	22	24	465	755.87	5.48
	c7552	207	108	23	716	1223.07	5.35
ICCAD Contest [13]	test01	12	4	3	9	41.34	3.21
	test02	12	4	8	35	62.56	3.36
	test03	57	20	10	114	305.77	3.31
	test04	80	68	23	1540	4422.60	5.32
	test05	64	34	13	339	1158.70	3.56
	test06	96	64	15	1835	4169.58	3.85
	test07	50	32	22	605	1484.97	5.18
	test08	108	64	25	938	1805.42	5.45
	test09	96	41	27	1486	4230.36	5.95
	test10	48	64	23	710	1724.32	5.17
	test11	24	40	22	616	1291.56	4.19
	test12	264	26	20	3629	9406.08	4.29
	test13	48	5	20	286	577.67	4.86
	test14	16	1	13	36	48.23	3.98
	test15	152	97	26	1118	2554.48	5.47
	test16	139	4	23	345	483.53	5.08
	test17	95	68	12	186	577.67	3.70
	test18	20	17	9	42	72.78	3.41
	test19	96	285	25	441	809.89	4.39
	test20	121	17	21	1730	3597.60	4.36
EPFL [14]	adder	256	129	2	128	1481.73	3.17
	bar	135	128	5	917	1805.42	3.25
	div	128	128	39	20 670	52 893.09	7.36
	hyp	256	128	43	129 673	205 216.87	15.07
	log2	32	32	45	14 019	32 574.49	7.32
	max	512	130	22	1599	2450.82	5.69
	multiplier	128	128	28	10 176	20 170.01	5.15
	sin	24	25	36	2757	4497.77	6.62
	sqrt	128	64	37	13 653	18 493.28	8.03
	square	64	128	28	9400	16 904.03	4.92

distinct logic cells in a design, TC is the total number of logic cells in a design, DA design area (μm^2), and ET is Auto-OPS's execution time on a design (s). The Auto-OPS performance evaluation was performed on an Apple M1 Pro machine with an 8-core CPU and 8-core GPU with an operating frequency of 2.06-3.22 GHz and 8 GB of memory.

According to Table I, Auto-OPS can automatically prepare a GDS-II file of a design for performing OP in simulation in a few seconds. Even for the most complex case study (*hyp* benchmark circuit from [14]), which contains 129673 logic cells, it only takes 15.07 s to generate the OP active regions composition representation of the GDS-II file. Our evaluation results demonstrate the applicability and ease of integration of Auto-OPS into a designer's flow for designing OP secure Integrated Circuits (ICs). Moreover, it must be noted that the active regions' geometry modeling of logic cells has been validated in literature [9]. In this work, we scaled up the active regions geometry extraction of a standard cell library to create an OP representation of the GDS-II file to ease the process of performing OP security evaluation on a large design using any logic cell design. Auto-OPS omits the need for manual extraction of active region geometry of a GDS-II which is laborious, and impractical for a real design to evaluate the security robustness of the design against OP pre-silicon [8].

It must be noted that commercial extraction tools such as *Calibre* [16] are capable of extracting transistor-level information, i.e., spice netlist, from a GDS-II file. However, spice simulation must be performed to identify the active regions of a logic cell based on the applied input pattern. Spice simulation can be time-consuming for each input pattern for a large design and adds another step to active region identification, which is error-prone. Hence, Auto-OPS speeds up the process of identifying active regions of each logic cell in a large design GDS-II file to perform OP by omitting the need for a spice simulator. Furthermore, the applicability and limitations of the OP simulation are discussed in depth in [11].

IV. Conclusion

We presented the first framework to automate OP simulations for an entire layout using an open-source 45 nm standard cell library. Auto-OPS was evaluated on several benchmark circuits to demonstrate its scalability on large design files. Auto-OPS automates the extraction of transistors' active regions and prepares the GDS-II file for OP simulation in seconds which can easily be integrated into the ASIC design flow. This provides a cost-effective solution for evaluating the security robustness of a design against OP during pre-silicon. Designers can use any logic cell library or custom cells to assess the security robustness of their designs against OP attack with minimal effort. Future work will focus on geometry model extraction for an extension of OP attack, namely LLSI analysis, and improving our geometry extraction model further using real OP experimental setup on our fabricated chip using a commercial 28 nm node.

References

[1] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *Communications of the ACM*, 2020.

[2] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *Communications of the ACM*, 2020.

[3] M. T. Rahman *et al.*, "The key is left under the mat: On the inappropriate security assumption of logic locking schemes," in *IEEE HOST*, 2020.

[4] S. Tajik *et al.*, "On the power of optical contactless probing: Attacking bitstream encryption of fpgas," in *ACM CCS*, 2017.

[5] S. Parvin *et al.*, "Toward optical probing resistant circuits: A comparison of logic styles and circuit design techniques," in *ASP-DAC*, 2022.

[6] ——, "Hidden in plain sight: A detailed investigation of selectively increasing local density to camouflage and robustify against optical probing attacks," in *ITC India*, 2023.

[7] ——, "Lo-RISK: Design of a low optical leakage and high performance risc-v core," in *IEEE COINS*, 2023.

[8] ——, "FELOPi: A framework for simulation and evaluation of post-layout file against optical probing," in *DATE*, 2023.

[9] V. K. Ravikumar *et al.*, "Understanding spatial resolution of laser voltage imaging," *Elsevier MR*, 2018.

[10] N. H. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson, 2011.

[11] S. Parvin *et al.*, "OPTI-Sim: Performing optical probing simulation on layout design files," *IEEE TCAD*, 2024.

[12] F. Brglez, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran," in *IEEE ISCAS*, 1985.

[13] I. Contest. (2022) Problem a. [Online]. Available: http://iccad-contest.org/Problems.html

[14] L. Amarú *et al.*, "The EPFL combinational benchmark suite," in *IWLS*, 2015.

[15] J. E. Stine *et al.*, "Freepdk: An open-source variation-aware design kit," in *IEEE MSE*, 2007.

[16] Siemens Digital Industries Software. (2024) Calibre design solutions. https://eda.sw.siemens.com/en-US/ic/calibre-design/. Accessed: 2024-06-28.