CIRCUIT OBFUSCATION AND LOW-OVERHEAD SECURITY STRATEGIES IN FORMALLY DEFINED SYSTEM-ON-CHIPS

Ckristian Ricardo Esteban Duran Blanco

Trabajo de grado presentado para optar el título de Doctor en Ingeniería, área Ingeniería Electrónica

Director

Elkim Felipe Roa Fuentes

Ingeniero Electrónico, PhD

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2022

Dedicated to

This work is dedicated to humanity.

Acknoledgements

I want to thank many people who have been present with me during the development of this doctorate.

First and foremost, I am thankful to Professor Elkim Roa, who has seen the opportunity in me since my master's. Thanks to his actions, I have developed my skills in different fields like language, electronics, scientific research, and professional growth.

Next, I want to thank Professor Cong-Kha Pham for his support during the research stay at the University of Electro-Communications. I have developed research in processor security and low-power circuits and connected with very different cultures like the Vietnamese and the Japanese.

I am deeply grateful to the research lab Onchip, which I had the pleasure of working with, specially to Hector Gomez, that introduced me to the laboratory and to my director in the first place. Thanks to Andres Amaya, Javier Ardila, Luis Rueda, Juan Moya, and Christian Torres for introducing me proper to the world of analog design and allowing me to integrate their designs into chips. Hanssel Morales for all his co-work in digital designs, processor design, verification schemes, and programming tasks. Ronaldo Serrano, which has brought me support on the long nights. Juan Pablo Romero and Wilmer Ramirez for assisting in digital circuits.

I want to say thanks to the professors of the committee. Professor Conf-Kha Pham, who gave me perspectives on integrated circuits. Professor Carlos that have been supporting me with the process of graduation. Professor Rodolfo Villamizar realized a full-body review of the dissertation. Professor Fredy Segura and Johan Sebastian Eslava, for all the feedback.

Lastly, I want to thank my family very sincerely. My mother, Maria Edid, who has taught me about different things in life and given me her best love and emotional support. With his hard-working mentality, my father, Ricardo Duran, has been my inspiration for not giving up. And my brother, Elian Duran, which I admire profoundly due to his way of dealing with adversities in a totally different fashion than mine.

Contents

List of Figures

List of Tables

Abstract

Title: CIRCUIT OBFUSCATION AND LOW-OVERHEAD SECURITY STRATEGIES IN FORMALLY DEFINED SYSTEM-ON-CHIPS [*]

Author: Ckristian Ricardo Esteban Duran Blanco [**]

Keywords: Circuit obfuscation, standard cells, embedded security, AES, SAT, formal verification, functional verification.

Description: Different consumer electronic circuits are susceptible to different kinds of attacks which can compromise the authentication and safety of data. For digital circuitry and mixed-signal systems, standard cells are used to integrate the system with electronic design automation (EDA). Placement and routing of standard cells output the final layout of a system but can be attacked with decapping and imaging tools due to its hardware visibility. Such cells can be generated multiple times to perform obfuscation of circuits to mitigate the visibility and extraction. Another layer of security is intrinsic inside of the system with cryptographic accelerators and memory safety. Here we show two different placement algorithms for standard cell generation, a circuit obfuscation procedure using the previous standard cells, system-level cryptography and memory protection, and chip generation with formal and functional verification. We modify the placement algorithm of standard cells to constrain different solutions to generate several different layouts to be applied in circuit obfuscation. Furthermore, the system implements cryptographic accelerators for authentication and security purposes. This work presents also a oblivious obfuscator for memory with low-overhead in area and timing. The system is output by a chip generator, which can output RTL code, perform padring generation, and an always-on domain for low-power management. The SoC can be integrated easily in a VLSI flow. According to the processor's specifications, the generated circuit is formally and functionally verified with several constraints for assuming and asserting conditions.

---

[*] Doctoral Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Elkim Felipe Roa Fuentes, Electronics Engineer, PhD.

Resumen

Título: OFUSCACIÓN DE CIRCUITOS Y ESTRATEGIAS DE BAJO SOBRECOSTO EN SYSTEMAS-EN-CHIP FORMALMENTE DEFINIDOS [*]

Autor: Ckristian Ricardo Esteban Duran Blanco [**]

Palabras Clave: Obfuscación de circuitos, celdas estándar, seguridad embebida, AES, SAT, verificación formal, verificación funcional.

Descripción: Los diferentes circuitos electrónicos de consumo son susceptibles a diferentes tipos de ataques que pueden comprometer la autenticación y seguridad de los datos. Para circuitos digitales y sistemas de señal mixta, se utilizan celdas estándar para integrar el sistema con automatización de diseño electrónico (EDA). La ubicación y el enrutamiento de celdas estándar dan como resultado el diseño final de un sistema, pero pueden ser atacados con herramientas de destapado y generación de imágenes debido a la visibilidad de su hardware. Dichas celdas se pueden generar varias veces para realizar la ofuscación de los circuitos para mitigar la visibilidad y la extracción. Otra capa de seguridad es intrínseca dentro del sistema con aceleradores criptográficos y seguridad de la memoria. Aquí mostramos dos algoritmos de colocación diferentes para la generación de celdas estándar, un procedimiento de ofuscación de circuitos usando las celdas estándar anteriores, criptografía a nivel de sistema y protección de memoria, y generación de chips con verificación formal y funcional. Modificamos el algoritmo de ubicación de las celdas estándar para restringir diferentes soluciones para generar varios diseños diferentes que se aplicarán en la ofuscación del circuito.Además, el sistema implementa aceleradores criptográficos con fines de autenticación y seguridad. Este trabajo presenta también un ofuscador ajeno a la memoria con poca sobrecarga en área y tiempo. El sistema es generado por un generador de chips, que puede generar código RTL, generar padring y un dominio siempre activo para la gestión de bajo consumo. El SoC se puede integrar fácilmente en un flujo VLSI.

---

[*] Tesis Doctoral

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Elkim Felipe Roa Fuentes, Ingeniero Electrónico, PhD.

De acuerdo con las especificaciones del procesador, el circuito generado se verifica formal y funcionalmente con varias restricciones para asumir y afirmar condiciones.

Introduction

The rapid increase of consumer electronics has affected the lifestyle of people for generations. More and more electronics are introduced into daily lives for solving problems in several fields. Namely, in the information field, it is widespread for people to manipulate, store, send, and receive data in several forms, such as text, photos, videos, IDs, and passwords for authentication, among others. In the world, as of April of 2022, for every second, around 150000 GB of information traffic circles the internet which are divided roughly into 3 million e-mails sent, 100000 google searches, 90000 videos watched, and 10000 published Real Time Statistics Project (2022).

Information is handled in critical devices such as terminals and servers that people use every time. Consumers and corporate, government, and military data are stored in devices that may be accessible through a communication channel. More recently, information management like storing and communication require several layers of implementation in such critical devices for security and reliability. Interactions on such devices require authentication and validation carried through communications protocols containing cryptography and protocol handshaking. Some examples of these interactions are connecting to the internet through website or phone applications, wireless card authentication in trains and buses, biometric confirmation, and online payments.

Devices carrying the data are implemented using semiconductors. Nikolic *et. al* Nikolic et al. (2018) discussed the semiconductor industry evolution and evaluated some challenges in a survey. Regarding semiconductor demand, the first wave came with creating mainframes,

scientific computing, and data processing. The second and third waves came with creating personal computers, the internet, and smartphone apps. This evolution took a time of approximately 40 years. A further challenge approaches with the newer applications such as healthcare, telemedicine, education, autonomous driving, gaming, entertainment, visualization, and the cloud. According to the survey, the challenge comes with the costs associated with software development, verification, validation, and engineering effort related to design and testing.

The previously mentioned survey addresses the challenge of circuit generation, where is described a perspective of technology scaling and automated hardware generation. Some of the hardware generators feature processors, buses, peripherals, and analog designs Chang et al. (2018); RISC-V Foundation (2019a). These hardware generators include several extensions to connect different standard interfaces such as DDR and flash memories, high-speed serial links such as USB and SATA, and DSP functions. The generated circuit can be simulated using FPGA acceleration, where it can be verified functionally. With the scalability of these generators, the implemented silicon can contain designs that can fulfill the requirements for handling data for consumer electronics.

With these improvements in consumer electronics, circuit designs like processors, buses, and digital peripherals existent in the market, maybe vulnerable to attacks. Circuits that manage protocols like USB, DDR, and SATA, among others, can be compromised about the data stored in the multiple levels of memory. Several possible exploits can arise at different security levels, such as illegal access to memory, unauthorized code execution, and communication or side-channel capture. Implementation weaknesses can exploit systems due to the exposition of silicon. An attacker can access memory by exploiting it to obtain stored critical data by

extracting the system circuit behavior. Circuit extraction can be accomplished using reverse engineering with decapping and imaging techniques.

This document will address some of the challenges to overcoming these exploits in security. Notably, we will address the problems related to memory access, imaging reversing and circuit extraction, channel capture, and code injection. This document describes solutions to such security problems around layout obfuscation, cryptography peripherals, formal and functional verification, and memory obfuscation. The perspective of the solutions is wrapped in a chip generation process which creates the systems-on-a-chip to be protected.

Achilles heel of chip security

The main problem for chip security is the rapid growth of consumer electronics, making security a major concern in different fabrication, implementation, and utilization points. To further illustrate the problem, we will detail some of the security problems in the process of design and implementation of any system. Figure 1 presents several stages of design and implementation of a system-on-chip. The system is designed in the generation flow stage with the processor cores, buses, and peripherals. Security peripherals can be included to accelerate the execution of encryption algorithms such as AES. Some of the problems concerning security arise in this stage. RAM access can be possible by corrupting the memory contents Branco and Gueron (2016). If the memory can be accessible, an attacker can inject code into such memories to execute unauthorized programs. Furthermore, using DMAs, some peripherals can execute algorithms in out-of-bounds regions, which an attacker can exploit to capture critical data such as keys.

In the electronic design automation (EDA) stage, some problems can arise due to design

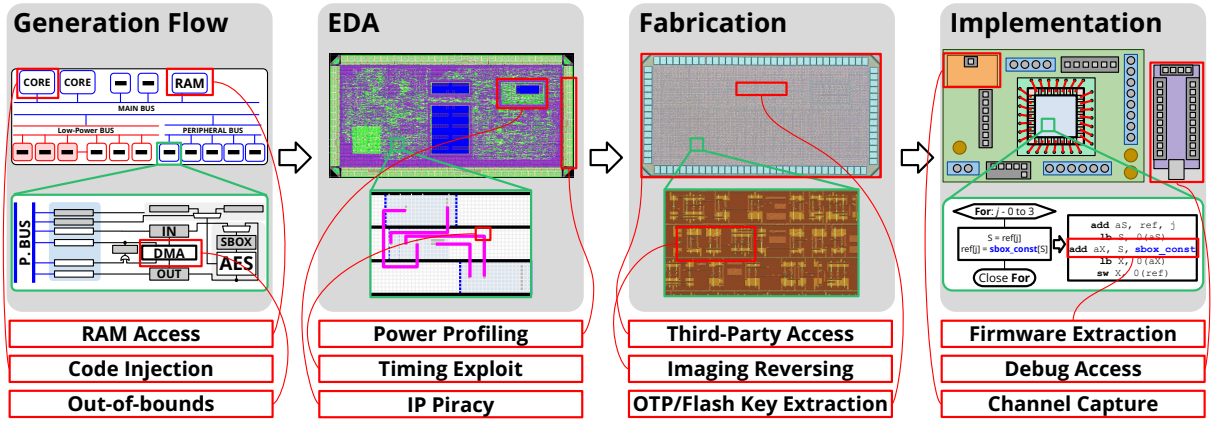

Figure 1. Examples of problems in security within the stages of design and implementation.

constraints applied to the system. For instance, power traces in both voltage and current can be profiled to cause glitches in specific circuits. Such glitches can also cause code injection or memory corruption mentioned before. The glitches can also be caused by executing timing exploits in the circuits by operating with irregular or high-frequency clocks. Intellectual property (IP) piracy can be a threat if critical circuits are not protected in layout or signaling.

In the fabrication stage, the final design is submitted to a technology foundry to manufacture the chips. This submission implies that any submitter needs to rely on third-party entities, which can create particular security concerns. After the chip is manufactured, an attacker can extract the circuit using imaging tools and apply reverse engineering to reveal the behavior of specific critical systems. In the same manner, attackers can extract critical information such as keys stored in non-volatile storage IP such as a one-time programmable (OTP) or flash memory.

Finally, in the implementation stage, external attacks are possible, which compromise the device's behavior. For starters, extracting the firmware can lead to code analysis and reverse engineering of the internals of the final device. If the system contains debug interfaces, an attacker can issue commands to access the memory or inject code into the system. Channel

capture of any signal or power domain is also possible.

Thesis Overview

Numerous scenarios of security flaws can arise for a specific design. This dissertation will detail some of these problems and propose solutions to them. The book focus on layout obfuscation and memory protection applied to formally- and functionally-verified systems that contain security hardware for encryption such as AES. Layout obfuscation uses a standard cell generator whose output can scramble digital cells with multiple cell layouts to mitigate circuit recognition. The included security hardware contains low-overhead primitives protected by the layout obfuscation. Because it is essential to preserve the correct behavior of circuits to avoid simple exploits such as code injections or out-of-bounds access, this work also focuses on functional and formal verification to ensure the system's reliability.

Standard cell generation for layout obfuscation. A first focus of this dissertation is *imaging reversing* previously stated in figure 1. If the design includes layout obfuscation, it can reduce the susceptibility to reverse engineering using imaging. Systems include mostly digital circuits that are integrated using standard cells. A way to achieve layout obfuscation in such systems is being capable of creating and manipulating the internals of standard cells. Aspects such as the layout creation and netlist design can be tuned in ways such as the readability of critical digital circuits becomes difficult Gomez et al. (2019); Rajendran et al. (2013).

Standard cells are essential to perform layout obfuscation of a digital circuit. Standard cells are microcircuits that follow standardized rules to be integrated by EDA algorithms Wang et al. (2009). These circuits perform different logic functions in a digital circuit. A compilation of these cells, named standard cell library, combines several combinational and sequential logic

circuits. Combinational logic includes logic primitives created by N and P transistor pairs. Sequential logic includes memory functions per bit, such as flip-flops and latches. The standard cells are designed with a fixed layout style to be implemented in placement and routing grids Uehara and Vancleemput (1981); Chi Yi Hwang et al. (1993); Sung Mo Kang (1987). Standard cells are firmly attached to the technology node and are usually kept secret in the foundry. Consequently, most of the designs are locked in a technology node, and there are few portable ways to do circuit design reuse between nodes. Because of this limitation, a standard cell generator needs to be used with a conjunction of an obfuscator algorithm to mitigate the hardware visibility of an implemented circuit.

The fundamental step for VLSI using EDA is to generate the standard cell library to be implemented for digital circuit implementation and further obfuscation. Figure 2 shows a standard cell obfuscation procedure involving the placement and routing. The obfuscation inputs the placement of several different netlists according to the different topologies stored for a specific function. In placement, a symbolic placement is solved for each of the netlists that are read. The obfuscation accumulates these solutions, which adds more constraints to the placement solver. This accumulation prevents the generation of a previous layout, which generates the different layouts for obfuscation presented in figure 3. Each of the placement solutions is passed through the routing algorithm. Finally, a geometry translation program places the geometries of obfuscated cells to be usable in a technology node.

Contribution. In this dissertation, several algorithms for placement are proposed to generate standard cells from netlists. The former is a placement algorithm based on pseudo-boolean satisfiability (PB-SAT). The PB-SAT algorithm contains an optimization for

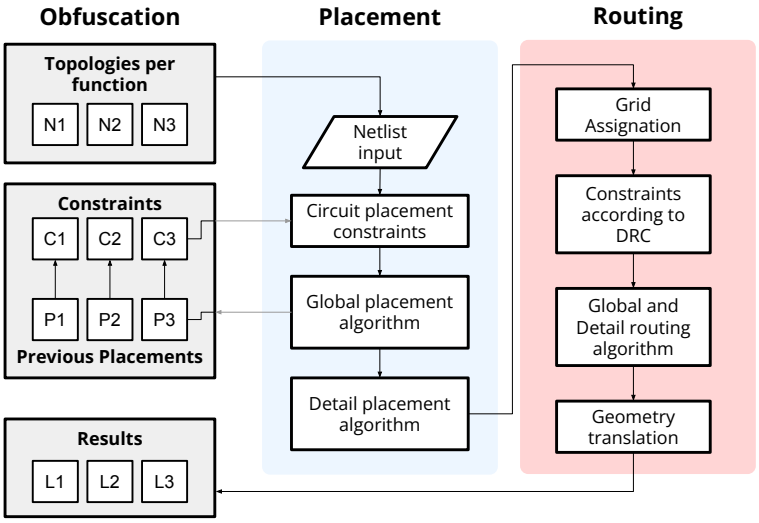

Figure 2. Standard cell obfuscation procedure.

complete routing and pre-layout algorithm awareness. Route awareness is optimized to circumvent routing congestion according to pins location. A graph-based formulation is described later of P-N transistor pairs with a first-depth search algorithm. The transistor pairing uses the previous PB-SAT algorithm as an aided placement. The aided placement provides a better pairing for P-N transistors for the graph's edges, allowing better abutment and less route congestion. The proposed algorithms are implemented in a complete automatic standard cell generation procedure, fulfilling commercial technologies node design rules.

Layout obfuscation. As stated before, layout obfuscation in circuits mitigates the reverse engineering of circuits using imaging tools. Obfuscation of circuits via standard cells is possible by using several layouts of the same function Gomez et al. (2019) or by using similar layouts with different functions Rajendran et al. (2013). Multiple equivalent layouts can be output for a particular layout style in the obfuscated generation of standard cells with a fixed height. Figure 3 presents a model of obfuscation using different layouts of standard cells. Gomez *et. al* Gomez et al. (2019) presents an algorithm for standard cell camouflage modifying

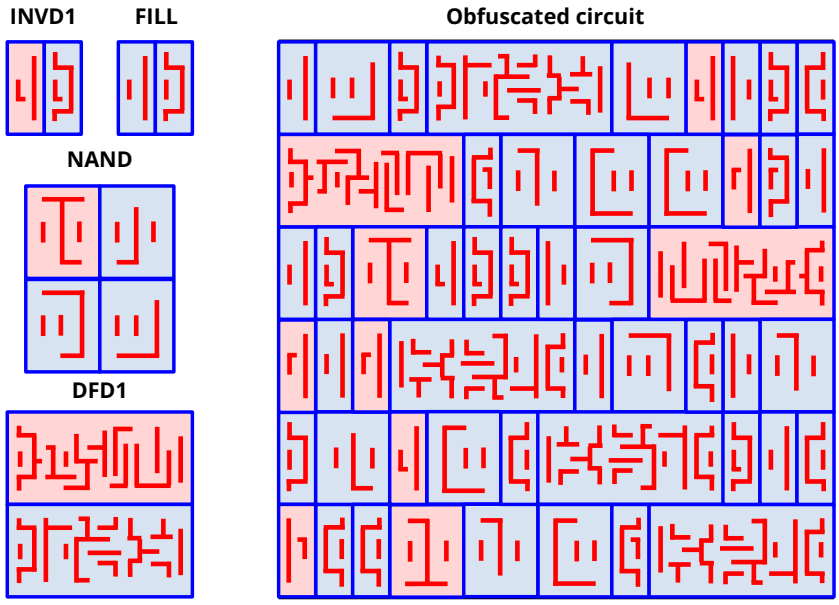

Figure 3. Circuit obfuscation using standard cells.

the digital circuit netlist with alternate cells. The cells provided for camouflage offer several layouts with the same function. The identification of standard cells in imaging software can be mitigated significantly due to assumptions of existing only one type of cell per layout, as indicated in pale red in figure 3.

Contribution. This dissertation also shows the implementation of the standard cell generator into a layout obfuscation technique. The previously mentioned standard cell placement algorithms are modified to output multiple layouts of a single netlist. For each cell function, several layouts are included in different libraries that can be swapped, obfuscating the circuit in layout but preserving the original behavior. The cells are replaced in the digital post-synthesis netlist, which then a place and route EDA tool can route. It is demonstrated that reverse engineering using imaging tools decreases significantly compared to regular standard cells.

Low-energy AES cryptography. Besides the layout obfuscation of the digital circuits, is necessary the modeling of internal security of the device for general purposes. In figure 1 we talked about implementation problems such as *channel capture*, and *firmware extraction*. Security hardware is necessary to mitigate such problems thanks to ciphering the data to send and the program executed. Implementation of security hardware for authentication and cryptography also enhances the safety of information according to execution privileges within a system Hoang et al. (2020); Lee et al. (2020). The cryptography peripherals contain critical primitives that should be secured in layout to avoid reverse engineering. For instance, the advanced encryption standard (AES) is a widespread implementation used for security applications Mathew et al. (2014). In AES, the SBOX can be secured with masking strategies Canright (2005). The SBOX masking is a critical primitive operation that needs to be secured against identification and profiling and also is designed to be low-overhead compared to the system.

A trade-off between power and time needs to be achieved to develop such primitives. The main challenge is to handle memory transportation in the crypto-accelerator. Banerjee *et al.* in Banerjee et al. (2018) introduce a crypto-processor that authenticate nodes in a network. The data transport in this system needs to be pushed and pulled by a processor, which in turn represents a power and energy overhead due to the use of buses, memories, and register communication lines Horowitz (2014). An efficient cryptography accelerator can be implemented in several sub-designs of the system, which potentially enables main memory security without significant overhead.

Contribution. Different approaches for AES-128/256 Sbox acceleration schemes with low overhead are proposed. We designed a Sbox unit to be connected as a logic unit. The

unit was tested as a RISC-V ISA custom instruction in the RISC-V processor to reduce software execution times. This dissertation performs instruction analysis over the SBOX instruction, where it is deduced that execution is concentrated in-memory access most of the time. The unit was included in an AES hardware core. With push-pull from the main processor, the algorithm significantly increases performance over the software version with the custom SBOX instruction. This core was further modified with direct memory access (DMA), which avoids using the main processor to access the data. The solution offers energy savings due to fast execution with low-power overhead for the SoC.

Chip generation. Going back to work in Nikolic et al. (2018), digital implementations of the overall system circuits enable the construction of hardware generators. Automated hardware generators exploit design reuse for digital and analog circuit hardware descriptions and specifications. Cheng *et. al* Chang et al. (2018) presents a framework for the development of process-portable analog and mixed-signal circuit generators. This framework uses code to supply circuit description, specifications, and geometry configurations for automatic layout generation compatible with several process nodes. Layout generation uses layout engines that create DRC-clean layouts from specified scripts in a circuit. Bachrach *et. al* Bachrach et al. (2012) present a hardware description language (HDL) based on the Scala programming language named Chisel. The designer in Chisel can construct a set of libraries that can be reused and configured according to different configurations. This HDL outputs mainly a Verilog register-transfer level HDL synthesizer using modern FPGA or ASIC circuit synthesizers. This behavior allows projects, like the Rocket-chip generator RISC-V Foundation (2019a), to create very-large-scale integration (VLSI) for a hardware-configurable RISC-V processor

with any peripherals.

Contribution. A chip generation is proposed to integrate all the presented systems in this dissertation. The proposed chip generator contains RTL generation, padring generation, and always-on domain creation with a configurable power management unit (PMU). The system can be implemented using FPGA and verified using simulation.

Formal and functional verification. Specifications bind the previous digital implementations in hardware generators. These specifications contain detailed information about the behavior of the algorithms for cryptography primitives and execution schemes and interactions for processors. The generated system should be verified with the specifications to avoid problems such as *code injection* stated in figure 1. The register-transfer level (RTL) implementations of the digital circuits must be able to execute cryptography and processor instructions reliably. The reliability avoids security flaws due to inadequate implementation or design constraints such as protocol handling.

In the case of processors, each instruction needs to be verified against a golden model that follows the instruction set architecture (ISA). A straightforward methodology is to simulate the behavior of the cores. The simulation result is then compared to a golden model to ensure the correct execution of the instructions. Executing tortures with metric optimization can ensure the coverage of most of the test cases Schiavone et al. (2018); Corno et al. (2005). As an alternative, formal verification can set a series of constraints and assertions to verify critical aspects of the execution inside the processor signals Symbiotic EDA (2019); Reid et al. (2016).

Contribution. This dissertation proposes a verification scheme combining two domains, simulation- and formal verification, establishing a methodology for exclusive error de-

tection. Functional verification drives automatic program generation using genetic algorithms to maximize test coverage and the contrast against an instruction set simulator. An interface carries specific processor states according to the ISA specification in the formal verification approach. By combining these two, we present a reliable way to perform more accurate instruction verification by increasing processor state coverage and formal assertions to detect different kinds of errors. Compared to extensive torture test sets, this approach reaches a more significant number of internal states by taking advantage of the exercised abstractions. Among the remarkable results, the proposed approach detected a RISC-V ISA specification gap revealing ambiguity from two different verification perspectives.

Memory obfuscation. Most of the critical storage of an algorithm is stored in memory. Several strategies exist for system-on-chip security, which mostly involve access restrictions Liang et al. (2016); Krishnakumar et al. (2018), or encryption of the memory Yang et al. (2005); Lu et al. (2015b); Awad et al. (2017). However, memory corruption can lead to unauthorized *RAM access*, previously stated in figure 1 Branco and Gueron (2016). Memory obfuscation offers a different approach, where the memory contents are still accessible but are scrambled through the memory space. An oblivious memory obfuscator can secure the information contained in an algorithm that can only be accessed during the execution. The contents are obfuscated once the execution is finished without a straightforward reversing method.

Contribution. This dissertation proposes an SoC with memory obfuscation capabilities for the secure execution of encryption algorithms such as AES. The main memory features an oblivious obfuscator with low overhead. The obfuscator contains a register that holds a random number used to scramble the contents of the memory randomly. This register is

automatically reset when a program exits machine mode from the RISC-V processor included in the SoC. The test function implemented into this obfuscator uses the SBOX from the AES to perform a 1-to-1 translation over a 1KB region of the memory. This approach represents low-overhead in area and timing.

Thesis outline. This document now start to describe in detail each one of the contributions to security in systems-on-a-chip previously stated. The book is organized into five chapters. Chapter 1 presents the two standard cell generators, the first with PB-SAT and the second with graph reversal. Chapter 2 contains the AES encryption schemes in the SoC, as well as the chip generation that contains such SoC with the formal and functional verifications. Chapter 3 contains the memory obfuscation and the layout obfuscation applied to an SBOX unit and the oblivious obfuscator. Chapter 4 concludes this book and states future work.

1. Standard Cell Generation

First, we are going to solve the problem of *imaging reversing* for systems-on-a-chip. The implementation of systems requires primarily digital circuits. These circuits can be reversed and extracted because of imaging tools that can recognize the layouts of individual cells. It was stated that layout obfuscation could be applied to address the problem of *imaging reversing*. The first step to performing layout obfuscation is creating a standard cell generator because digital circuits are composed of cells. If a generator exists, the implemented algorithms can be constrained to different output kinds of layouts that can be used to obfuscate the circuit.

This chapter presents two standard cell generators with different placement algorithms. First, a pseudo-boolean satisfiability (PB-SAT) placement algorithm for double-row standard cells is presented. This placement algorithm is aware of the horizontal routing span to aid a priority-based maze router. The PB-SAT algorithm is then used to aid placement to the second proposed placement, the graph-based implementation. The cells were implemented in 180nm and 200nm technology nodes. These generators are further used for security in chapter 3 as part of layout obfuscators.

1.1. SAT & PB-SAT Standard Cell Generator

In this section, a PB-SAT formulation is proposed for the placement of standard cells. The model is made in similar ways to Iizuka et al. (2006) and Lu et al. (2015a). Models will be discussed for transistor vertical Gate/Diffusion pairing, Horizontal Diffusion Sharing, and Horizontal Wire Length. Solutions of the models explained will be translated to Conjunctive Normal Form (CNF) SAT using PBLib Philipp and Steinke (2015). Optimizations will be

shown using the SAT solutions with iterated algebraic calculations of the target minimization equation. Finally, an adaptable algorithm for solving both placing and routing will be presented.

1.1.1. Introduction. Standard cells provide the physical implementation of digital circuits in large-scale integrations. With the assistance of electronic design automation (EDA) software, digital circuits are synthesized, placed, and routed using standard cells Wang et al. (2009). Layouts for standard cells are generated by applying a series of algorithms: transistor folding, transistor placement, in-cell routing, compaction, and design rule (DRC) violation fixing. A methodology to generate cells solves the placement and routing for each of the cells, storing them in a stick format that translates into technology node geometries de Dood et al. (2019).

Multiple placement algorithms perform horizontal positioning to achieve minimum-width cells. Stochastic algorithms like simulated annealing Guruswamy et al. (1997) or genetic algorithms Lazzari et al. (2007) have been reported as placement solutions whose optimizations are limited by the stop conditions. Micro-cell layout formations for grouping sets of circuitry Sadakane et al. (1995); Gupta et al. (1996) solve the placement and routing problem, but this do not create width-minimum standard cells. On the other hand, deterministic integrations report minimum-width placements by using: graph algorithms Maziasz and Hayes (1991); Bar-Yehuda et al. (1989); Hwang et al. (1990), integer linear programming models (ILP) Cortadella (2013); Gupta and Hayes (1996, 1998); Lu et al. (2015a), and Boolean Satisfiability (SAT) formulations for placement Iizuka et al. (2006) and routing Ryzhenko and Burns (2012).

Previous works perform isolated algorithms during the automatic cell layout generation. Proper algorithms linking are limited by routing Lu et al. (2015a); Cortadella (2013), or DRC

violations fixing Jo et al. (2019); Bykov et al. (2016). Limited linking between placement, routing, and other algorithms restricts the capabilities of optimal layout generation. These capabilities are necessary to support custom cell generation and to smooth algorithm migration for different technology nodes.

Here, an algorithm featuring SAT formulations for placement is proposed with routing optimization and considering global constraints. Although the formulation of dual CMOS cells is described in columns similar to the work in Iizuka et al. (2006), here new optimizations are introduced to improve routing. Transistor vertical gate/diffusion pairing and horizontal wire length were optimized to link the next-step routing algorithm. The proposed algorithm is verified in a full automatic standard cell generator with final generated layouts. By linking the placement with the routing, the final layouts accommodate a 30% less routing congested output than the base SAT formulations. In contrast to related work, complex cells are fully routed using only maze-routing algorithms.

1.1.2. Placement Formulation. The proposed placement is partially based on the work described in Iizuka et al. (2006); Lu et al. (2015a). The input netlist is composed of N-transistors (N), P-transistors (P), and nets (X). Placement is done in a column-based solution, as shown in Figure 4. The initial number of columns of the model is determined by $W = max(N,P)$. Table 1 defines the variables in this formulation. The position of a transistor is determined by $C_{n,p}(i,k)$ that instances the i-th transistor in the respective k column. The flip state is also determined in a transistor through $F_{n,p}(i)$ which enables inverted connections for the diffusions. Transistors placed in the same row side by side must have the same diffusion connection. The additional SAT variables for optimization are: *different gates* ($D_G(k)$),

Table 1
Definition of Variables for the proposed placement SAT formulation

Position Variable	**# Variables**	**# Var. per Transistor**	**Description**	**Based on**
$C_p(i,k)$	$P \times W$	W	P-tran. instantiation	Iizuka et al. (2006)
$C_n(i,k)$	$N \times W$	W	P-tran. instantiation	Iizuka et al. (2006)
$F_p(i)$	P	1	Flip state P-transistor	Iizuka et al. (2006)
$F_n(i)$	N	1	Flip state N-transistor	Iizuka et al. (2006)
Difference Variable	**# Variables**	**# Var. per Column**	**Description**	**Based on**
$D_G(k)$	W	1	Different gates	Iizuka et al. (2006)
$D_D(k,s)$	$W+1$	2	Different diffusions	This place
Presence Variable	**# Variables**	**# Var. per Net**	**Description**	**Based on**
$NetP(i,k,s)$	$(2*W+1) \times X$	$2*W+1$	Net instantiation	This place
$NetLR(i,k,s)$	$(2*W+1) \times X$	$2*W+1$	L to R net inst.	This place
$NetRL(i,k,s)$	$(2*W+1) \times X$	$2*W+1$	R to L net inst.	This place
$Net(i,k,s)$	$(2*W+1) \times X$	$2*W+1$	Net horizontal path	This place

different diffusions ($D_D(k,s)$), and *net presence* ($NetP(j,k,s)$). After some boolean detections ($NetLR(j,k,s)$ & ($NetRL(j,k,s)$)), the *net horizontal path* ($Net(j,k,s)$) is determined which indicates the horizontal span for each net in the placement. The overall horizontal path *Net* with D_G and D_D will be minimized for all the nets to output better transistor placement for a next stage routing algorithm.

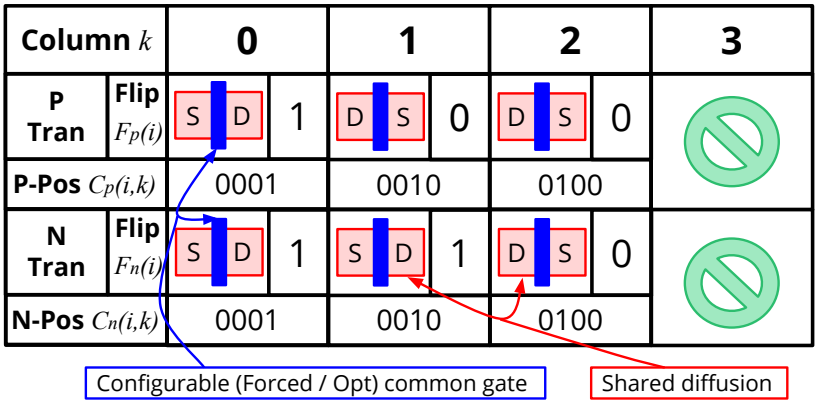

Figure 4. Description of the positioning using SAT variables.

Notice that each i-the transistor have its own set of variables per feature, which is now referred as an object named $stdobj$. Each $stdobj$ have a set of SAT variables for evaluating features inside the model. In this model exists three kinds of $stdobj$ referred as: Transistors, Columns, and Nets. Columns evaluate whenever a vertical gate or vertical diffusion is different. Nets have instantiation spots and horizontal path for length analysis. These variables are a function of i which can represent a Transistor of a Net, k which represent the Columns, and the side s which can be Left, Center or Right. A clear view of this information is exposed in figure 5. All variables are shown in this figure, $D_G(k), D_D(k,s), Net(i,k,s)$, are involved in the optimization, which will be explained later in this section.

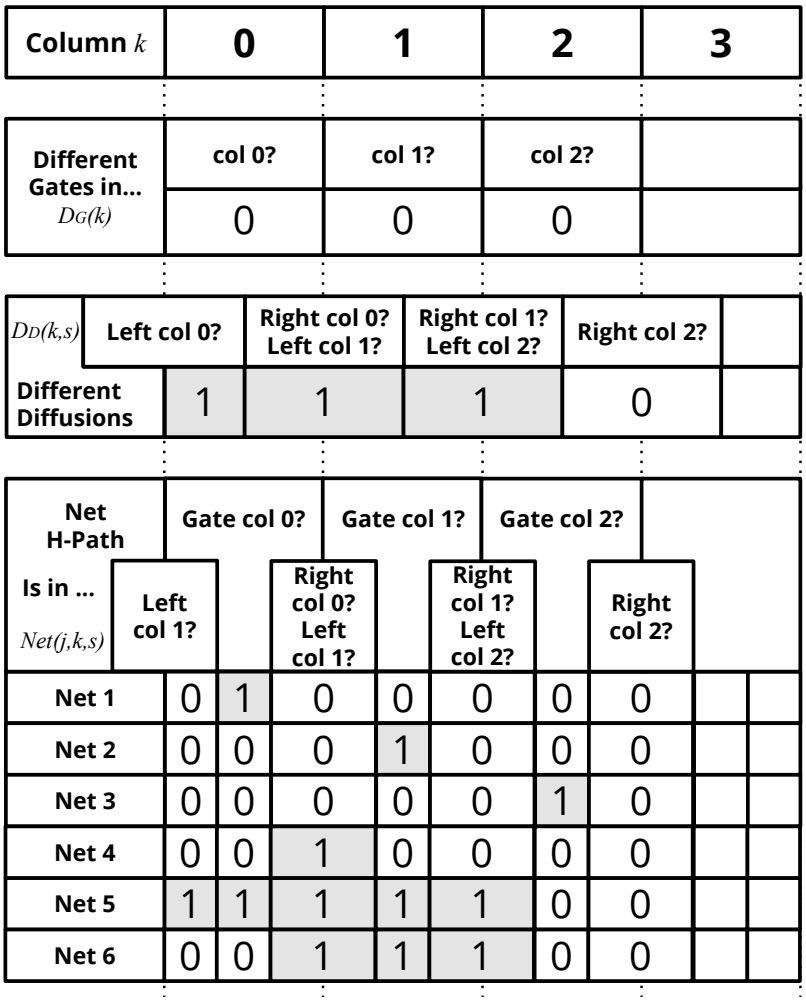

Figure 5. PBSAT Optimizable variables.

A clause generator constrains the SAT variables. The clause generator consists of a situation tester that iterates all possible combinations of the SAT variables, constraining its values if a clause condition is met. Figure 6 presents all the situations where the SAT variables are constrained. The situations are classified into three main types: position clauses, common terminal clauses, and net presence clauses. When a series of variables are constrained to a value, the following conjunctive normal form (CNF) equation is applied to the SAT solver:

$$Clause(Var, Val) = \bigvee_{n=0}^{\#Var} Var_n \oplus Val_n$$

where *Var* represents the SAT variables, and *Val* is the value of such variables to restrict. Position clauses in Figure 6 control the restrictions when instancing any transistor in a specific column. These clauses guarantee width-minimum standard cell placement Iizuka et al. (2006). Among the position clauses exist four situations with their descriptions as follows:

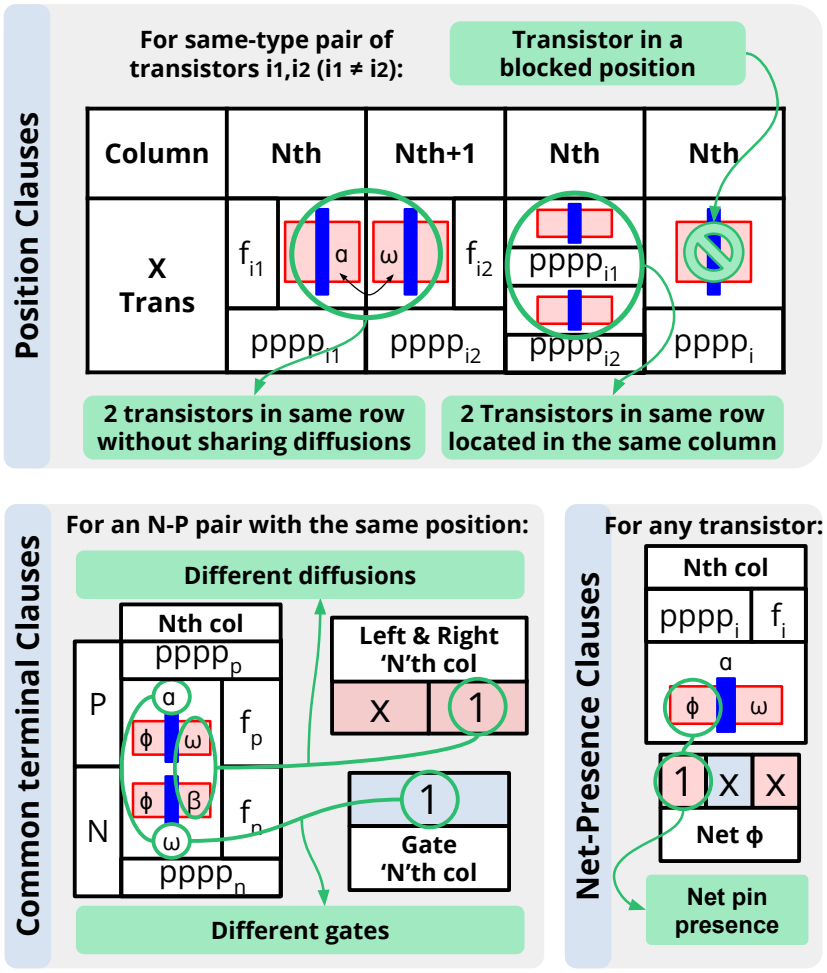

Figure 6. Clauses evaluating different situations to constraint the transistor placement.

- **2 transistors in same row without sharing diffusions**:

  Iteration: *k = [0,W-1), f_1 = [0,1], f_2 = [0,1], i_1, i_2 in N then i_1, i_2 in P, $i_1 \neq i_2$*

  Clause if: *Net(i_1,f_1,Right) $\neq$ Net(i_2,f_2,Left)*

  Var: *{$C_{n,p}(i_1,k)$,$C_{n,p}(i_2,k+1)$,$F_{n,p}(i_1)$,$F_{n,p}(i_2)$}*

  Val:*{1,1,f_1,f_2}*

  The position and flip of two different transistors i_1 and i_2 in the same row are blocked if they cannot abut.

- **2 Transistors in same row located in the same column**:

  Iteration: *k = [0,W), i_1, i_2 in N then i_1, i_2 in P, $i_1 \neq i_2$*

  Clause if: *Always*

  Var: *{$C_{n,p}(i_1,k)$,$C_{n,p}(i_2,k)$}, **Val:**{1,1}*

  The position of two different transistors i_1 and i_2 in the same row and column is forbidden.

- **Legal instantiation of transistors**:

  Iteration: *k_1 = [0,W-1), k_2 = [k_1+1,W), i in {N P}.*

  Clause if: *Always*

  Var: *{$C_{n,p}(i,k_1)$,$C_{n,p}(i,k_2)$}, **Val:**{1,1}*

  A transistor cannot be instantiated in two different positions k_1 and k_2.

- **Transistor in a blocked position**:

  Iteration: *k = [0,W), i in N and P.*

Clause if: *Blocked(k)*

Var: *{$C_{n,p}(i,k)$}*, ***Val:****{1}*

Block the position k of a transistor i if placed in a column blocked by external configurations.

The common terminal clauses and the net-presence clauses stated in Figure 6 constrain the optimization variables. The common gate and diffusion terminals set to zero if transistors position reflect the same pin in both N and P transistors, in any particular column position k,s. Likewise, the net-presence H-path is minimized if a particular net span does not pass through the column position. An initial evaluation of the optimization is calculated with a SAT solution of the previous constraints, and set as the initial optimization. The optimization expression is as follows:

$$Minimize : \sum D_G + \sum D_D + \sum Net$$

The common terminal clauses involve two situations:

- **Different diffusions**:

  Iteration: *k = [0,W), s = [Left,Right], f_n in [0,1], f_p in [0,1], i_n in N, i_p in P*

  Clause if: *NetN(i_n,f_n,s) $\neq$ NetP(i_p,f_p,s)*

  Var: *{$C_n(i_n,k)$,$C_p(i_p,k)$,$F_n(i_n)$,$F_p(i_p)$,$D_D(k,s)$}*

  Val: *{1,1,f_n,f_p,0}*

The optimization variable for common diffusions cannot be minimized (set to 0) if the flip state of any pair of P-N transistors does not match the diffusion nets vertically in any particular k,s position.

- **Different gates**:

  Iteration: *k = [0,W), i_n in N, i_p in P*

  Clause if: *NetN(i_n,Center) $\neq$ NetP(i_p,Center)*

  Var: *{$C_n(i_n,k)$,$C_p(i_p,k)$,$D_G(k)$}* ***Val:*** *{1,1,0}*

  The optimization variable for common gates cannot be minimized (set to 0) if any P-N transistors pair does not match the gate net vertically in any particular k position.

The final optimization constraints involve the net-presence clauses shown in Figure 6. The net-presence will constrain the presence variables in 1 if the positioned transistor contains the net. Figure 7 depicts an example of the net-presence clauses. The position is determined of the pins (*NetP*) according to the transistors' position using the SAT clauses. After this, the net's leftmost and rightmost position are determined in a thermo-code form (*NetRL*,*NetLR*). The horizontal span net path (*Net*) is then calculated by performing an AND gate between the previous leftmost and rightmost variables. In this case, a net's positions are handled with l bins instead of columns and sides k,s. The transformation expression for the horizontal bins is $l = 2 * k + sl$, where sl is 0, 1, and 2 for left, center, and right respectively. The possible situations are:

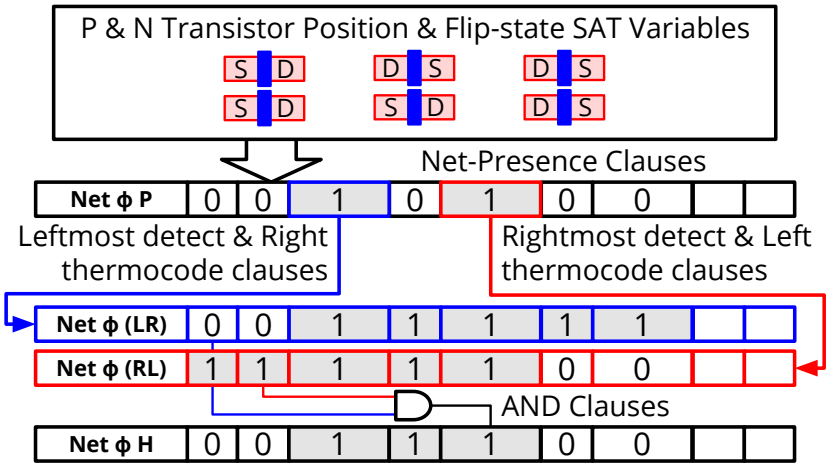

Figure 7. Clause formulation for the horizontal net path optimizer.

- **Net pin presence according to transistor position**:

  Iteration: $k = [0,W)$, $s = [Left,Center,Right]$, f *in* $[0,1]$, i *in* N *then in* P, $j = Net(i,f,s)$

  Clause if: $Net(i,f,s) \not\subset \{VDD,VSS\}$

  Var: $\{C_{n,p}(i,k),F_{n,p}(i),NetP(j,k,s)\}$ ***Val:*** $\{1,f,0\}$

  The presence of a j net in the k,s position must be active (not set to 0) if the transistor contains the net.

- **Leftmost detect & Right thermocode**:

  Iteration: $l_1 = [0,2*W)$, $l_2 = [l_1,2*W+1)$, j *in* X

  Clause if: *Always*

  Var: $\{NetP(j,l_1),NetLR(j,l_2)\}$ ***Val:*** $\{1,0\}$

  If a pin presence is located in the l_1 position, $NetLR(j)$ cannot be zero for all l_2 positions to the right.

- **Rightmost detect & Left thermocode**:

  Iteration: l_1 = [0,2*W), l_2 = [0,l_1], j in X

  Clause if: *Always*

  Var: $\{NetP(j,l_1), NetRL(j,l_2)\}$ ***Val:*** {1,0}

  If a pin presence is located in the l_1 position, $NetRL(j)$ cannot be zero for all l_2 positions to the left.

- **AND clause**:

  Iteration: l = [0,2*W), j in X

  Clause if: *Always*

  Var: $\{NetLR(j,l), NetRL(j,l), Net(j,l)\}$ ***Val:*** {1,1,0}

  If both thermo-coded variables are set to 1 for a position l, the horizontal path cannot be optimized (set to 0).

1.1.3. Placement Algorithm. All the previously described situations are encapsulated in the situation clause generator. These situations are evaluated and constrained in SAT clauses into the placement algorithm procedure. Algorithm 1 presents the placement procedure. This algorithm uses the situation clause generator after analyzing the netlist, which contains N, P, and X. After the clauses, the PB-SAT library will convert the SAT problem to solve. The first iteration of the solutions does not activate the optimization. If an SAT solution exists, optimization is enabled to search for the best solution. The optimization sets the objective of the minimization according to the previous solution evaluation minus 1. When the SAT problem is

Algorithm 1 Proposed placement algorithm

```
1:  procedure PLACEMENT(analysis,iterations)
2:      analysis.W ← Max(#N,#P)
3:      for i = 0 to iterations do
4:          PBSat ← initPBSat()
5:          PBSat.variables ← Sum(#Variables)                    ▷ Table 1
6:          PBSat.clauses ← ClauseGen(analysis)                  ▷ Figure 6
7:          PBSat.OPTExpr ← MINIMIZE()
8:          PBSat.OPT ← Off
9:          psolved ← Solve(PBSat.getSAT)                        ▷ Philipp and Steinke (2015)
10:         if psolved then
11:             do
12:                 best ← PBsat.solution()
13:                 PBsat.OPT = PBsat.Eval(best) − 1
14:                 psolved ← Solve(PBSat.getSAT)
15:             while not psolved
16:             analysis.{Pos,Flip} ← Translate(best)
17:         else
18:             analysis.W ← analysis.W + 1
19:         end if
20:     end for
21: end procedure
```

no longer satisfiable, the placement result is converted using the best solution. If a placement fails, the number of columns is increased by one ($W = W+1$).

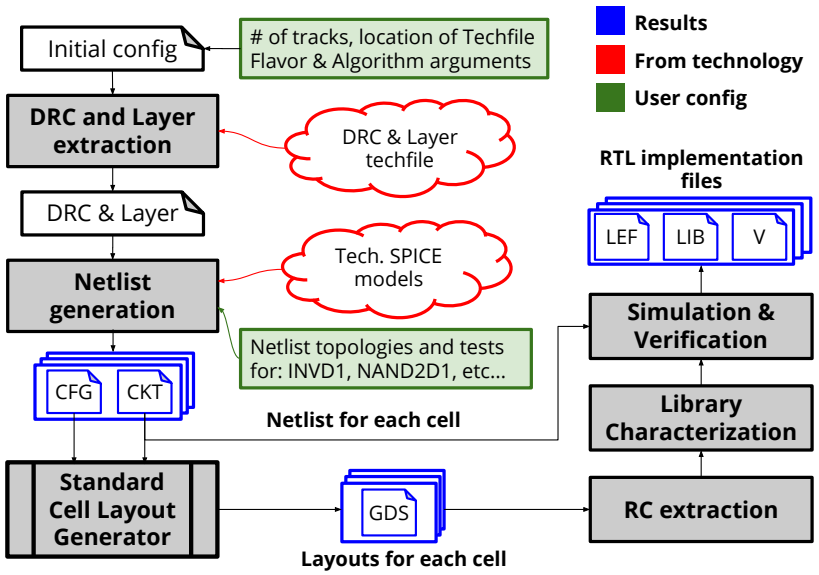

Figure 8. Standard cell automatic layout generator.

Algorithm 2 Standard cell layout generator

```
1: procedure LAYOUTGEN(Netlist,config)
2:     DR ← DRCallPossible(config)
3:     found ← false
4:     analysis.{N,P,X} ← Netlist.Ext(DR)
5:     Placement(analysis,conFigureplaceIter)          ▷ Proposed placement
6:     if exists analysis.Pos then
7:         analysis.Bins ← Stick(analysis)
8:         analysis.Nets ← Pins(analysis)
9:         Routing(analysis,conFigureplaceIter)        ▷ Maze router
10:        if exists analysis.Routes then
11:            analysis.geom ← Geometry(DR)
12:        end if
13:    end if
14: end procedure
```

1.1.4. Routing and Generation. The proposed placement algorithm is implemented inside a fully automatic standard cell generation procedure, as summarized in Figure 8. The generator takes input configurations from technology characteristics and then passes through design rule (DR) extraction and netlist generation. The extracted design rules are limited to fully legalize any placement and routing solution. The netlist generation scales transistors assuring optimal electrical switching, and it performs adequate transistor folding. After generating the layout for each cell, RC extraction and library characterization are performed to

generate RTL and final library files for EDA integration.

The layout generation is included in the overall generation procedure, which output a GDS for each cell in Figure 8. Algorithm 2 describes the automatic cell layout generation. The algorithm input takes the previous netlists and configurations to issue legal placements. Processed design rules (*DR*) with the input netlist are stored in the *analysis* object. The dummy routing algorithm uses an A-star maze solver with a simple net ordering based on horizontal wire length. The final layout is generated using the previously processed netlist, placement, and routing solutions.

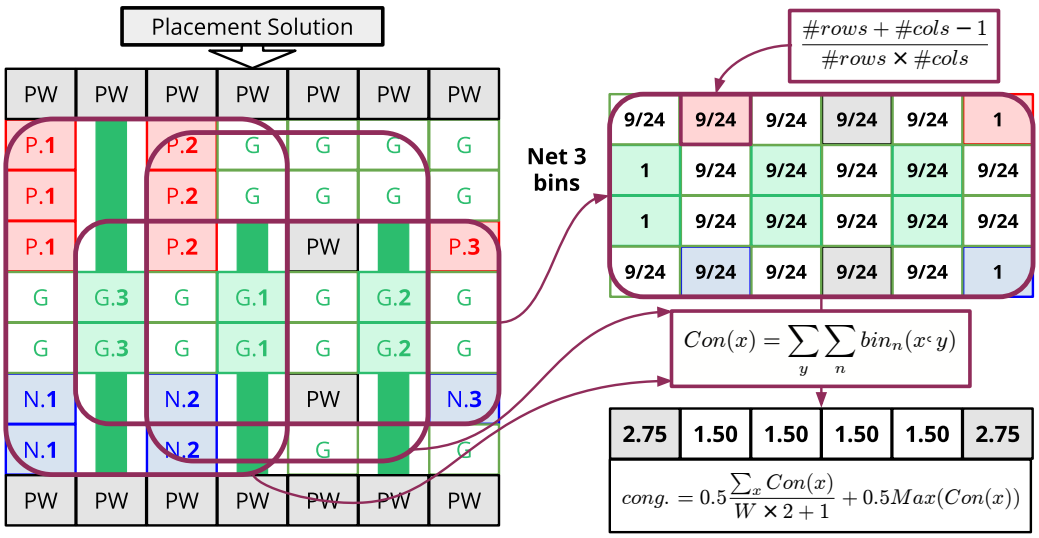

Figure 9. Route congestion estimation, based in the work described in Jo et al. (2019).

1.1.5. Results. The proposed placement algorithm is evaluated inside the standard cell generator using a benchmark that estimates routing congestion. Figure 9 provides a route congestion estimation formulation extracted from Jo et al. (2019) used as the benchmark. Each net has a matrix, whose elements represent a probability of occupation in the bins. The routing algorithm sets the matrix size and the design rules, which uses both diffusion and gate

locations as the rows, and the number of the configured tracks for the columns. According to the transistor's position and width, a probability matrix is set for the pin bins. Other bins in the matrix are modulated using the number of rows and columns. The congestion per track ($Con(x)$) is calculated by accumulating all the nets congestion and the bins vertically. The total reported congestion ($cong.$) is a weighted addition between the average ($\sum Con(x)$) and the maximum ($Max(Con(x))$) of all track metrics.

Design rules and geometries of a commercial 180nm technology node were employed to generate cells with a height of 9 tracks. Table 2 exhibits an overall comparison of the proposed placement with an implementation of the base algorithm in Iizuka et al. (2006). To summarize, several types of standard cells were reported like basic combinational logic cells (INVD1, NAND2D1, NOR2D1, BUFFD1, AO21D1, NAND8D1), a tri-state buffer (BUFFTD1), a half and a full adder (HAD1, FAD1), a D-latch (LND1), and a D-flip-flop (DFD1). For the basic combinational cells, performance results are identical to the algorithm in Iizuka et al. (2006). However, the original algorithm cannot fully route the placements of the tri-state buffer and the D-latch. In contrast, the proposed algorithm performs a full routing of the tri-state buffer and the D-latch placements. The D-latch and D-flip-flop display overall congestion reduction down to 30%.

Samples of generated standard cell layouts are shown in Figure 10. Routing results in these layouts demonstrate the advantages of linking placement with transistor design, basic design rules, and route optimization to generate compact standard cells.

Table 2
Routabiliy Performance Comparison for 9 tracks cells.

Cell	PB-SAT Iizuka et al. (2006)			This place			Route Vert. Cols.
	cong. Figure 9	Routed Done	Routed Total	*cong.* Figure 9	Routed Done	Routed Total	
INVD1	1.50	2	2	1.50	2	2	3
NAND2D1	1.60	4	4	1.50	3	3	5
NOR2D1	1.70	4	4	1.60	3	3	5
BUFFD1	1.40	5	5	1.40	5	5	5
AO21D1	1.57	8	8	1.55	8	8	9
NAND8D1	1.58	12	12	1.58	12	12	17
BUFFTD1	0.91	13	14	0.87	14	14	15
HAD1	0.88	20	27	0.83	22	27	25
FAD1	0.98	45	48	0.96	45	48	39
LND1	1.28	20	22	0.97	22	22	23
DFD1	1.27	28	34	0.81	30	33	31

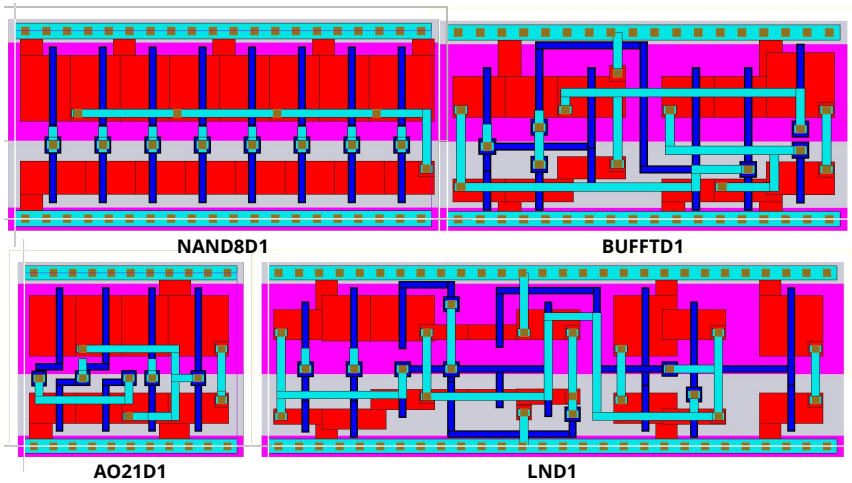

Figure 10. Samples of generated layouts with the proposed placement algorithm and a classic maze router.

1.1.6. Summary. The presented section describes a placement algorithm applied to a fully automatic standard cell generation procedure. The proposed transistor placement formulation uses SAT clauses, with optimization in both terminals and net spans for routing im-

provement. PB-SAT is employed to convert the optimization steps into CNF clauses. The standard cell generation procedure is aware of design rules, transistor sizing, and routing optimization. The linkage of algorithms into a single procedure demonstrated a reliable method to generate legal layouts. The congestion was reduced to 30% compared to the original placement algorithm, as demonstrated with the commercial technology node's final generated layouts.

1.2. Graph-based Standard Cell Generator

In this section, a graph-based formulation is proposed for the placement of standard cells. This placement is based on algorithms presented in Hwang et al. (1990); Hsich et al. (1991). The graph reversal algorithm first starts with the identification of clusters. These clusters are then paired using N-P transistors. Compared to the original algorithm, this proposal implements the placement described in section 1.1 as an aided placement. The result of this pairing presents better matches for N-P transistor pairs. Finally, result implementations will be presented by also adding an SAT-based routing algorithm presented in Ryzhenko and Burns (2012).

1.2.1. Introduction. In different technology nodes, the standard cell library allows the designer to implement very large-scale integrations (VLSI). Electronic design automation (EDA) software takes a characterization of the library and transforms hardware description language (HDL) into a layout according to different integration parameters Wang et al. (2009). Usually, the technology provider only delivers a set of cells with certain constraints. Each cell layout is stored in a stick format, then translated into geometry using design rules (DR) and manual labor de Dood et al. (2019).

A dynamic approach is necessary to break the fixed nature of the standard cell generation. Dynamic generation of standard cells allows EDA to implement circuits with different

constraints and design standards. Zhao *et al.* presents an AES implementation that requires diodes inserted in the body of the cells to achieve low power. Gomez *et al.* utilizes different layouts of the same cell to obfuscate the output circuit Gomez et al. (2019).

The main problem with the dynamic generation of the standard cells is fully automating the layout process through placement and routing algorithms. Routing feasibility in standard cells depends strongly on the placement of the transistors due to inherited routing congestion caused by the position of the terminals. Ryzhenko-Burns presents a routing algorithm where routing up to the first metal can achieve 59% of the cells Ryzhenko and Burns (2012). Reducing the congestion would allow better routing and mitigate the scenic routing for the remaining cases. Jo *et al.* presents an optimization for the placement which favors congestion of routing. This kind of routing optimization can be integrated in different placement scenarios like simulated annealing Guruswamy et al. (1997), genetic algorithms Lazzari et al. (2007), circuit grouping Sadakane et al. (1995); Gupta et al. (1996), graph algorithms Maziasz and Hayes (1991); Bar-Yehuda et al. (1989); Hwang et al. (1990), integer linear programming (ILP) Cortadella (2013); Gupta and Hayes (1996, 1998); Lu et al. (2015a) and boolean satisfiability (SAT) Iizuka et al. (2006).

In section 1.1, a pseudo-SAT (PBSAT) placement of cells was optimized for routing. The routing is more feasible for maze solvers by optimizing the horizontal route span according to the terminals. This section proposes yet another placement algorithm by combining a graph-based formulation with the previous PBSAT as an aided placement. The graph reversal algorithm contains several phases: clustering, pairing, and chaining. With the extracted clusters from a particular netlist, the pairing is usually performed by using a set of rules such as a prior-

ity list Hwang et al. (1990); Hsich et al. (1991). The aided placement will be used to determine the proper P-N pair transistors to be used in the graph reversal because of the horizontal span optimization. The results show a 1/3 less of track in routing congestion, and 6× to 30 × faster routing using Ryzhenko and Burns (2012).

1.2.2. Placement Algorithm. A placement algorithm was implemented according to the proposals of Hsich et al. (1991); Hwang et al. (1990); Jo et al. (2018). The algorithm presented in 3 is divided into the following steps: clustering, pairing, chaining, and cell congestion evaluation. The procedure takes an input netlist, and the number of iterations then outputs the placement using the *PutSolution* procedure.

Algorithm 3 Graph-based placement main algorithm

```
1:  procedure PLACEMENT(netlist,iterations)
2:      analysis ← NetlistAnalyze(netlist)
3:      clusters ← Clustering(analysis)
4:      pairs ← Pairing(clusters)
5:      G ← BuildBipartiteGraph(pairs,analysis)
6:      lb ← getLowerBound(pairs,analysis)
7:      S ← Chaining(G,∅,∅)
8:      for s in S do
9:          s.CellCon ← CellCon(s,analysis)
10:     end for
11:     choose s such as s.CellCon is the minimum
12:     analysis ← PutSolution(s)
13: end procedure
```

The clustering algorithm is described in algorithm 4. This algorithm first searches for all primary outputs in PO. Such outputs are defined as nets with at least one connection between any P and N transistors through the drain p_d, n_d or the source p_s, n_s. The algorithm iterates through all found primary outputs and explores all the transistors connected to the iterated output. The clustering procedure ($Cluster$) searches only the P or N transistors in a depth-first

fashion. The connections use the pins to the source (t_s) and drain (t_d) of each transistor as edges. The search stops whenever it encounters another primary output (PO), a previously-explored net (N_p), or a supply (like V_{DD} or V_{SS}). This stop condition is accomplished by recording the explored nets in N_p, removing the explored transistors, and recursively calling the same $Cluster$ procedure. The output of the clustering procedure is stored in P_t, N_t, is then stored in the final $clusters$ variable as several sets of transistors. Any remaining transistors not extracted from the $Cluster$ procedure are stored as a separated cluster, but this typically happens when the circuit is asymmetric (Ex: more N transistors than P transistors).

Figure 11 shows an example of the graphic procedure to find all primary output associated transistors. The extracted clusters can be different according to the order of PO. In the case of Figure 11a), if PO2 is iterated first, cluster 2 is detected early. In Figure 11b), if PO1 is iterated first, the propagation goes through all the transistors regardless, making pass-gates part of the first cluster. For this reason, the algorithm has to contain a special filter with pass-gates, which are defined as pairs of transistors P and N that have the same drain and source connections.

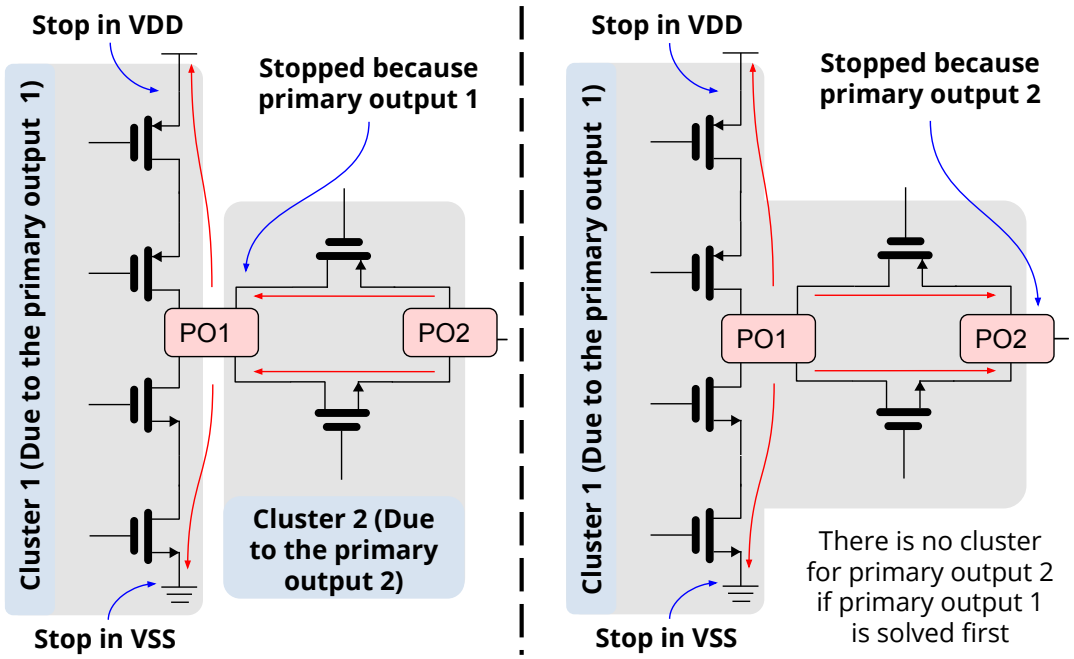

Figure 11. Circuit clustering procedure.

The pairing is in charge of assigning each P-transistor with a matching N-transistor to do the chaining process. This method should maximize as many vertical connections as possible. The pairing is executed for each extracted cluster. Since transmission gates are identified as individual clusters, no pairing is needed. For the other compound gate clusters, one of two methods displayed in Figure 12 is executed. The priority method is the original pairing procedure proposed in Hwang et al. (1990), which sorts the transistors by size, then pair the transistors according to a list of priorities. The pairing of P-N transistors must be unique, meaning a taken transistor cannot appear twice in two different pairs. The second method is a placement-aided alternative which is proposed section 1.1. A smaller and faster placement algorithm is performed for pairing purposes for all formed clusters. The PB-SAT nature of the aided placement optimizes the P-N transistors positioning to match common columns for the formed pairs. With this method, the formed pairs will have matched characteristics such as common gates and com-

Algorithm 4 Clustering algorithm

1: $clusters \leftarrow \emptyset$
2: **procedure** CLUSTERING(*analysis*)
3: $N \leftarrow$ N transistors in *analysis*
4: $P \leftarrow$ P transistors in *analysis*
5: $PO \leftarrow \{n_c \mid n_c \subset \{p_d, p_s\} \bigwedge n_c \subset \{n_d, n_s\} \; \forall \; p \in P, n \in N\}$
6: **for** n_c in PO **do**
7: $P_t \leftarrow Cluster(\emptyset, P, n_c, \emptyset, PO)$ ▷ Find cluster N for net n_c
8: $N_t \leftarrow Cluster(\emptyset, N, n_c, \emptyset, PO)$ ▷ Find cluster P for net n_c
9: $clusters \leftarrow clusters + \{P_t, N_t\}$
10: $N \leftarrow N - N_t \; ; \; P \leftarrow P - P_t$ ▷ Remove the clustered
11: **end for**
12: **if** $P \neq \emptyset \bigvee N \neq \emptyset$ **then** ▷ Circuit is asymmetric
13: $clusters \leftarrow clusters + \{P, N\}$
14: **end if**
15: **end procedure**
16: **procedure** CLUSTER(I,T,n,N_p,PO)
17: $TC \leftarrow \{t \in T \mid n \subset \{t_D, t_S\}\}$ ▷ Get transistors connected to n
18: $N_p \leftarrow N_p \cup n$ ▷ Add the current net n to n_p
19: $O \leftarrow I \cup TC$ ▷ Add the transistors t to output
20: $T \leftarrow T - TC$ ▷ Remove the transistors TC from T
21: **for** t in TC **do**
22: **if** $n = t_d$ **then** $d \leftarrow t_s$ **else** $d \leftarrow t_d$ ▷ Get the opposite net in d
23: **if** $d \notin n_p \cup PO \cup \{V_{DD}, V_{SS}\}$ **then** ▷ Stop condition
24: $O \leftarrow Cluster(O, T, n, N_p, PO)$ ▷ Explore further
25: **end if**
26: **end for**
27: **return** O
28: **end procedure**

mon diffusions, as well as optimizing parts of the placement with horizontal span to optimize the routing.

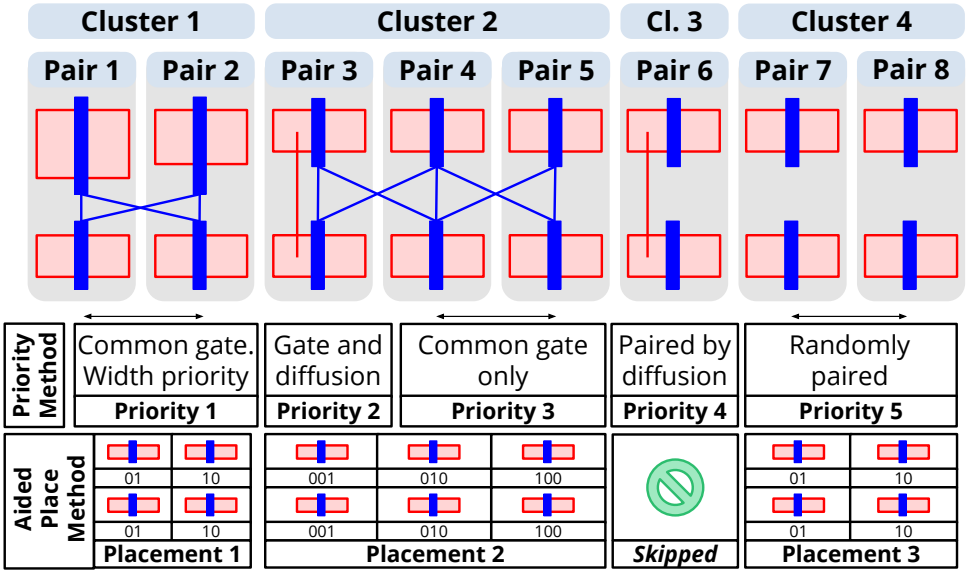

Figure 12. Circuit pairing procedure.

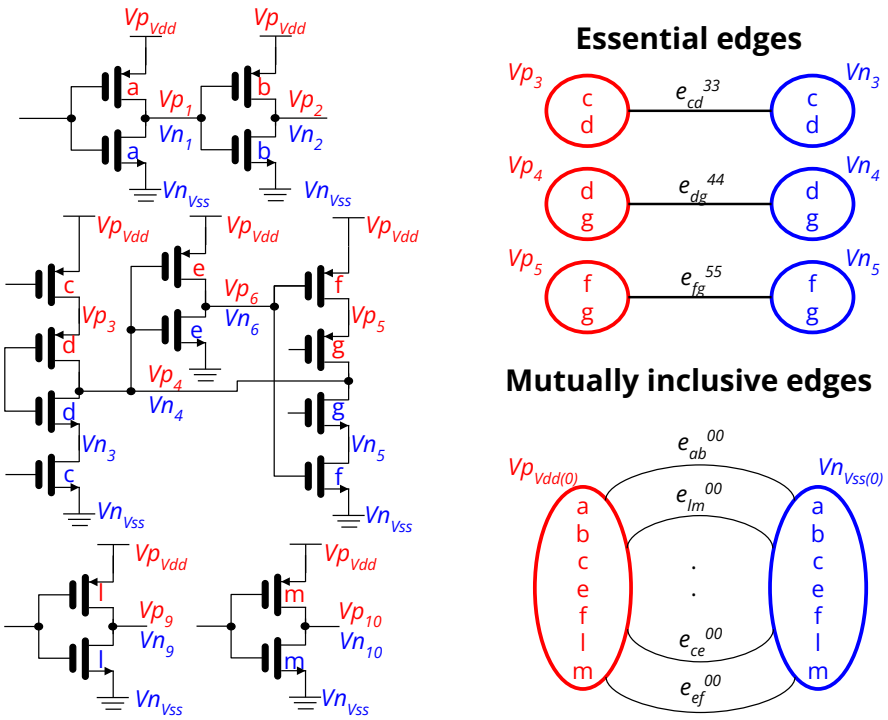

Figure 13. Bipartite graph generation example for a latch.

With the previously formed pairs, a bipartite graph is created for further chaining Hwang et al. (1990). Figure 13 shows an example of the bipartite graph for a latch. The vertices of the bipartite graph are the nets for individual N and P regions. Each vertex contains all the transistors that are connected to such a net. An edge is a possible abutment between two paired P-N transistors connected by a pair of P-N nets (vertices). The edges in this design are mutually exclusive or mutually inclusive, depending on the transistor pairs and connections. For example, the sequence of two pairs *ab*, *jk*, and *im* are mutually inclusive because their abutments can be realizable at the time. In the opposite case, the sequence of two pairs *ce*, ef, and fi are mutually exclusive because their abutments cannot be sequential for a given solution. Although the bipartite graph seems non-Hamiltonian, the exploration of the bipartite graph removes progressively mutually-exclusive edges. This behavior avoids the repetition of abutments in the solution.

The chaining algorithm is described in Algorithm 5. After defining the bipartite graph, a series of edge chains is output as the possible solutions (S), which start empty. The currently built chain is supplied via B, which also starts empty. The *first iteration* will choose any edge e, then starts a new chain in B' with the chosen edge in both directions e_{kl}, e_{lk}. The procedure also removes the chosen edge and the mutually exclusive edges (XOR) from the graph G. Other abutments are searched by calling the same function, possibly activating the *following iterations*. In these iterations, the algorithm will search for any matching edges (A) from the right transistor pair l. If there are matching edges, the edge e is iterated from the matching edges, and e is aligned and added to the last chain of B. This iteration also removes the added edge and the mutually exclusive edges from the graph G. A new chain is created using the *first iteration and*

new chains procedure if there are no more matching edges from $edgesAbut$. If in any function call the graph is empty, a solution was possible in the chains B, which is added to the solutions object S. The number of chains are also limited from the beginning from the lower bounds predictor lb.

Algorithm 5 Chaining algorithm

```
1:  procedure CHAINING(G, B, S)
2:      if E(G) = ∅ then return S + {B}                ▷ Add this solution
3:      if |B| ≥ lb then return S                       ▷ Stop if solution is greater than LB
4:      if B ≠ ∅ then                                   ▷ Following iterations
5:          l ← B_last(last) → l                        ▷ Get l from last edge e_kl^ij in chain
6:          A ← edgesAbut(E(G), l)
7:          if A ≠ ∅ then                               ▷ If not abutable edges, create chain
8:              for e in A do
9:                  B' ← B
10:                 B'_last ← B_last + {align(l, e)}    ▷ Abut the aligned edge
11:                 G' ← G
12:                 E(G') ← E(G') − XOR(G, e) − {e}     ▷ Remove exclusive
13:                 S ← Chaining(G_p, B_p, S)
14:             end for
15:         end if
16:     end if
17:     First iteration and new chains
18:     for e in G do                                   ▷ Pick any edge in the graph
19:         for e_x in {e_kl, e_lk} do                  ▷ Iterate both orientations
20:             B' ← B + {e_x}                          ▷ Create new chain with oriented edge
21:             G' ← G
22:             E(G') ← E(G') − XOR(G, e) − {e}         ▷ Remove exclusive
23:             S ← Chaining(G_p, B_p, S)
24:         end for
25:     end for
26: end procedure
```

The possible solutions are generated in sequences of abutments. The solutions' form is presented as chains of oriented edges. A possible solution for the latch in Figure 13 is:

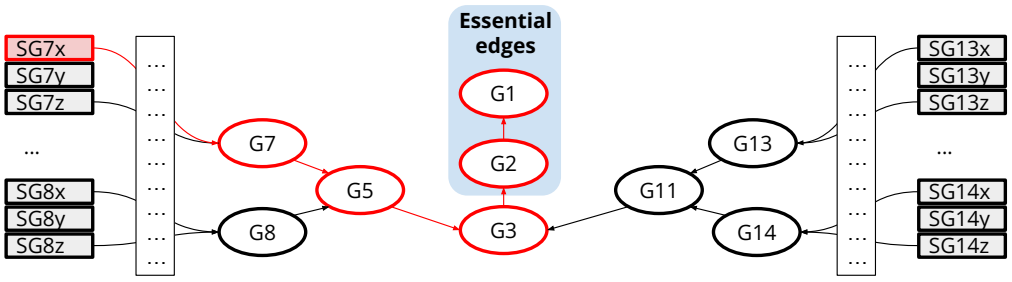

Figure 14. Algorithm searching tree outputs.

$$S(best) = \{e_{ab}^{00}\}, \{e_{cd}^{33}, e_{dg}^{44}, e_{gf}^{55}, e_{fe}^{00}\}, \{e_{lm}^{00}\}$$

The tree of possible solutions is generated like the Figure 14. Given all the solutions by the algorithm, a best-search procedure is implemented using an optimization function. Jo *et. al* Jo et al. (2018) proposes an optimization for routing. The algorithm proposal generates a complete search tree then calculates a benchmark for routing estimation. This estimation is based on horizontal bin congestion according to the design rules in the target technology. The final part of the algorithm 3 searches for the minimum congestion. The detailed definition of the routing bins is described in Figure 9. The pin connections are detected in the N, P, and G bins for any transistor in the chain. Depending on the sizes, any possible connection is marked to the transistors as a pin. After defining the ports, the columns and rows are accumulated according to the Jo *et. al* Jo et al. (2018) formulation for any given solution from the search tree.

1.2.3. Results. The proposed algorithm for placement, along with the previous placements Hsich et al. (1991); Iizuka et al. (2006) and the placement previously discussed in section 1.1 were implemented as standalone generators. The routability is evaluated of the

generated placement results using the same netlists for each standard cell circuit. The route congestion estimation, previously used as an optimization metric, is also used as a benchmark to evaluate the final result in routing terms divided by the number of tracks Jo et al. (2019). Additionally, a router based on SAT formulations and a depth-first search (DFS) maze algorithm was implemented using the algorithm presented in Ryzhenko and Burns (2012). This router generates several routes using DFS, and these routers are introduced in the SAT clauses to find a feasible layout using only the first metal. The output layout is delivered in stick diagrams but also translated into the technology layout using the design rules.

Table 3
Place and Route Performance Comparison for 12 tracks cells.

Cell (#Tr)	**Hsich et al. (1991); Jo et al. (2019)**			**PBSAT Iizuka et al. (2006)**			**PBSAT + Route Aware (Sec. 1.1)**			**This Place**		
	Cong	**Runtime [s]**		**Cong**	**Runtime [s]**		**Cong**	**Runtime [s]**		**Cong**	**Runtime [s]**	
	÷12	**Place**	**Route**	÷12	**Place**	**Route**	÷12	**Place**	**Route**	÷12	**Place**	**Route**
INV(2)	0.625	0.00	0.84	0.625	0.00	0.79	0.625	0.02	0.83	0.625	0.00	0.80
NAND2(4)	1.244	0.00	2.92	1.389	0.00	4.63	1.244	0.08	2.92	1.244	0.02	2.94
NOR2(4)	1.244	0.00	5.12	1.383	0.00	5.40	1.244	0.12	5.08	1.244	0.02	5.12
AND2(6)	0.697	0.00	53.14	0.752	0.01	32.00	0.752	0.40	31.10	0.689	0.05	30.99
OR2(6)	0.686	0.00	16.71	0.747	0.01	15.67	0.747	0.52	15.33	0.686	0.06	15.18
BUFF(4)	0.733	0.00	8.00	0.733	0.01	8.17	0.733	0.07	8.13	0.733	0.02	8.04
XOR2(12)	0.996	0.00	96.76	1.094	0.11	133.54	1.094	23.37	101.10	0.954	0.16	94.67
XNR2(12)	1.022	0.03	99.47	1.094	0.14	141.85	1.094	37.24	78.95	0.954	0.16	89.20
AO21(8)	0.785	0.00	161.37	1.639	0.02	218.77	1.198	2.87	165.70	0.779	0.38	168.02
OA21(8)	0.770	0.00	110.55	1.670	0.02	176.48	1.186	3.81	107.62	0.761	0.36	113.26
NAND8(16)	1.325	0.62	220.73	2.139	0.42	974.53	1.252	939.84	137.51	1.252	8.53	167.79
NOR8(24)	1.471	**1.07**	1493.83	1.806	136.47	341.56	2.390	226337.32	227.93	1.270	2261.72	3266.30
BUFT(12)	1.049	0.05	144.39	1.888	0.09	126.53	1.436	181.30	100.77	0.966	0.30	129.51
MUX2(12)	1.002	0.00	48.46	2.094	0.12	61.32	1.214	42.75	49.99	0.960	0.23	42.60
MUX3(20)	1.084	11.76	213.97	2.988	4.64	N/A	1.252	4165.92	137.97	1.101	**1.93**	289.88
MUX4(26)	1.206	107055.02	683.94	2.778	471.12	N/A	1.779	60551.38	335.68	1.101	**53.46**	**296.01**
LN(16)	0.690	4.56	257.92	1.288	1.93	486.08	0.689	1083.04	374.28	0.683	4.42	1273.39
HA(20)	1.204	105.38	**4753.67**	2.456	15.21	N/A	1.644	9294.24	5773.15	1.136	**13.35**	9313.22
DF(24)	0.795	49.20	**4794.39**	1.565	17494.38	N/A	1.120	36013.82	11130.78	0.748	**6.22**	6786.71
DFCN(28)	0.805	307.55	4270.82	2.872	6345.29	N/A	1.481	84346.43	4130.80	0.887	**11.55**	4502.45
FA(36)	N/A	N/A	N/A	2.567	7275.34	N/A	2.113	53535.05	N/A	**0.865**	34470.61	**36725.45**

Table 3 presents the comparison between the different placement algorithms. The generator utilizes a 200nm technology node for the design rules. The algorithms are executed in a Linux environment with Intel(R) Core(TM) i9-9820X CPU at 3.30 GHz with 32 GB of DDR3 RAM. The algorithms were executed standalone to compare the runtime between the based works and this proposal fairly. The following 12-track cells are reported: combinational cells (INV, NAND2, NOR2, AND2, OR2, BUFF, XOR2, XNR2, AO21, OA21, MUX2), combinational cells with several inputs (NAND8, NOR8, MUX3, MUX4), a tri-state buffer (BUFFTD1), a half and a full adder (HA, FA), a D-latch (LN), a simple D-flip-flop (DF), and a D-flip-flop with asynchronous reset (DFCN). The most superficial combinational cells present a similar placement time, routing time, and congestion. Our proposal shows XOR2, XNR2, AO21, OA21, and MUX2 with slightly worse placement times but better congestion outputs, allowing the router to solve faster. The original LiB algorithm can output routable results for some complex cells like MUX3, MUX4, DF, DFCN, and HA, but the runtime is $6\times$ to $30\times$ slower. LiB also outputs these complex cells with an additional average of 1/3 less of single-track routing congestion ($12.22 - 11.93$), which causes worse scenic routing. The NAND/NOR8 presents better results in PBSAT. Our proposal is the only implementation that allows the full-adder to be feasible for routing.

Figure 15 presents the previously generated standard cells layouts. These layouts are generated using the placement presented in this section and the routing algorithm present in Ryzhenko and Burns (2012). This generator does not implement a compaction algorithm, making the standard cells generated from the stick diagram outputs.

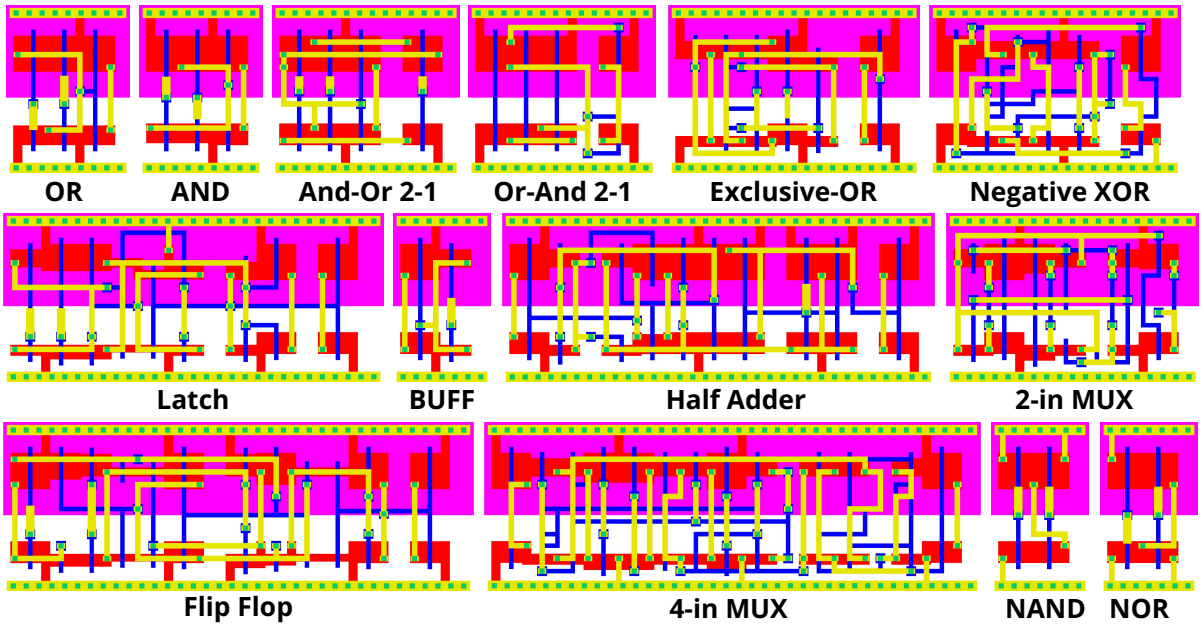

Figure 15. Samples of generated layouts with the proposed placement algorithm and a classic maze router.

1.2.4. Summary. This section presented a placement algorithm by combining a graph-based algorithm with a PBSAT-based aided placement for the optimized pairing of the transistors. This aided placement is run in the netlist's formed clusters to identify the best pairing instead of using priorities. The optimized pairs allow the router to have better congestion metrics. The congestion is generally reduced for complex cells around 1/3 of single-track, allowing less scenic routing. The execution is observed 6× to 30× faster routing by applying a maze-SAT routing algorithm than the original graph transversal method. This algorithm also allows complex netlists like the full adder with 36 transistors to be implemented entirely with routing up to the first metal.

2. Low-Power Security in Formal Verified Circuits

Previously we talked about *imaging reversing* for systems, and we described standard cell generators which can be later used for layout obfuscation. As for the system, we need to secure the data for sending and the programs to be executed. Ciphering can be used to prevent *channel capture* and *firmware extraction*. The advanced encryption standard (AES) is general security ciphering used for obscure data through channels. The cipher can be applied to the firmware or the sent data. In the special case of low-power systems, the designed hardware needs to accomplish low overhead in both area and power.

Security hardware like AES can be reusable by using hardware generators. This generator contains the processor and buses that connect all the peripherals. For chisel Bachrach et al. (2012), the result is a register-transfer level (RTL) file that can be used in digital flows for ASIC and FPGA. As stated in Nikolic et al. (2018), these hardware generators also contain several extensions to manage commercial protocols such as USB, DDR, and SATA.

The system created by the hardware generator needs to be verified to prevent security concerns about *code injection* and known exploits due to faulty implementations. Any system should be verified against the specifications for the processor and protocols used by different buses. The instructions are verified according to the instruction set architecture (ISA) specification in the case of processors.

In this chapter, a system for low-power security is presented. This system features a low-energy AES implementation which comes from a discussion over different acceleration schemes for the algorithm's execution. The integration of such a system is done by a chip

generator named Olinguito. This generator can output RTL code for placement and routing tools. In addition to the RTL, the utilities for handling analog implementations are shown in the always-on domain that allows energy management. The processor of the generator is verified by a set of tests in both formal and functional domains. A set of formal constraints are created for each instruction according to the ISA specification. As for the functional approach, the execution in simulation is compared to a golden model while searching for the maximum coverage of cases using a genetic algorithm.

2.1. AES Sbox Acceleration Schemes

This section describes a system with a low-energy AES implementation in a system-on-chip. First, an SBOX acceleration as a custom processor instruction is presented. This custom instruction is used in software AES, and then the execution characteristics are measured to find the heaviest operations. Finding out that the algorithm mainly relies on memory access, an AES core with direct-access memory (DMA) is presented. Push-pull and DMA executions of the AES core are compared to find the lowest energy consumption in this system-on-chip.

2.1.1. Introduction. Sensors for data-sensitive applications need secure communications to send data in a network, which demands cryptographic algorithms. These sensors also demand minimum energy to be battery-powered Duran et al. (2020). The cost of energy is illustrated in Figure 16, where the access to data stored in SRAM and DRAM represents up to 6400 times the energy for calculations in an ALU Horowitz (2014). The implementation of cryptographic accelerators demands high energy costs due to the interaction of data stored in the system. Regardless, in minimum energy systems with sensitive data, it is necessary to implement an efficient cryptographic accelerator to decrease energy consumption Zhang et al.

(2018); Mathew et al. (2014); Zhao et al. (2015) and make resistant against side-channel attacks Bilgin et al. (2015); Yu and Köse (2016). A trade-off between the processing and transport of data needs to be found for low-energy systems.

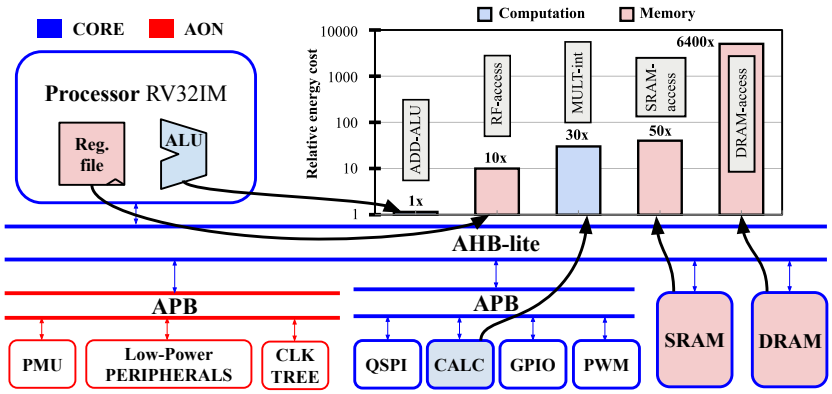

Figure 16. Relative energy cost of different hierarchical computation components.

A common acceleration target is the advanced encryption standard (AES) due to its usage in security Banerjee et al. (2018); Banerjee et al. (2020); Banerjee and Chandrakasan (2021); Mathew et al. (2014). Software implementations of the AES algorithm can be power-hungry caused by excessive operations or memory access due to the characteristics of the algorithm. A primary example of excessive power consumption is the substitution byte operation (Sbox) in the AES cipher. The standard defines a series of operations computed in a number field named Galois Field (GF) to perform substitution of text in a non-linear process NIST (2001). In software, this Sbox is usually implemented in RAM as a look-up table, and calculations of ShiftRow, MixColumns, and RoundAdd are performed using general-purpose instructions, which increase the operation energy due to excessive RAM access. The Sbox can be either implemented as a ROM look-up table or calculated using combinatory logic in AES hardware. The other specialized operations of the AES are also implemented as combinatory logic. Banerjee

et al. designed an SoC where the cryptography core is a dedicated state machine for datagram transport layer security (DTLS) control Banerjee et al. (2018). Further enhancements were added to the cryptography core to support TLS Banerjee et al. (2020) and elliptic curve cryptography (ECC) Banerjee and Chandrakasan (2021). Zhang *et al.* propose a modified RAM memory with computation capabilities Zhang et al. (2018). In memory operations, shifting and rotation are optimized using wired shifting and dedicated rotators, including dedicated hardware, to perform the AES cipher's substitution byte operation (Sbox) operation. Hutter *et al.* designed a co-processor aiming at supporting elliptic curve digital signature (ECDS) and AES algorithms Hutter et al. (2011). The co-processor is based on a 16-bit datapath and employs a dedicated RAM macro. Furthermore, Sayilar and Chiou proposed a completely new architecture based on processing units called processing elements (PE) Sayilar and Chiou (2014). Each unit contains basic operations applied in cryptographic primitives.

Cited examples of AES implementations Banerjee et al. (2018); Banerjee et al. (2020); Banerjee and Chandrakasan (2021); Kundi et al. (2020) do not consider memory access and data transportation in an SoC. The AES cores usually add memories near the processing unit or implement non-standard logic cells. A hardware-based AES crypto-acceleration engine is proposed based on memory-mapped schemes to alleviate SRAM access. The crypto-accelerator implements a Sbox unit that can be configured to generate a combinational or pipelined architecture. The design of the Sbox was first tested as an R-type ISA custom instruction in RISC-V, assisting an AES-256 software implementation. Comparatively, a software implementation with the Sbox custom instruction achieves 100 times less energy consumption than the pure software alternative. This proven Sbox unit is included in an AES crypto-accelerated hardware design

whose input and output registers are mapped into the register map. By pushing and pulling these registers, the core reaches down to 8.08 pJ/bit. The usage of a low-power direct memory access (DMA) engine is also exploited with a DMA automaton for faster data fetch and save Morales et al. (2019). The DMA-based AES hardware core reaches down to 3.58 pJ/bit when active. The overall SoC with the hardware acceleration of the AES achieved a 9.7pJ/bit energy consumption executing AES-256, which presents a three-fold improvement over previous reported AES implementations.

2.1.2. Sbox Unit and Software Execution. Banerjee et al. in Banerjee et al. (2018) show a handshake processor for the Datagram Transport Layer Security (DTLS) for network authentication. The same author presents a similar architecture with a crypto-processor for pairing-based cryptography (PBC), which is a variant of elliptic curve cryptography (ECC) Banerjee and Chandrakasan (2021). These systems contain an in-core mapped memory to perform the algorithms for the cryptography calculation. The accelerator can store up to 90 instructions for ECC and PBC calculation in the PBC variant. The energy efficiency of the implementation is high due to the close transport of data between the in-core memory and the processing core. Although dynamic power can be lowered using clock-gating, there is no report for the data transportation from the main memory performed by the processor core. Our primary focus is to demonstrate acceleration by creating a processing unit for AES like the Sbox. The unit is applied to a custom instruction and execute it by an AES encryption program in a RISC-V processor Waterman et al. (2014). With this approach, the cost of the execution is evaluated from the perspective of the SoC.

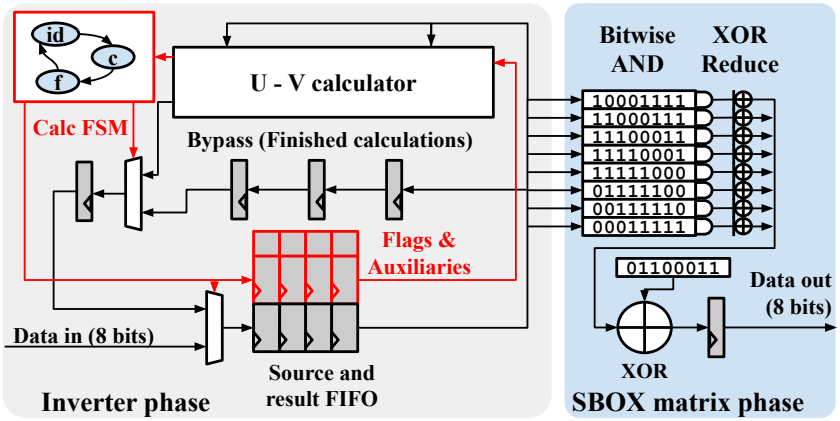

Figure 17. A detailed view of the Sbox architecture using the inverter algorithm.

Sbox Unit. The Sbox requires several calculation resources to perform the substitution in the Galois Field GF (2^8) NIST (2001). This operation is composed of an inverter, and a binary matrix multiplication Hankerson et al. (2004). Software implementations of the AES typically resort to substitute the byte with a 256-byte long table. This substitution avoids calculating the Sbox. A Sbox unit in hardware can be implemented using this approach or performing the calculations Canright (2005); Hankerson et al. (2004).

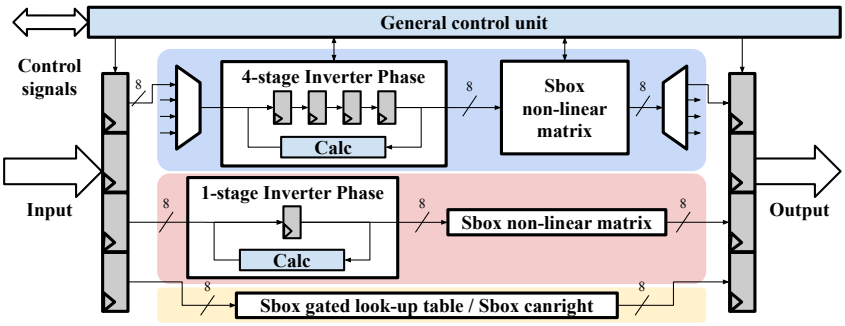

Figure 18. Sbox architectural implementations for logic unit usage. Blue implements a 4-stage inverter Sbox calculator. Red implements four times a 1-stage inverter Sbox. Yellow implements 4 times a combinational Sbox.

The first implementation of the Sbox unit uses a FIFO for buffering calculation steps. Figure 17 shows the architecture of the implemented Sbox in hardware according to the U-

V polynomial calculation described in Hankerson et al. (2004). The inverter phase (in grey) buffers the calculation of a GF inversion and takes up to 15 iterations to perform a GF inversion for an 8-bit entry. When the degree of the polynomials reduces to zero, the matrix phase (in blue) applies the AES affine. This unit allows 32 bits input calculations, inserting four units of data in the 4-deep FIFO.

In Figure 18 the FIFO approach is implemented, along with a Sbox look-up table. A 4-stage inverter FIFO implementation is shown in blue, a 1-stage inverter FIFO in red, and several combinational approaches in yellow. The last one supports a regular look-up table or implementations of reduced Galois Field to $GF(2^4)$ Canright (2005). The units contain a valid-ready decoupled interface, which triggers the input to calculate in 32-bits beats. The hardware can execute any of the three implementations for any 32-bit calculations.

For evaluating the execution of the Sbox unit, a custom instruction is created in a RISC-V processor. Due to the AES algorithm, the Sbox is the heaviest calculation due to the resources of implementing either a GF inverter or a look-up table NIST (2001). The memory Sbox table will be replaced with the custom instruction, reducing the SRAM access Horowitz (2014).

Execution Schemes. Figure 19 describes the implementation in software of the Sbox. The software is based in a lightweight AES library intended for embedded systems. The included Sbox function enters 16 bytes to perform the substitution. The software approach iterates four parts of 4-byte memory. The next 4-byte block's reference is calculated from the i variable as an offset of the data vector to speed up load times using wider loads and stores in the software implementation. The Sbox is calculated depending on using iterated look-ups into memory or using custom instruction. In the instruction approach, the 32-bit data is loaded

directly into a register. The custom instruction is executed, and then the substitution is stored back into memory. No wait states are needed in the application, as the Sbox unit interacts with the processor for the handshake of the result.

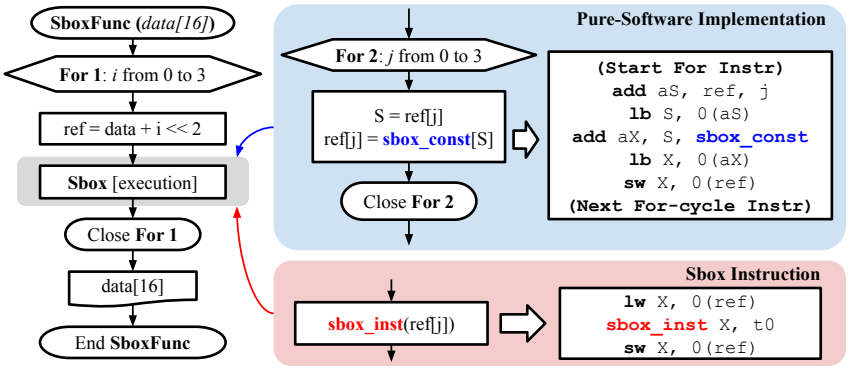

Figure 19. Comparison between pure-software implementation (blue) and the Sbox instruction (red).

The recurrence of instructions is shown in Figure 20. This figure includes the pure-software and the custom Sbox instruction approaches executing AES-256. The Sbox instruction implementation reduces the memory instructions by 15% (lbu) and 36% (add/addi). This approach also presents a diminution of 32-bit transactions (less sw/lw). Although the execution decreases (5927 to 5249), the data movement still impacts considerably. These results demonstrate that interactions in the memory are not concentrated in the Sbox operation. Although the memory operations are reduced, the other calculations of the AES need to be done in-core to avoid memory interaction.

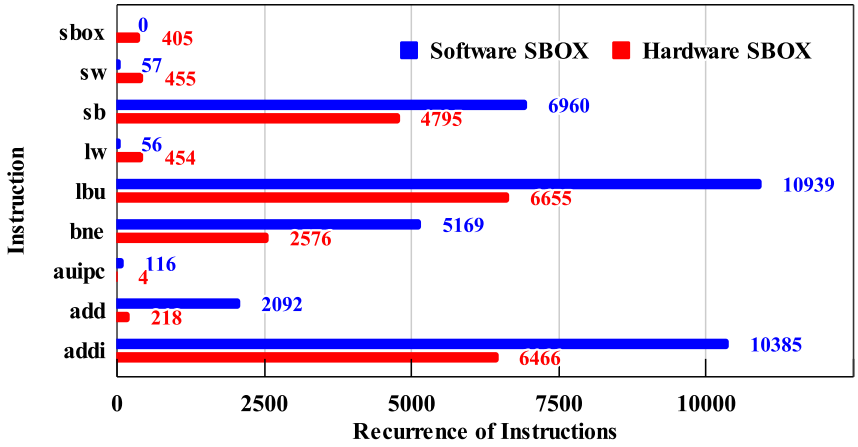

Figure 20. Frequency changes of the RISC-V instructions employed in TinyAES in pure-software (blue) and using the Sbox instruction (red) for AES-256 in a 128-bit block.

2.1.3. AES Core Base Line. A systematic approach is implemented in the perspective of the SoC to alleviate memory access. The implemented SoC, presented in Figure 21, features a RISC-V ISA processor with RV32IM and a total of 4kB SRAM divided between a scratchpad and a system RAM. The power scheme is designed to perform energy harvesting using several analog devices for power monitoring, and the use of active and sleep modes Duran et al. (2020). The Sbox logic unit is implemented in the processor of the SoC as the custom instruction. This SoC is one-time generated with a configurable hardware generator Bachrach et al. (2012).

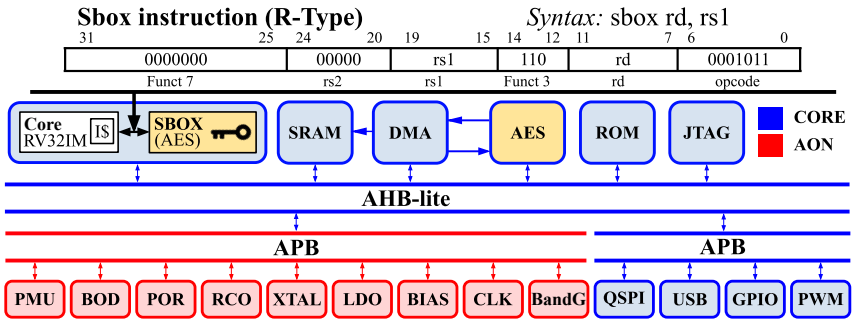

Figure 21. Block diagram of implemented SoC with the R-type Sbox proposed instruction included.

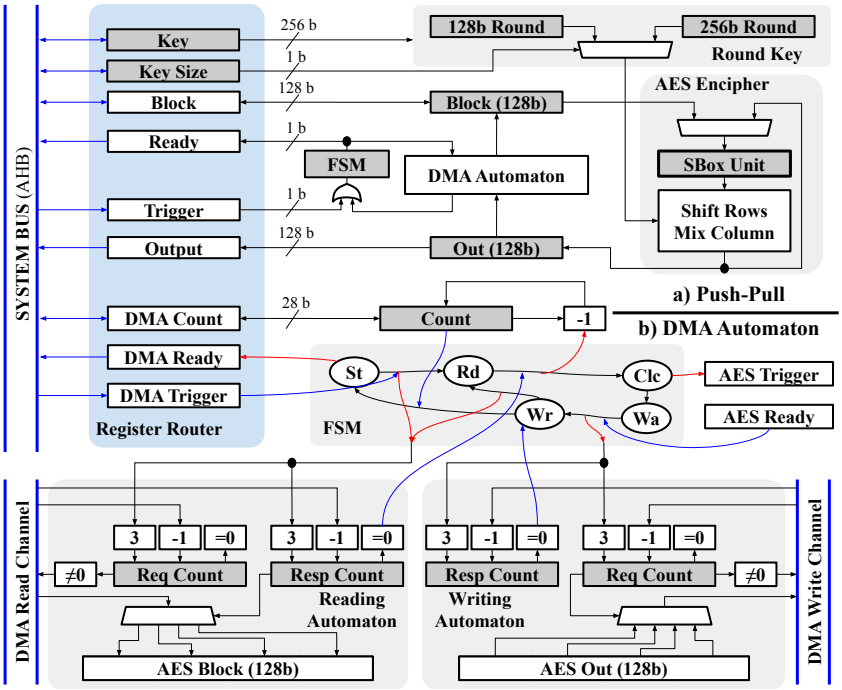

Figure 22. Hardware implementation of the AES core as a peripheral using a) *Push-Pull* registers and b) DMA automaton.

The SoC implements an AES hardware core accelerator, including the previous Sbox unit. Figure 22 presents the AES hardware, which is controlled by an AHB register router. The accelerator works by pushing and pulling registers to perform AES CBC encryption for a block of 128-bit data in 22a). The program first needs to push the 256-bit *Key* through the AHB bus to calculate the AES. Using a *push-pull* approach, the processor needs to push all the data to the 128-bit data register, then perform the *Trigger* register. This peripheral is configurable in 128-bit or 256-bit modes using the *Key Size*. If the *trigger* is asserted, the finite-state machine (FSM) expands the key. A second trigger starts the calculation of the AES encipher. For more than 128-bits blocks, the trigger can be asserted several times. The *output* can be pulled in segments using the register router. The software needs to wait for states for the *Ready* register. For the *push-pull* approach, the processor handles the movement of data.

The calculation of the AES using registers will reduce the data movement between the calculation steps. The central processor still handles the data movement by pushing and pulling by loading and saving into general-purpose registers. This behavior still represents an increased energy usage. To avoid this processor usage, an direct memory access (DMA) automaton is introduced inside of the crypto-accelerator. The automaton is depicted in Figure 22b), which interacts with the original AES core implementation. This automaton exploits the implemented low-power DMA controller included in the current SoC Morales et al. (2019).

In contrast to the *Push-Pull* alternative, the DMA automaton can be used to automatically access the memory and perform the calculations without the processor. The automaton performs reading and writing with separated architectures. Each automaton contains request and response counters, which the state machine keeps track to push and pull the data automatically to the data register. When the DMA trigger is asserted, the automaton first pulls 128-bits of data into the data register, then triggers and waits for the AES core. When the core is ready, the second part of the automaton pushes the resulting 128-bits data from the output into the AHB bus. The state machine keeps track of the number of iterations to input and output the control signals, which the program can read.

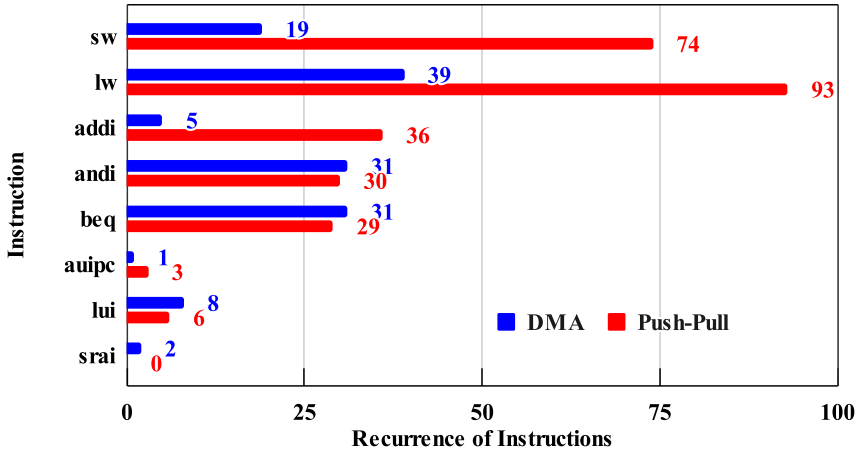

Figure 23. Comparison of the frequency of execution of RISCV-instructions using the *Push-Pull* (red) hardware approach and the DMA automaton (blue) in a block of 128 bytes.

Figure 23 reports the recurrence of instructions' executions in *Push-Pull* and DMA modes over a block of 128 bytes. The key needs to be initially pushed to the registers and expanded in the AES core in both modes. The base save and load instructions (sw/lw) for a key of 256 bits requires eight instructions for each one. In the *Push-Pull* case, the memory needs to be constantly loaded from the system RAM and stored to the peripheral registers, and vice-versa. This approach justifies the main changes of the save (sw), load (lw), and referencing (addi) needed to perform AES-256 through the 128 bytes of data. The DMA version still needs only a particular configuration in both the DMA registers and the DMA-automaton inside the AES core. The excess load instructions found in the DMA version can be justified to the processor constantly loading the DMA-ready flag register. Other kinds of instructions do not represent significant differences, probably attributed to main memory reference calculations.

2.1.4. Results. Table 4 summarizes the synthesis results of the different implementations of the Sbox logic unit approaches in a 180nm CMOS technology. The execution of the Sbox was measured using AES-256 in the software implementation previously described.

The fastest SBox execution belongs to the combinational approaches, but the pipelined inverter only has a 50% overhead for execution. The smallest implementation of the Sbox corresponds to the combinational Canright Sbox, probably due to its optimizations. The Look-up table and the Canright Sbox are similar in power and energy. Energy consumption for the pipelined approaches has at least six times more overhead than pure-combinational Sbox, which is still a good approach compared to the pure software implementations that offer 100 times less.

Table 4
Implementation results for only the Sbox in 180nm technology, executing AES-256.

Sbox impl.	**Cycles 32 Sbox**	**Area [μm^2]**	**Cells [NAND]**	**Power*** **[μW]**	**Energy+** **[pJ/bit]**
1x 4-Pipe	411	28715	5639	3.37	2.85
4x 1-Pipe	207	36692	7205	2.78	1.18
Look-up	**138**	31623	6210	0.87	0.23
Canright	**138**	**11509**	**2260**	**0.77**	**0.21**
Software	717	35317*	5793	103.1	154.05

**Simulation power at 0.4 V, 3.01 MHz*
+Measured energy with 1kbit data
**Size of a 8x256 RAM*

Figure 24 plots the energy consumption per bit of the AES engine performing 1kbit of memory encryption for both *Push-Pull* and DMA implementations. Table 5 reports this energy consumption for the SoC in the different implementations, along with area and performance. The execution of the AES engine is verified at low voltage and frequency by encrypting and decrypting the cyphertext and then running a software CRC-8 checksum. At 0.4V of supply with 3MHz of frequency operation, 8.0pJ/bit (22.2pJ/bit for the SoC) is achieved for the *Push-Pull* approach and 3.6pJ/bit (9.7pJ/bit for the SoC) for the DMA approach. A 2.7x penalty is present for the energy consumption relative to the AES core compared to the SoC. The area

overhead for the SoC moves to 73% when including the AES-DMA.

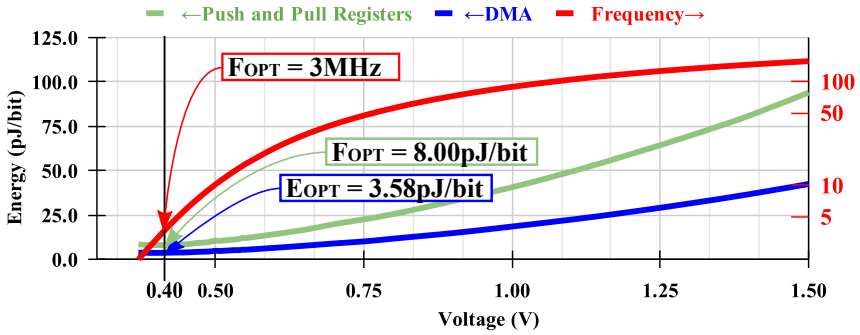

Figure 24. Energy consumption of the AES core with register push/pull and the DMA automaton.

Using Duran et al. (2020) as a sensor system, the energy consumption is 190pJ/cycle at 10MHz in active mode. With AES in software, the system has an energy efficiency of 50900pJ/bit. If the AES is implemented in hardware, the sensor system would consume 2142pJ/bit in push-pull mode and 92.77pJ/bit in DMA mode. The impact of AES in hardware is about 550 times better against software for sensor systems in active mode.

Table 5
Results for the execution of the AES-256 in the SoC with the proposed acceleration.

	Perf [Mbps]	**Cells [NAND]**	**Power* [μW]**	**Energy** [pJ/bit]**
Pure-software	0.011	**76334**	**30.4**	3351.1
SW Sbox tbl	0.024	78642	32.6	1518.2
HW Push	2.67	132375	51.4	22.2
HW w/DMA	**6.16**	141630	51.9	**9.7**

** Measured at 0.4V, 3MHz. ** Measured with 1kbit data.*

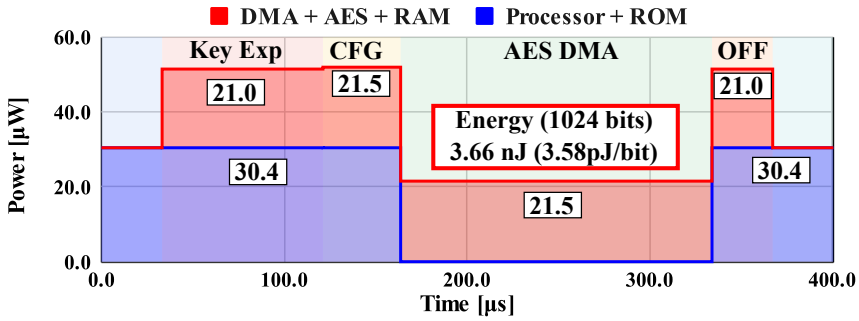

Figure 25. Deconstruction of the energy consumption in the SoC at 3 MHz.

Table 6 summarizes a performance comparison with other works at the minimum energy. Such comparison is also scaled to the 180nm technology using Stillmaker and Baas (2017). To compare the energy in the whole SoC, the power of the Sifive E310 microcontroller is included at 100MHz in 180nm Sifive, Inc. (2021). The work reported in Banerjee et al. (2018) includes its own SoC energy running dhrystone. Without scaling, results in Banerjee et al. (2018) present a similar SoC energy consumption compared to the *Push-Pull* AES scheme. The reported efficiency is better in Deng et al. (2020), but the implementation is intended for high-performance systems. Except for high-performance, this proposal presents a 3x improvement in the SoC energy consumption for the DMA approach.

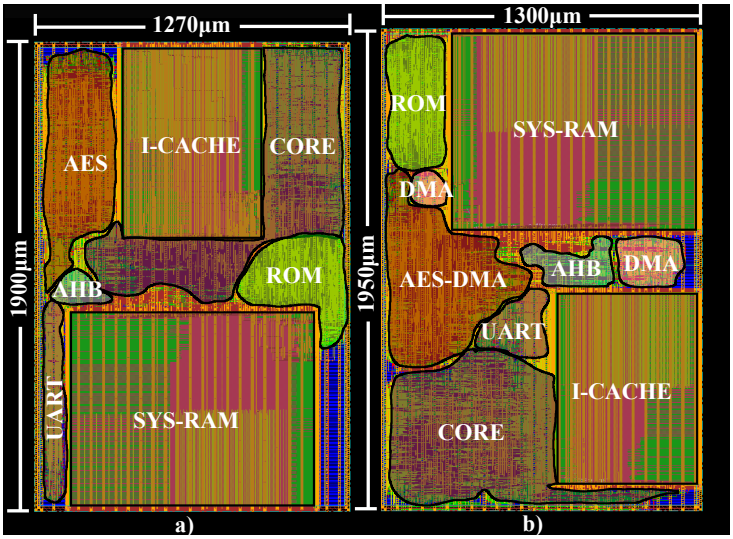

Figure 26. Place and route comparison between the push-pull (a) and the DMA (b) implementations.

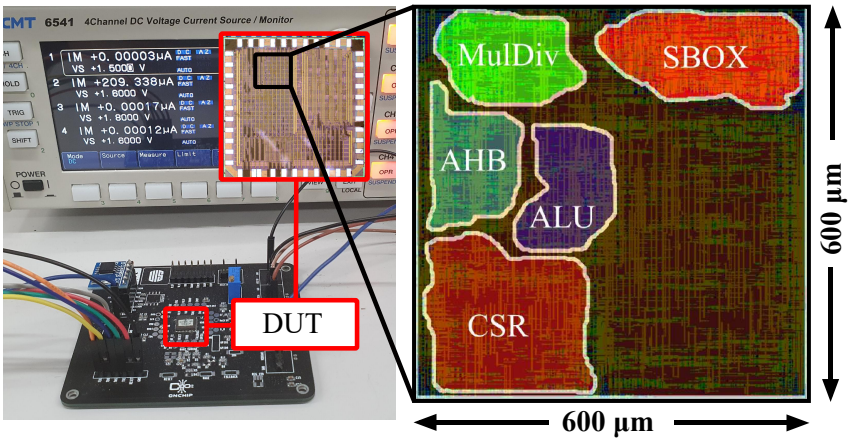

Figure 27. Device under test (DUT) with the micro-photograph of the chip.

Figure 25 depicts a deconstruction of the energy consumption in the SoC, running at 3 MHz. In DMA mode, the key pushing and key expansion last 88 μs, and the DMA configuration last 42.5 μs. The time of calculation of the AES over a block of 1024 bits is 170.4 μs. With total energy of 3.66 nJ, the 3.58 pJ/bit energy consumption previously reported is justified.

Figure 26 presents place and route results for the *Push-Pull* and DMA implementations in 180nm. The macro size slightly increases due to the addition of the DMA and the DMA

automaton in the system. The DMA and the bus increases the system area in a 7% according to table 6. Finally, Figure 27 depicts the measurement setup with die microphotograph details of the prototype fabricated in a 180nm CMOS process. This die includes the custom instruction with the SBOX unit, highlighted in red.

2.1.5. Summary. This section presents different approaches to include an AES core into a RISC-V microcontroller for energy harvesting. An Sbox logic unit is presented first, which can be implemented inside of a RISC-V processor with a custom instruction or as a part of an AES core. With Sbox custom instruction is demonstrated that software implementations' energy consumption decreases from 100 times in pipelined mode and 400 times with a combinational-only design. Regarding the AES hardware core, a register *Push-Pull* is implemented from the AHB bus to handle the AES data, which needs a general-purpose processor to transport the memory. To avoid using the processor, a DMA automaton is introduced that uses a low-power DMA to transport the memory to encrypt data. This design was implemented in a 180nm process node. A minimum of 3.57 pJ/cyc energy consumption if reached for the AES core and 9.7pJ/bit for the whole SoC.

2.2. Olinguito Chip Generator

The integration of digital circuits in security systems is carried away by a chip generator. Circuit generators create RTL and layouts that require a standard cell library to bind logic circuitry and other obfuscation Wang et al. (2009). In addition, chip integration for extensive scale designs requires the positioning of analog modules (floorplanning) and pad ordering (padring generation). This section will discuss the process involved in integrating low-power secure systems using chip generation.

2.2.1. Introduction. Novel circuit generators have been presented Nikolic et al. (2018); Chang et al. (2018); Bachrach et al. (2012); RISC-V Foundation (2019a), but interaction with analog instances and external chip signals is a problem yet not addressed. Design of floorplan, domain region definition, and testing platforms are time-consuming tasks that decrease design performance and opens windows of errors to signal integrity, especially in analog instances. Padring generation is also a time-consuming task because most of the designs for consumer electronics need to be aware of off-chip net interconnections due to PCB or standardized package specifications. Padring creation is often associated with floorplanning, where initial placement and interaction between analog instances and pads is a concern yet to be addressed.

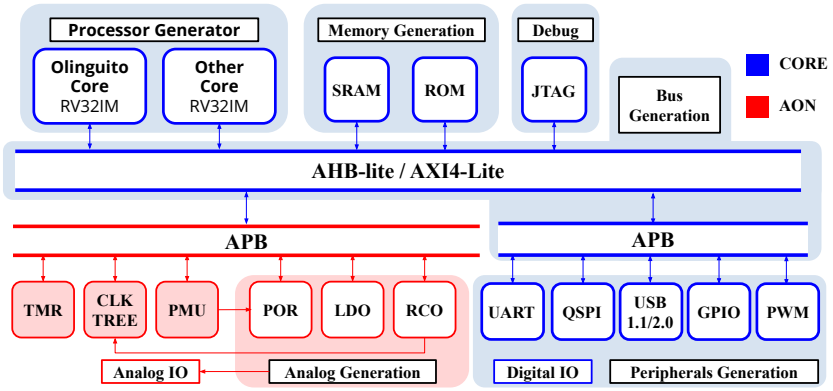

Figure 28. Configurable general architecture.

2.2.2. Chip Generation. A chip generator, Olinguito, is presented. This generator is based on the Chisel library Bachrach et al. (2012). The generator implements a RISC-V processor with peripherals in a similar way of RISC-V Foundation (2019a). Figure 28 presents the architecture of the system-on-a-chip (SoC) which can be generated. Olinguito will implement several RISC-V processors via the processor generator. The RAM and ROM are provided

via a memory generator. Several core-domain peripherals (DMA, USB, GPIO, UART, etc.) are integrated using several peripheral generators, and expose Digital IO according to the peripheral. Peripherals for power management (POR, LDO, RCO, etc.) are controlled by a power management unit (PMU) in an always-on domain (AON). Analog implementations also will expose IO signals. The device can be debugged using a JTAG interface.

The flow to integrate this chip generator is shown in Figure 29. Olinguito outputs an RTL and a series of tests to be executed using formal verification. The formal verification engine can export the necessary interfaces and outputs Symbiotic EDA (2019). Additional to the verification, this generator creates specifications for padring generation. According to the specifications and configurations of the implemented circuits, libraries of padring generation are used to carry the positioning of analog circuits. These outputs are used in a VLSI flow to do the chip generation.

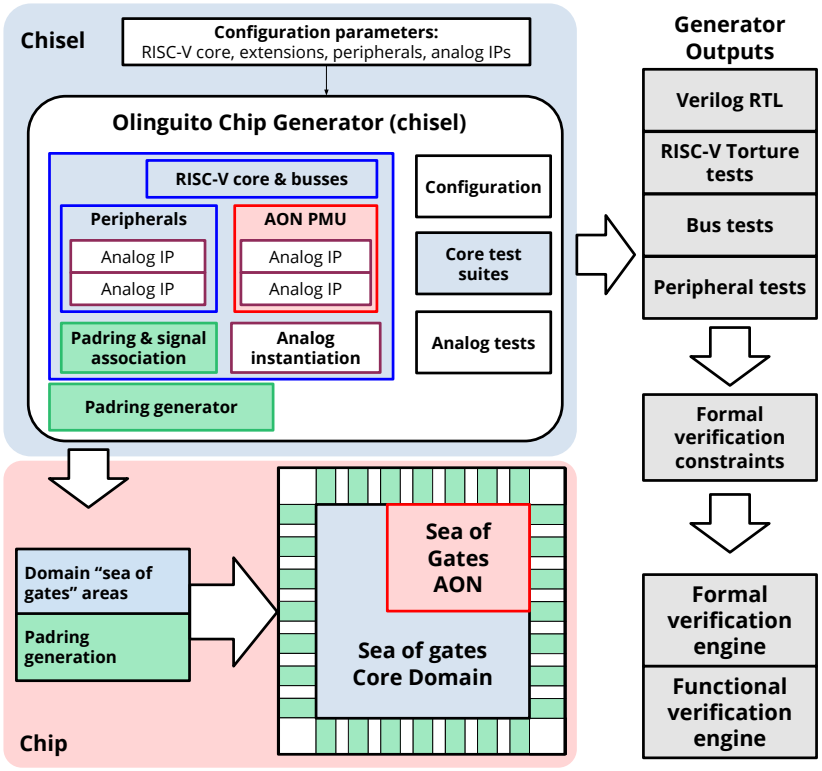

Figure 29. Generation flow platform.

Figure 30 presents the process of design for manufacturing a chip using the chip generator. A series of configurations and platform connections conform the VLSI chips which adds or subtracts features of the system. The chip can first be mapped into an FPGA where it can be verified in a list of FPGA boards. The design is not limited to the FPGAs currently supported, and more types of boards can be added if necessary. After verifying the system in FPGA, the chip can go through an EDA flow which places, routes, and verifies the digital and analog circuits. From this point the chip generator can output emulation test cases for softcore simulation in Verilog. The chip can be finally tested using functional simulations in both post-layout netlists in the case of digital circuits, and transistor simulation if necessary for the analog circuits.

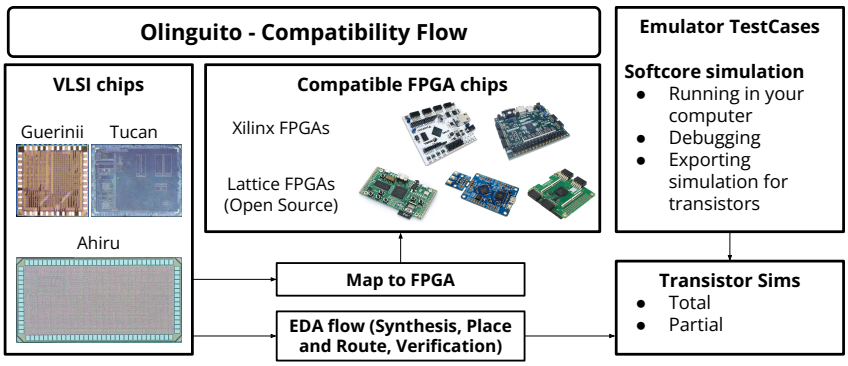

Figure 30. Flow for designing one of the chips in Olinguito.

Any peripheral might require pads to connect external signals to the core or the AON domains. This generation is automatically carried on by the chip generator using a padring generator. The designer interacts with a pad library extracted from the technology. A package library can be added to make the design aware of signal definitions in the package or the PCB. Figure 31 presents the current padring generation flow. The pads are collected by demand, then placed in signal order according to constraints from a configuration file, from a fixed package or from a PCB design. Automatic pad positioning has been proposed for a flip-chip process, where the padring is a swarm of dots in a matrix layout Brist and Park (2015); Park (2010); Lee and Chang (2012).

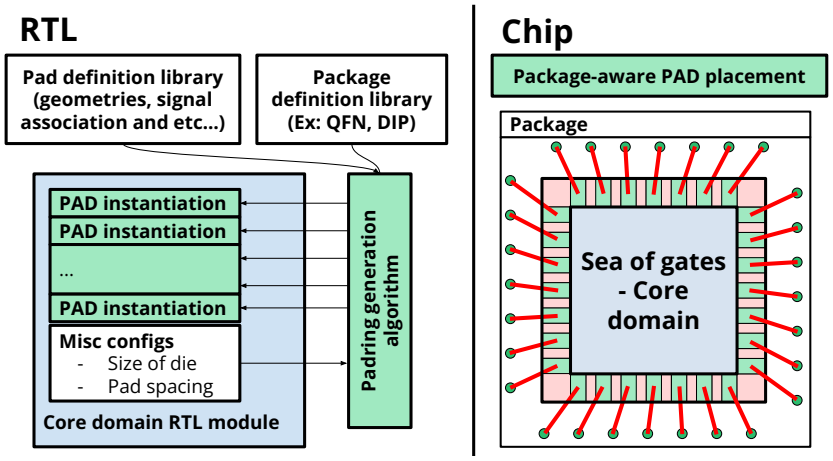

Figure 31. Padring generation flow.

2.2.3. Always-on Domain. The proposed PMU logic inside the AON domain integrates programmable state machines to control sleep modes. A BOD notifies the PMU core algorithm to trigger sleep mode sequences depending on the supply voltage level. Fig. 32 illustrates these notifications where graphs 1-4 show different types of battery events. These events compare the supply voltage behavior (blue) with a threshold level from the BOD (red) set by the PMU algorithm. Event (1) happens when the battery-level discharges and triggers a brown-out detection. The PMU lowers the BOD threshold in the algorithm and waits for a voltage rise event (2) while performing energy harvesting. Throughout these events, the MCU enters sleep mode. If it does not raise the supply voltage above the lowered threshold during a timeout, the PMU algorithm commands the MCU to enter deep-sleep mode. Once the desired voltage level is reached, the BOD threshold is set to a higher value accordingly. In this state, the PMU waits for a rise in the supply voltage above the maximum threshold (3). The PMU allows active mode execution once a supply voltage rises to a safe supply value. In active mode, the supply voltage is measured using the maximum BOD threshold in order to keep alert for a

voltage drop due to an additional loading (4).

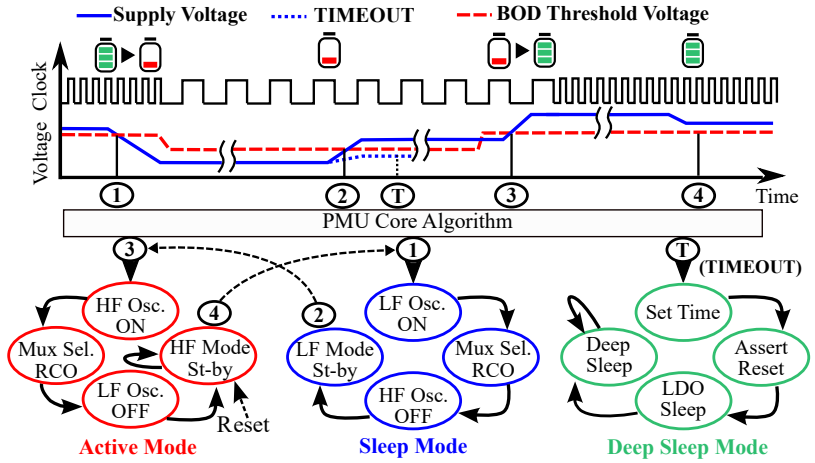

Figure 32. PMU brown-out notification handling scheme.

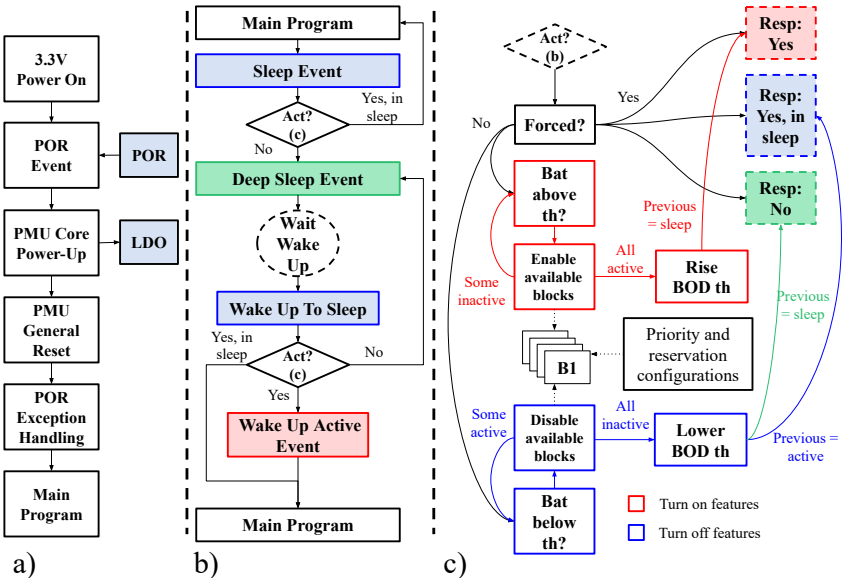

Figure 33. Power management unit main algorithm: a) Initialization algorithm; b) Sleep and deep-sleep operation; c) Block diagram of sleep events.

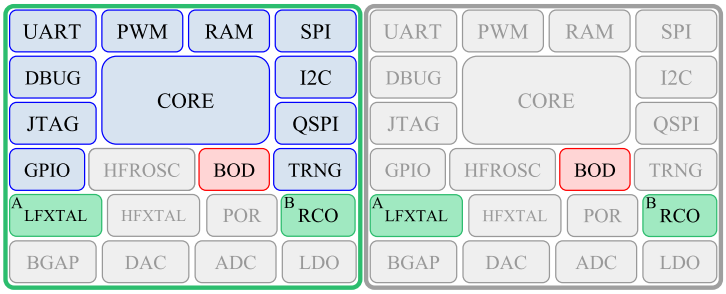

Figure 34. Block throttling in sleep and deep-sleep operation modes.

Fig. 33 describes the PMU algorithm. When powering up with a 3.3V, the POR issues an event to perform a PMU initialization. The initialization turns LDO on and triggers general resets and exceptions to the core to execute the main program (Fig. 33a). The main program can start the event-driven algorithm from the PMU (Fig. 33b). The main program enables blocks and triggers active mode if the battery level rises above the BOD threshold. Although the PMU regularly drives the power operation, the main program may force operation modes and reserve blocks if necessary. The PMU may trigger operation mode transitions according to the battery supervision of an integrated BOD. The PMU logic has the potential to choose to keep running the main program in sleep mode or to trigger a deep-sleep transition (Fig. 33c). This logic may also set the BOD threshold voltage and withdraw blocks to user-prioritized blocks. In cases where all available blocks are withdrawn, the PMU decreases BOD threshold down to values that require to switch to sleep or deep-sleep mode with selected clock sources and user-prioritized blocks (Fig. 34).

2.2.4. Results. Figure 35 shows a chip microphotograph manufactured in a 180 nm process node created by the Olinguito chip generator. The floorplanning of analog circuits and were partially generated by the chip generator. Areas where the standard cells should go for power domains are still partially manual. In the case of 1.6x1.6 mm implementations,

the designed area for AON is always fixed, and the remainder is assigned for regular digital circuitry. The positioning of the pads are constrained according to this chip design, generating a total of 44 pads.

Figure 36 shows another chip microphotograph manufactured in a 65nm process node. The floorplanning is similar as the chip presented in figure 35 being valid only for chip spaces of 2x1 mm. The AON this time is limited as a only-digital circuit which contains control of a digital PMU that controls an external power source. The padring spans a total of 94 pads, which the Olinguito system actually controls 43. The remaining 51 pads are part of a 8-bit processor which the chip generator also interfaces.

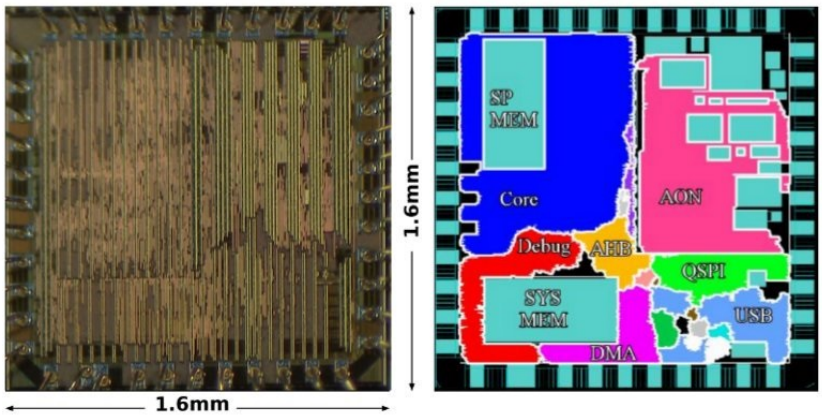

Figure 35. Guerinii chip implemented in a 180nm technology node.

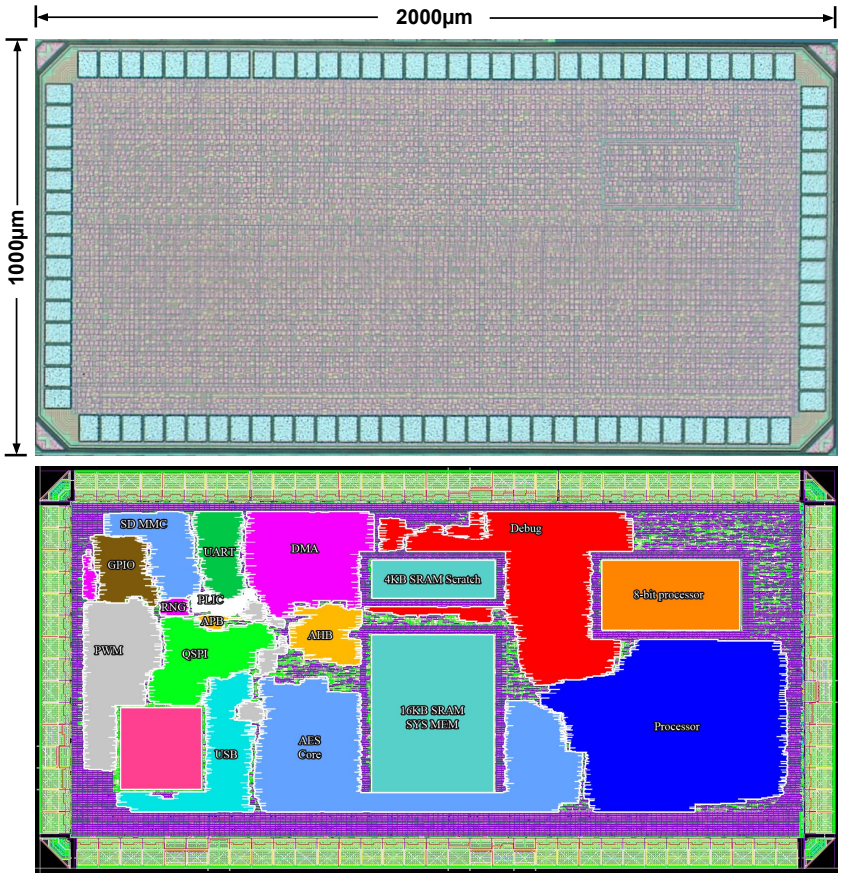

Figure 36. Ahiru chip implemented in a 65nm technology node.

2.2.5. Summary. This section describes a chip generator, Olinguito, that is based in the Chisel library for generation of low-power and low-energy systems for security. The system integrates several RISC-V processors with AHB and APB buses. Peripherals such as RAM, ROM, GPIO, SPI, USB, UART and PWM are attached to the buses. An special always-on domain is available for low-energy tasks such as power modes, sleep and wake-up procedures, and possible power measurements if the analog peripherals are present. The intractability of the analog peripherals are managed by a power management unit (PMU) that is controlled by a programmable finite-state machine. This generator can export constraints about the analog circuits for further integration in EDA tools. Furthermore, the connections to the chip

pads are managed by a padring generator which exposes the positions of the pads according to the package constraints. The proof of concept is demostrated in two different chips in 180nm and 65nm.

2.3. Functional and Formal Verification

This section describes the implementation of verification schemes using both functional and formal approaches. Processors from the Olinguito chip generator can be verified using the RISC-V formal interface or a RISC-V simulation-based verification output. Both worlds will be implemented and compared to detect possible failures in the processor.

2.3.1. Introduction. The instruction set architecture (ISA) is the main specification of any processor implementation, and it contains detailed information about the instruction execution, the data registers, and the memory interactions through an external bus. Processors are commonly described by using a hardware description language in a register transfer level (RTL). RTL implementations must be able to execute any instruction specified in the ISA. To verify the processor against the ISA, designers and certifiers run a set of tests to verify the processor by using hardware simulation. Processor architectures must be verified to give solidness over the instruction execution before and after the circuit is synthesized in a technology node.

Running an extensive software test set inside a hardware description simulation is a common approach to perform ISA verification in processors. This test set is compared to an execution model, which checks the execution results for each instruction specified in the ISA. However, guaranteeing an accurate execution verification model to cover all processor states is challenging. Moreover, the processor model might not be accurate enough to perform comparisons against the ISA specification. A RISC-V case-of-study conduits comparisons to sim-

ulation models implementing an evolutionary framework with a coverage-based optimization function Schiavone et al. (2018). In order to verify the instruction simulator, coverage-guided fuzzing may be performed Herdt et al. (2019). Authors in Corno et al. (2005), apply a different approach by using evolutionary algorithms to compare the execution of two different processors.

Although a coverage-based optimization aims to envelop all the processor states, formal verification can explore all possible states, according to specifications, using assumptions and checking errors using assertions Symbiotic EDA (2019). For Arm processors, a specification language named "architecture specification language" (ASL) allows any Arm processor to be scaled in the RTL environment Reid et al. (2016). Arm verifies its processors in any architectural implementation by extending the ASL language to include formal properties. A RISC-V formal verification framework allows the implementation of verification intellectual property (IP) specified in formal verification assertions Symbiotic EDA (2019). The test framework may be suited to several architectural implementations through a *RISC-V formal* interface by adding internal processor states.

In this section, a verification scheme is presented that combines two functional platforms. A μGP-based RISC-V program generator is implemented which uses a genetic algorithm to find the best program that covers most of the processor states. For every individual generated from the genetic algorithm, a comparison is performed against a simulation program as a golden model. Such simulation can be performed by any RISC-V simulator, but for this proposal, the *Spike* simulator was chosen following the suggestion by the RISC-V foundation RISC-V Foundation (2019b). In addition to the program generator, a second approach based on

formal verification is also utilized. A set of verification IP are attached with formal properties named *RISC-V formal* Symbiotic EDA (2019). This IP compares internal states of the processor to a formal specification derived from the RISC-V ISA. The detection of induced errors is shown conducted in both simulation and formal approaches. The framework was capable of detecting a consistent misunderstanding in the ISA for asynchronous processor interruptions.

2.3.2. Using *Spike* and *µGP*. The simulation-based verification framework was established on *µGP* to generate effective test programs Corno et al. (2005). *µGP* stands on a test program generation algorithm, handling an evolutionary core. Results from test programs were compared which run in both the processor and a RISC-V instruction simulator. The RISC-V foundation provides *Spike* RISC-V Foundation (2019b), the golden RISC-V ISA simulator within a functional model for instruction execution. *Spike* is compliant with the RV32I or RV64I base ISA, supporting M, A, F, D, and Q extensions among extra features.

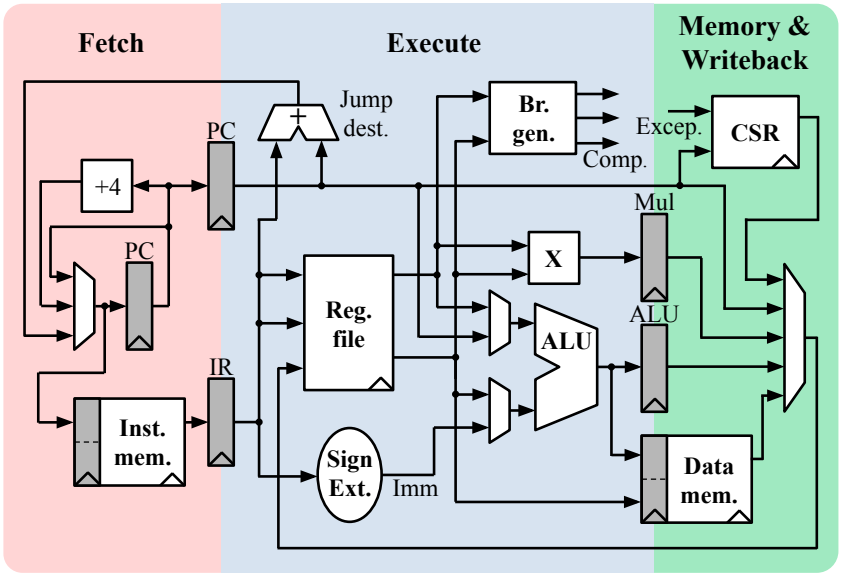

Figure 37. Microarchitecture of the processor under verification (PUV). PUV comprise 3-stage single issue in order pipeline.

An in-house 32-bit RISC-V ISA based processor was designed in the Olinguito chip generator, which is the processor under verification (PUV) in this framework. The PUV is an ultra-low power consumption processor intended for low-performance tasks, comprises a three-stage pipeline single-issue in-order, and is compatible with I, M, and C RISC-V extensions. Figure 37 describes the microarchitecture of the PUV.

Figure 38 summarizes the methodology for simulation-based verification used in this proposal. μGP automatically generates a set of programs called individuals by using a constrained genetic algorithm. According to the feedback (the fitness function computed by the collecting coverage metrics), the best programs are improved, and μGP aims to increase the covered code while reaching more states on the PUV hardware. μGP is formally described in Sanchez et al. (2011). In the current experiments, μGP uses the evolutionary standard setup provided by the tool.

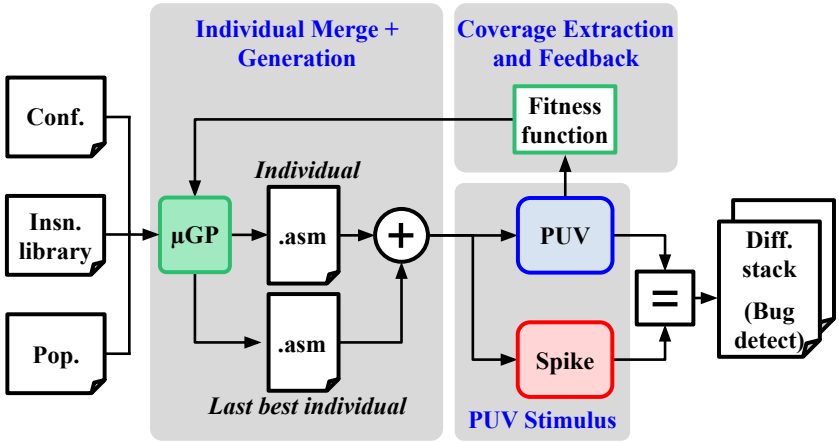

Figure 38. Test generation and simulation-based verification methodology for the PUV a RV32IM based processor.

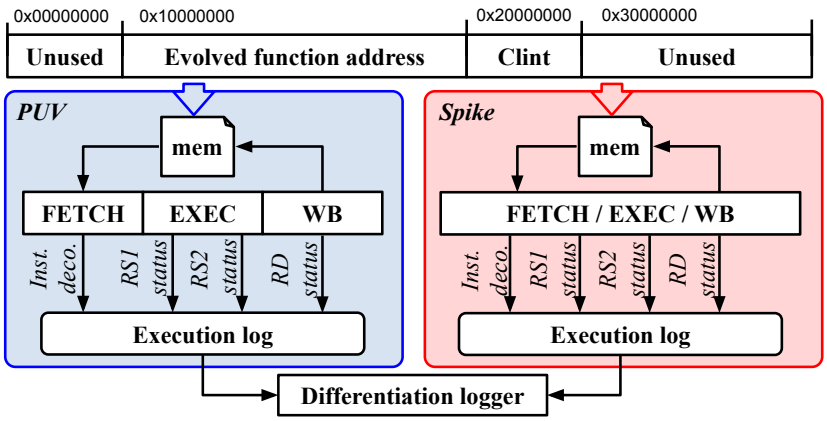

Figure 39. Parallel execution on the PUV and *Spike* using the same memory model. The execution is logged from the states of each model.

The framework loads the programs directly into the simulated memory. The PUV and *Spike* execute the individuals simultaneously, and the execution states are compared to find discrepancies in the ISA interpretation. Figure 39 shows the state comparison using an execution log. The PUV simulation logs the execution by each clock cycle. This log is compared on valid states with the execution in the *Spike* simulator. The execution state is compared in the program counter and the register states from all the pipeline stages.

Coverage metrics were extracted from the execution of the test program in the PUV using commercial tools. In this case, the NCSim simulator and the incisive comprehensive coverage (ICCR) were used, the latter a code coverage analysis tool, as described in Figure 40. Metrics coverage data is related to the percentage of executable statements during the simulation, the percentage of the Boolean expression evaluated, among others. Finally, the extracted coverage is used to compute the fitness value that is fed back to μGP, guiding the test program generation until one of the stop conditions triggers.

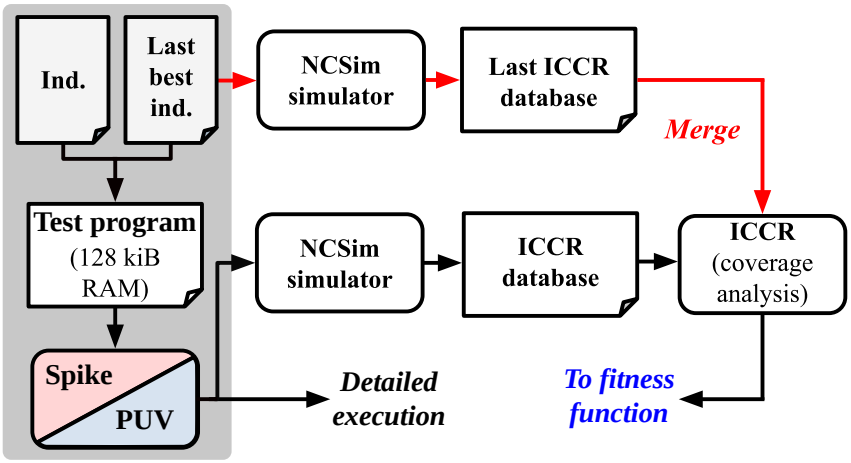

Figure 40. Simplified block diagram of the code coverage extraction processes using Cadences tools.

2.3.3. Using *RISC-V formal*. *RISC-V formal* is a non-invasive processor-independent formal verification description of RISC-V based processors Symbiotic EDA (2019). It consists of a processor-independent formal description of the RISC-V ISA and the specification for the *RISC-V formal* interface (RVFI) among additional features. This interface transmits the execution status of the current instruction when the calculation is ready for all the internal registers. The current version of the interface allows us to verify against general-purpose registers, the program counter (PC), fetched instructions, a generalized memory interface, control and status registers (CSR), and some internal processor flags.

Figure 41 exhibits a simplified diagram of the interconnection using the *RISC-V formal* interface (RFVI) to verify a processor formally. The RVFI must carry the final state of execution of any instruction to a formal environment. In the processor, all the necessary signals were took from any stage of the processor architecture to the write-back stage (MEM & WB). In this way, the formal properties inside the *RISC-V formal* environment compare the instructions specifications against the internal finish-states of the processor.

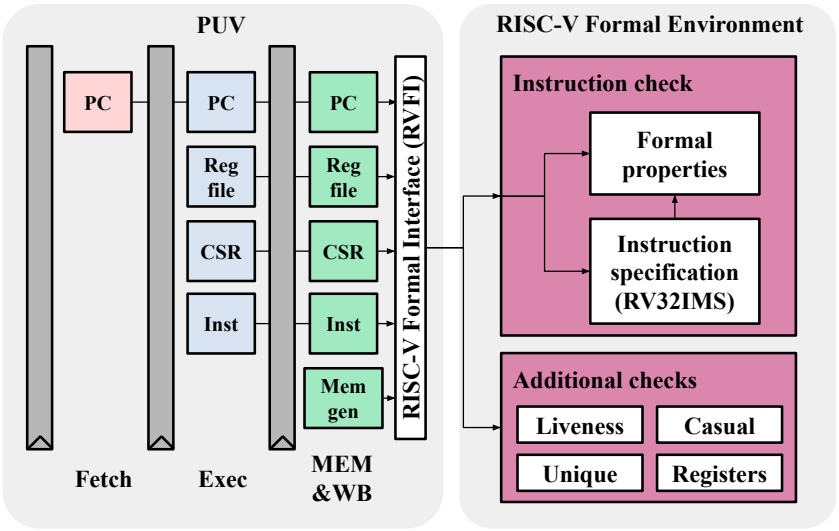

Figure 41. Simplified block diagram of an interconnection using a *RISC-V formal* interface.

The processor formal verification scheme performs tests over the CSRs, but there are no tests available for the behavior of a RISC-V processor based on external interruptions. The RISC-V specification has three kinds of interruptions: external, software and timer. Depending on the processor mode, different flags should be triggered, and the processor should jump to the exception vector in the CSR. We observed that there is no precise specification about the interruption flag sequence to change the internal processor flags and the PC. Moreover, the RISC-V privileged ISA specification does not specify the execution status of the processor when an external interruption occurs.

2.3.4. Interruption Specification Absence. The RISC-V interruption specification does not give details of how an interruption must be performed Waterman et al. (2014). This description absence is related to the sequence in which a processor performs an interruption. That absence opens up different ways to implement an interruption depending on the core designer. Figure 42 presents a waveform that compares the interruption procedure in the PUV

and *Spike*. Both execute the main program (in blue) until the interruption is triggered. Once the interruption triggers the PUV, its control unit immediately stores the actual PC into the CSR and flushes the execution and write-back stages corresponding to the flushed PC. After storing the PC, the processor sets the PC to the interruption handler code (in red). Later, the last instruction of the trap-handler program reaches the write-back stage, and the PC returns to the stored value in the CSR, as depicted in Figure 43(a).

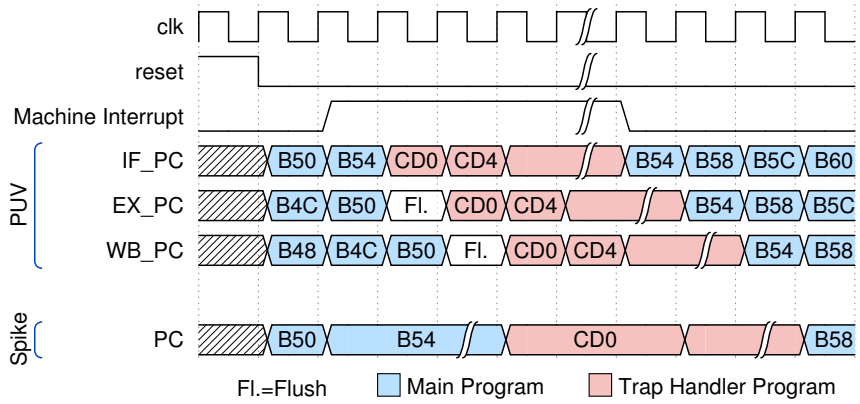

Figure 42. PUV and *Spike* waveforms describing the internal state interaction on the interruption trigger.

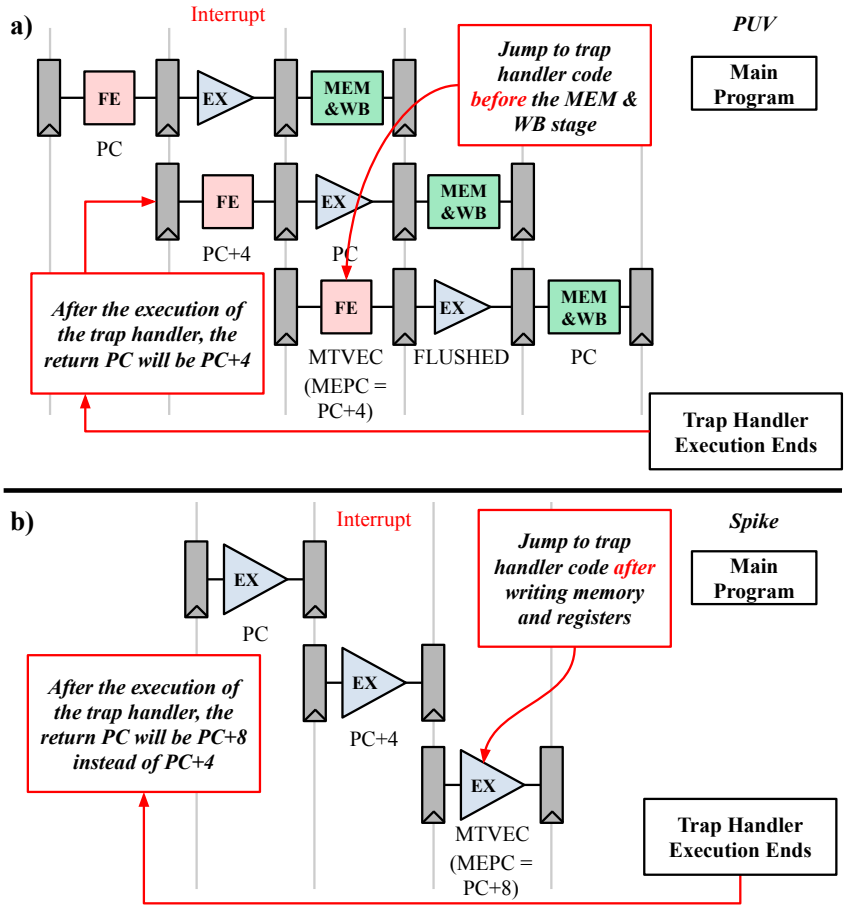

Figure 43. Different interpretations of the RISC-V ISA interruption procedure.

At the lower edge of Figure 42, the *Spike* approach of the interruption sequence is presented. *Spike* waits until the previous instruction (in blue) to the interruption is finished when the interruption occurs and stores the next PC into the CSR. Then, the PC jumps to the interruption handler program (in red). After the trap handler program execution ends, the PC returns to the stored value in the CSR, as shown in Figure 43(b).

In some instances, the *Spike* approach in a real processor implementation may cause a delay in trap handler program execution. For example, in multi-cycle operations such as multiplication, the trap handler program waits to be executed until the multiplication ends. This

strategy is more efficient in the sense that it does not need to flush the pipe. However, it could affect critical applications requiring a fast response to an interruption. The PUV approach reduces the execution latency of an interruption trap handler program. Furthermore, it avoids inconsistencies with instructions that may change the internal status of the memory or internal registers (such as CSRs and system bus requests).

2.3.5. The Best of Both Domains. The process of verification of the implemented RISC-V processor is combined using the Spike and μGP simulation comparison, and the description of the RISC-V specification using formal verification with Yosys Wolf (2022). The internal states of the processor were compared with the verification environment. Each one of the verification covers most of the cases according to the processor architecture. To reach the most cases possible, μGP maximizes the coverage metrics, and Yosys evaluates the assumptions and assertions in each simulation step using boolean satisfiability (SAT).

Some flaws inside the datapath of the PUV were introduced to verify the correct operation of the frameworks. Table 7 presents the detection over the μGP and *RISC-V formal* of different inserted errors inside the processor. The ALU misbehavior was tested by changing one bit to the final result, affecting the multiplexer order, and adding a segmentation stage. This misbehavior is detected in both frameworks– more than half of the μGP tests. The faulty comparison introduces a misinterpretation of the signed comparison by using the 30th bit as the sign instead of the 31st bit. RISC-V torture does not detect this flaw, and the detection probability on the simulation-based framework is minimal, making formal verification suitable to find these kinds of errors. The data hazard removes detection when writing a register to be used in the next instruction. Software tortures are more suitable for data hazard detection. Finally, it is

relevant to highlight that the interruption execution issue described in section III was detected by the proposed simulation-based approach.

Table 8 provides an implementation cost for μGP and *RISC-V formal*. μGP needs a simulation environment for the processor to be implemented, which uses a log to display the current status of the processor. *RISC-V formal* has a wrapper environment where the RVFI is required. μGP requires more software binding due to the need to push programs directly in RAM, and only a per-cycle hardware logging. *RISC-V formal* needs more hardware binding to connect and transform all signals to the RVFI. An additional setup is needed for the μGP in contrast with the *RISC-V formal*. The execution time is configurable for μGP, lasting 90 minutes for 450 individuals. The runtime for the 51 tests for the formal approach always run for 45 minutes using multi-threads.

2.3.6. Summary. A verification framework was presented combining two different verification approaches. The first one excites PUV and a golden RISC-V ISA simulator with a set of automatically generated tortures. Using a genetic algorithm called μGP, μGP drives the generation of testing programs following an maximization process over the coverage metrics, then a comparison is performed using the internal states of the processor with a execution model. The latter, *RISC-V formal*, defines a set of formal properties that follow the RISC-V ISA specification, through using open source formal verification tools, the behavior for each instruction is verified on the PUV against the set of formal properties.

If the verification implementation is accurate according to the specification, these two approaches must thoroughly verify the instruction set of RISC-V based processors. Unfortunately, most of the verification errors may be caused by insufficient coverage of all the cases.

By using Spike (the golden model) chose to compare in the framework, exposed in Fig. 38 detects at most one out of 450 executed tests for comparator flaws and data hazards. The formal properties specified in the RTL help to detect certain behavioral flaws according to the ISA specification. Thanks to the combination of the verification schemes, an absence was detected in the processor interruption state change, which the RISC-V ISA does not specify.

Table 6
Performance comparison.

	Tech	Freq [MHz]	Power [mW]	Gate Count	Energy [pJ/bit]	In SoC [pJ/bit]
Wang and Ha (2013)	FPGA	320	3770	N/A	269.64	1348.2†
Kundi et al. (2020)	FPGA	264	184	N/A	11.14	56.8†
Choi et al. (2020)	65nm 1.1V	100	5.8	6187	35.73 (822.6)	207.6† (707.9)
Deng et al. (2020)	65nm 1.1V	500	625	-	9.76 (38.6)	11.87† (47.0)
Mathew et al. (2014)	22nm 0.4V	100*	0.5*	1947*	12.88* (1573.3)	35.4† (4329.1)
Sayilar and Chiou (2014)	45nm 0.8V	1000	6179	-	48.27 (463.8)	49.1† (472.2)
Banerjee et al. (2018)	65nm 0.8V	16	-	-	5.56 (43.1)	92.2+ (1345.1)
Marshall et al. (2020)	28nm 0.8V	202	-	-	28.8 (419.7)	144.0† (2098.8)
Push	180nm 0.4V	3	0.0216	6553	**8.00****	**22.2****
		100	3.21		**40.6****	**154.8****
DMA	180nm 1.1V	3	0.0221	6791	**3.58****	**9.7****
		100	3.21		**18.3****	**67.9****

() Scaled to 180nm, 1.1V Stillmaker and Baas (2017) +40.36pJ/cyc SoC, 6.21nJ AES, 1kbit Banerjee et al. (2018)

† Sifive E310 106mW@100MHz scaled penalty added for SoC Sifive, Inc. (2021).

** Only AES-128 Encryption. ** AES-256 with 1kbit of data.*

Table 7
Error detection in the verification models.

Inserted error	μGP-Spike (450 Indiv.)	RISC-V formal (RV32IM)
ALU misbehaviour	256 Detected	Detected
Wrong comparison	1 Detected	Detected
Data hazard	1 Detected	Not detected
Interruption execution	306 Detected	Not detected

Table 8
Implementation costs between μGP and RISC-V formal

	μGP-Spike	**RISC-V formal**
Processor output	Output simulation log.	RVFI RTL port.
Testbench	RAM capability to insert external programs.	Provided. Connect RVFI.
Scripts	Adaptation needed. Build assembly and format to simulated memory.	Provided. Enable or disable flags for verification.
Exec. Time Intel-i7 9750	$\sim$90m for 450ind. 1-core	$\sim$45m for RV32IM 8-cores

3. Methods for Memory Security

In previous chapters, standard cells generators and security in formally verified system-on-chips were presented. The last focus of this book is to address the issue of *RAM access*. The processor mainly controls access to the memory. The main problem is that if the processor executes unauthorized code or the memory is accessed through debug side channels, the memory contents are exposed.

If memory security hardware exists, the overhead of such circuits is usually not significant to avoid additional timing and power constraints. The security hardware itself should be protected against reverse engineering to avoid the problem of *imaging reversing*. In chapter 1, different standard cells generators were described. Those generators must be used to perform additional layout obfuscation over the low-overhead digital circuits implemented in the memory security.

This chapter presents an additional layer of security-focused on memory obfuscation and layout obfuscation. An oblivious obfuscator with low overhead is presented to scramble the memory contents. Because low-overhead circuits tend to be small and easily identified, a layout obfuscation is also introduced. The methodology obfuscates the oblivious obfuscator layout inside the memory using several generated standard cells. The original algorithms presented in chapter 1 were modified to extract different placement results for each netlist of the standard cell library. Each placement is then routed, causing a drastic change in layout for each cell. This layout catalog is randomly replaced in the post-synthesis netlist to obfuscate the circuit.

3.1. Introduction

Security in microcontrollers can cause large overheads in area and power. For low-power applications like sensors, implementing security usually involves heavy penalties, making the system unfeasible. For high-end processors running an operative system, the software implements access control and memory paging to secure memory operations Bovet and Cesati (2005). For this approach, the memory paging takes place in sectors around 1KB, which is ideal for memory implementations in megabytes. In a microcontroller, memory protection through pages cannot be executed in this way, meaning software protection is usually not viable.

Some alternatives exist for small-scale applications on the hardware side that offer low overhead. Liang *et al.* presents memory protection in hardware for lightweight systems but only restricts the access of data in sectors. Liang et al. (2016). The custom SRAM cell implies modifying or creating SRAM generator which is usually unavailable for users. Lu *et al.* offers an efficient memory encryption for off-chip implementations using AES-GCM Lu et al. (2015b). ObfusMem presents a scheme for transaction encryption (address, request, and data) with the configuration being oblivious Awad et al. (2017). The RAM and the agent making the requests uses HMAC to authenticate the transactions. Yang *et al.* proposes memory protection for execute-only memories, where the contents have been encrypted off-chip Yang et al. (2005). Kumar *et al.* depicts a processor modification that checks all memory instructions, calls, and returns to avoid faulty software exploits Kumar et al. (2007b). This idea is further explored to check runtime and memory saves in sensor nodes Kumar et al. (2007a). Krishnakumar *et al.* presents protection against illegal memory accesses focused on memory objects where each has associated bounds. Krishnakumar et al. (2018). Zheng-Huang implements a redundant array of

independent memories, which offers protection for selective memory Zheng and Huang (2017).

Several types of protection can be implemented to either encrypt or prevent access to the memory. However, with the understanding of the internals of a system, memory protection can be defeated. For example, in Branco-Gueron's work, corrupting the memory can lead to memory access for systems with DMA blocking and memory encryption Branco and Gueron (2016). Obfuscation in memory contents and layout can resist attacks to prevent system identification. This chapter presents a system-on-chip (SoC) for low-power and low-area applications with obfuscation features. This SoC features security peripherals such as a TRNG, an AES core for low-power Duran and Roa (2021a), and an SBOX custom instruction. To secure algorithms executing security tasks, we implemented a low-overhead oblivious obfuscator as security in the SRAM controller. This controller is capable of scrambling the addresses coming from an AHB bus. The overhead is around 2.7% in area and 8% in power compared to a 4KB RAM. Because the overhead is low in area, the oblivious obfuscator and the masked SBOX were obfuscated in layout using the method described in Gomez et al. (2019). The 4-levels of standard cells were generated using a modified placement algorithm of the chapter 1, together with routing algorithms. The initial success of standard cell recognition is around 18.7% without prior knowledge of the several standard cells implemented. This SoC was implemented in a 0.2 μm technology with 1.3 mm^2 of area and 16 mW @ 10 MHz of power for the digital part.

3.2. Memory Obfuscation

Figure 44 shows the architecture of the system. The SoC is composed of a low-power RISC-V processor with RV32IM ISA Waterman et al. (2014). AHB and APB buses connect the processor with 32-bit transaction support. The 4KB SRAM memory is connected to the

AHB bus and secured with an oblivious obfuscator. The values for the obfuscator inside the SRAM come from a true-random number generator (TRNG) based on ring oscillators Serrano et al. (2021). The system supports UART for communications and SPI for running the initial bootloader and user-level programs. The debug module is attached to the AHB bus, which can stop the processor and examine memory regions. The AES-128/256 peripheral performs the encryption of sensor measurements and supports direct-access memory (DMA) for low-energy consumption Duran and Roa (2021a). A low-power DMA controller is attached to the AHB bus to support direct memory access to the AES peripheral Morales et al. (2019). Additionally, the processor has a custom [I]SBOX instruction to perform software AES encryption or decryption faster.

Specific critical security circuits are obfuscated in layout to ensure the system's security. In figure 44, everything labeled in red is obfuscated in layout to hide critical information in the system. In the case of the AES-128/256 and the [I]SBOX custom instruction, the SBOX unit contains a masking method that requires protection from reverse engineering. Furthermore, the oblivious obfuscator inside the RAM could have constant functions that are small enough to not infer in timing overheads. Any small security circuit inserted in the memory must be protected to prevent systematic reverse engineering. The method used for obfuscation in the layout is explained further in this chapter.

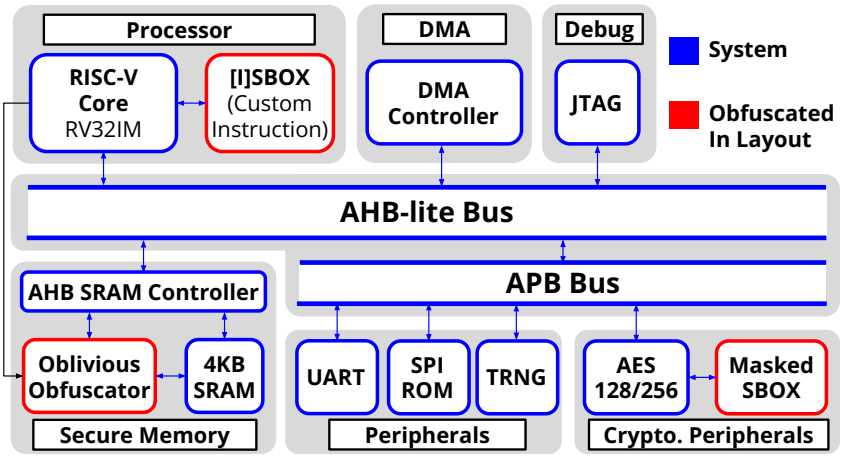

Figure 44. General architecture of the implemented system-on-chip.

The implemented architecture of the SRAM oblivious obfuscator is shown in figure 45. The oblivious obfuscator protects the data according to the execution mode of the RISC-V processor. The M-mode of the processor activates the circuit through the multiplexer before the address line in the SRAM. This implementation obfuscates the addresses by using SBOX as a 1-to-1 table translator. Because the SBOX supports 8-bit data, the translator can obfuscate 1KB of RAM, leaving 3KB of RAM unprotected for user-level usage. The address is translated first by an SBOX, then is XORed through a tag register. After the XOR, the address is translated again by another SBOX. The tag register is written through a register router whose value is fetched from the TRNG before using the obfuscated region of the SRAM. This register is write-only, and the register is reset when the execution mode is changed to user mode. Another layer of security can be added to the data lines, such as 1-cycle encryption, but this approach is out of this scope.

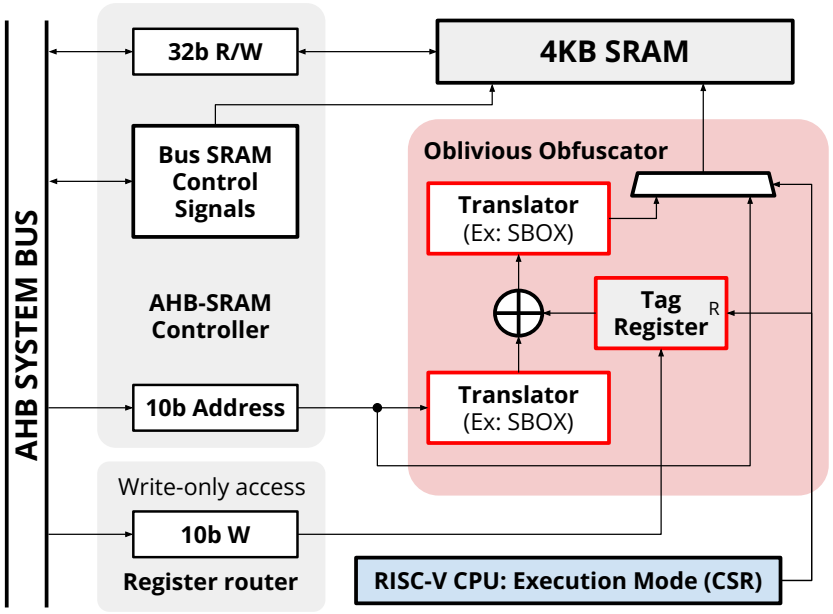

Figure 45. Oblivious obfuscation architecture of the SRAM controller.

The intended protection for this system is for bounding security around algorithms that handle critical data such as keys. In figure 46 we show a possible scenario where data from a sensor is measured and encrypted to be sent through a channel such as UART. Past the point of the bootloader execution, the program starts in user mode, pictured in blue. The measurement data is stored in *A* and is encrypted in *B* using AES. The *Encrypt* function prepares a reference *C* which can be accessed by machine mode, pictured in red. The function interrupts in software (MSIP) to switch the processor to machine mode. The *MEncrypt* function is then executed after handling the interrupt. The oblivious obfuscator in the SRAM is active but still does not have a tag stored. Before performing the actual AES encryption, the software fetches a value from the TRNG and stores it into the tag register (*TagReg[SRAM]*). Once the tag is set, the machine mode program can use the obfuscated 1KB memory for executing the AES algorithm either in software or hardware. Critical parts like key getting and salting can be done inside this

safe space. The machine mode program stack is also allocated in the obfuscated SRAM. The DMA is also capable of writing to this area according to figure 44 from the AHB bus. After the encryption is done, the machine mode returns the interruption to continue the execution in user mode. The user can send the ciphertext *B* through the channel.

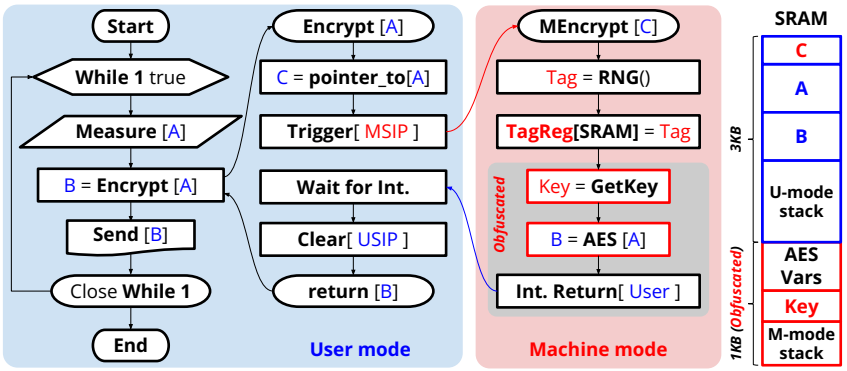

Figure 46. Example flowchart where the secure SRAM is bounded around an encryption AES execution.

3.3. Layout Obfuscation

We implemented a layout obfuscation technique based on a previous work involving generated standard cells Gomez et al. (2019). In this previous work, by randomizing the standard cells from a catalog of different layouts, a digital circuit could be obfuscated with minor overhead in timing, power, and area. The first step to obfuscate a digital circuit is to generate the standard cell library for obfuscation.

From a given netlist, a circuit can have different placement arrangements. By slightly altering the placement of transistors, the routing can significantly change without altering the behavior of the specified netlist. It is possible to output different placements for the same circuit by configuring the placement algorithm. For the PBSAT-based algorithm stated in section 1.1, the next placement can be extracted by constraining the previous successful placement. For

the graph-based algorithm described in section 1.2, reversing the tree and storing several best-solution chains can output several placements Hwang et al. (1990). In chapter 1, the standard cell generators were built with a focus on horizontal net optimization for routing algorithms. The placement algorithm can use this metric to output a series of standard cells with different layouts for the same netlist to output the best number of placements. Algorithm 6 presents a modification of the standard cell layout generator presented in section 1.1 (algorithm 2) to support multiple placements. In short, for a number N of placements generated by the placement algorithm, we try to route them using a maze router. This router can be something simple like prioritized routing or SAT routing involving a series of generated maze routes Ryzhenko and Burns (2012). A new geometry can be generated if the router can successfully connect all the terminals. This approach has demonstrated a great rate of success if the maze router can trace in polysilicon and 2nd level metal. Simple cells such as ties, inverters, fillers, and capacitors cannot use this approach.

Algorithm 6 Standard cell layout generator

```
1:  procedure LAYOUTGEN(netlist,levels)
2:      cells ← Placement(netlist, levels)                    ▷ N placements
3:      for cell in cells do
4:          cell.bins ← Stick(cell)                           ▷ DR constrained bins
5:          cell.stick ← Routing(cell.bins)                   ▷ Maze router
6:          if not exists cell.stick then                     ▷ If can't route
7:              cells ← Placement(netlist, 1)                 ▷ Append 1
8:          else
9:              cell.layout ← Layout(cell.stick)
10:             LIB.cells ← cell                              ▷ Append cell to library
11:         end if
12:     end for
13:     save(LIB)
14: end procedure
```

In this proposal, we generated four layouts per cell. Each complete library is named a *level*. Figure 47 shows some examples of 4-level standard cells generated. In the case of the AND, the different placements differ mainly on the detachment of the output inverter. The alternative placements detach the inverter and iterate the NAND placement around three possible placements.

Further levels will probably detach all the N-P pairs of transistors, increasing the area overhead. In the D-latch's case, the scrambling occurs in the coupled inverters and tri-stated

inverters in the middle of the circuit. The input and the output stage remain similar, but the routing changes drastically in the middle regardless of this similarity. The D-flip-flop shows some exciting results by detaching several chains of transistors. The different iterations of transistor ordering change the layout more heavily. The generated library spans around 18 different types of cells, including NAND, NOR, AND, OR, OR-ANDs, AND-ORs, XOR, Negated XOR, Multiplexers of 2/3/4 inputs, D-Latch, D-Flip-Flop with and without an asynchronous reset, Half-Adder, and Full-Adder.

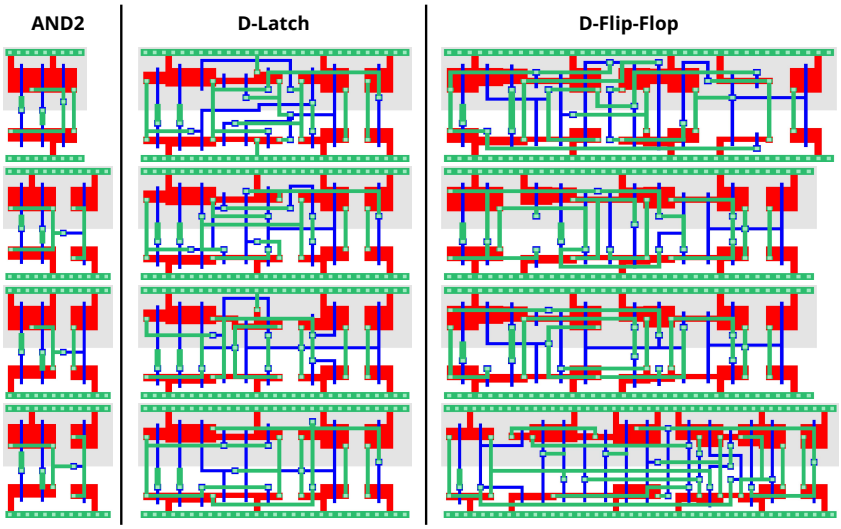

Figure 47. An example of a 4-level standard cell generation for 12-tracks.

In order to verify the impact on non-optimal gates, a characterization of generated cells is performed for different environmental and process conditions, using a library of 12-tracks height cells. The characterization extracts power and timing specifications for typical and worst cases. Finally, the area for all different shapes of the same standard cell remains constant, resulting in no penalties for this specification.

Table 9 depicts performance extraction for the typical case and worst case scenarios. In the worst case, timing performance overhead raises up to a 13% for NAND cell non-optimal

alternatives, whereas timing penalties for inverter variants remains less than 5%. Power performance can have an overhead up to 16% in the case of the NAND2D1. Overheads in non-optimal cells are due to additional routing usage that increases inner cell capacitances, degrading the performance. Besides, the usage of non-optimal placement can result in increased diffusion capacitances. Although this table presents a reduced number of gates, such as the inverter, NOR and NAND gates for one column size, a larger cell library has been used for complex circuits demonstrating experimentally that these three cells are a representative sample of the large set. Average performance indicates that a large timing penalty occurs if non-optimal cells appear inside a critical path which limits the placement to non-critical paths.

Table 9
Basic gates characterization at typical case (TC) and at low voltage supply, slow-slow process corner, and 125C (WC). This characterization is presented as the ratio between the specifications of cells with non-optimal layouts and cells with the optimal ones.

	INVD1							
	A		B		C		D	
CORNER	TC	WC	TC	WC	TC	WC	TC	WC
POWER	1	1	1.04	1.03	1.03	1.03	1.09	1.08
TIMING	1	1	1.01	1.02	1.01	1.02	1.04	1.05
	NAND2D1							
	A		B		C		D	
POWER	1	1	1	1.005	1.15	1.15	1.16	1.16
TIMING	1	1	1	1.004	1.11	1.12	1.12	1.13
	NOR2D1							
SPEC	A		B		C		D	
POWER	1	1	0.99	0.99	1.14	1.14	1.15	1.15
TIMING	1	1	0.99	0.99	1.10	1.12	1.11	1.12
	LND1							
SPEC	A		B		C		D	
POWER	1	1	1.072	1	1.053	1.073	1.019	1.009
TIMING	1	1	0.997	1.004	1.038	1.038	1.012	1.012

With the generated levels of standard cells, we can obfuscate the digital circuit. Algo-

rithm 7 depicts the basic algorithm for obfuscation. This algorithm is applied after the digital synthesis but before the place and route. The algorithm reads a synthesized netlist, and detects all the cells in the first level (*zero level*). The algorithm extracts all the available levels from the library for the implemented cell. We randomly pick a replacement from this catalog of cells and put it into the obfuscated netlist. This obfuscated netlist is then saved, placed, and routed using electronic design automation (EDA) proprietary tools. We can output fully-automated digital obfuscated circuits with minor overheads with this approach.

Algorithm 7 Layout obfuscation algorithm

```
1:  procedure LAYOUTOB(netlist,LIB,levels)
2:      for inst in netlist do                          ▷ Explore all instances
3:          zero ← inst.cell                            ▷ Cell in zero level
4:          cells ← LIB.cells[zero]                     ▷ Get the other cells
5:          cells ← cells.slice(levels)                 ▷ Use up to "levels"
6:          ob ← Random(cells)                          ▷ Pick other cell random
7:          inst.cell = ob                              ▷ Replace zero cell with obs
8:      end for
9:      save(netlist)
10: end procedure
```

3.4. Results

The SoC and the obfuscated circuits were implemented in a 0.2μm technology. Some of the masked SBOX circuits were obfuscated in layout using up to 4 levels of standard cells. We use *Degate* to test the resistance to gate reverse engineering. *Degate* is a program for

semi-automatic VLSI reverse engineering of digital logic in chip Degate Community (2022). Figure 48 shows an example of detection in *Degate* for a 4-level obfuscated masked SBOX circuit. We first extract the layout of the obfuscated circuit through diffusion, polysilicon, and first metal with blending and post-processing to mimic micro-photographs. The images are introduced into *Degate* for recognition. The levels of the standard cell library are also integrated into the software with the same blending and post-processing. In figure 48, the pictures show the detected cells in green and the non-detected cells in red. More of the standard cells get successfully detected if the number of detection levels inside *Degate* increases.

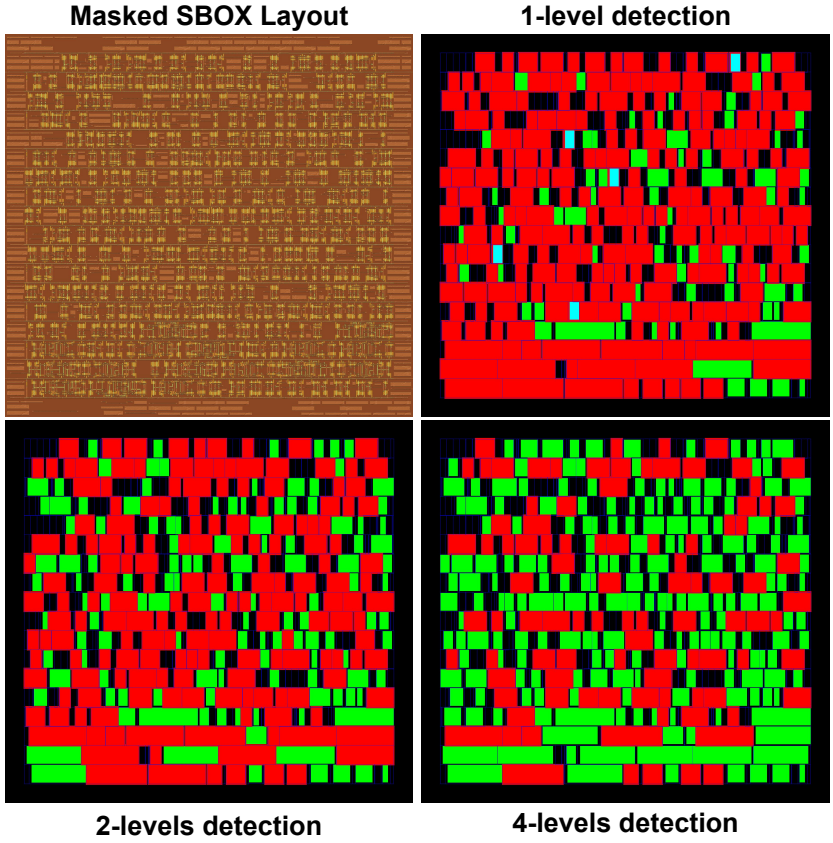

Figure 48. Example of detection results over a 4-level obfuscated masked SBOX implementation using several levels for detection. Green: Detected. Red: Not detected. Cyan: Wrong detected.

In table 10 we present a detailed report for the area of the cells detected by *Degate*. This table shows the results for different obfuscation levels and different detection levels introduced into the tool. For a non-obfuscated circuit, labeled as 1-level obfuscation, the base detection rate for *Degate* is around 62.6% without wrong detections. For 2-level obfuscation circuits, the base standard cell library will be detected around 29.7% with an additional 5.1% of wrong detection. This metric goes further down with a 4-level obfuscated circuit with an 18.7% detection and 1.1% the wrong detection. In this likely scenario, the attacker first tries to do reverse engineering using decapping tools.

Nevertheless, the attacker may eventually be capable of recognizing two out of the four possible levels shown in this scenario for each one of the 18 implemented cells. In this case, with 2-level detection, the 2-level obfuscated circuit returns to a similar detection rate of 72.1% compared to the non-obfuscated version. The 4-level obfuscation increases its rate of identification to 36.0%, but still is not totally reversed. The 4-level obfuscation finally reaches up to 63.8% of the detected cells by acknowledging all 4-levels of standard cells in the library in *Degate*.

Table 10
Detection results over a masked SBOX.

	1 level detect [%]			**2 levels detect [%]**			**4 levels detect [%]**		
SBOX	**D**	**ND**	**W**	**D**	**ND**	**W**	**D**	**ND**	**W**
1 lv	62.6	37.3	0.0	N/A	N/A	N/A	N/A	N/A	N/A
2 lv	29.7	65.1	5.1	72.1	27.4	0.4	N/A	N/A	N/A
4 lv	18.7	80.1	1.1	36.0	63.9	0.0	63.8	36.1	0.0

D = Detected, **ND** = Not Detected, **W** = Wrongly detected

Table 11 presents the implementation results of the SoC in the 0.2μm technology. The

first table presents the digital implementation of the processor, buses, and peripherals. The RISC-V processor occupies most of the area with 24.11%, and the connected SBOX instruction represents 4.57%. The AES core without the DMA controller occupies 19.65%, which is similar to the processor with the SBOX instruction. The DMA controller used for the AES core represents the low area with 5.7%. The combination of peripherals such as GPIO, UART, and SPI ROM represents roughly 16%, while the buses have 5%. The TRNG represents a very low overhead for the area with less than 1%. The debug system occupies 19.81%. Other devices such as JTAG, the reset system, and bus buffers represent 4.18%. In general, the digital part of the SoC is contained in 1.3 mm^2.

The implementation in area of the obfuscated SRAM is similar in size to the digital SoC with 1.28 mm^2. The digital part of the oblivious obfuscator was obfuscated in layout and inserted in the SRAM as part of the row and column decoders. The table 11 presents different obfuscation layouts levels. The overall overhead in area is about 2.7% compared to the RAM, regardless of the levels of layout obfuscation.

Regarding frequency and power, the system consumes about 16 mW of power at 10 MHz in nominal voltage of 1.8 V. The SRAM consumes a similar power with 16.5 mW also at nominal conditions. Power consumption of the SRAM obfuscator is around 1.5 mW, representing around 8% of overhead compared to the RAM, and 4.2% compared to the overall system. The original SRAM presents 51.2 MHz of maximum frequency. If the obfuscator is used, the memory operations need to operate up to 46.7 MHz, which represents 8.7% of overhead in timing.

The final layout of the implemented SoC can be seen in figure 49. This chip contains

Table 11
Implementation results of the SoC in a 0.2µm technology.

Digital SoC	**Area [μm^2]**	**Eq Cells [NAND]**	**Util [%]**
Processor	316261.60	19831	24.11%
SBOX Insn	60009.13	3763	4.57%
AES Core	257735.63	16161	19.65%
Debug	259930.74	16299	19.81%
SPI ROM	91500.06	5737	6.97%
DMA	74984.31	4702	5.72%
GPIO	65884.38	4131	5.02%
UART	54360.35	3409	4.14%
AHB	58488.14	3667	4.46%
APB	10037.67	629	0.77%
TRNG	7897.45	495	0.60%
Other	54841.98	3439	4.18%
Total	1311931.43	82263	100.00%

Obfuscated SRAM	**Area [μm^2]**	**Eq Cells [NAND]**	**Util [%]**
Obs 1	35464.78	2224	2.75%
Obs 2	34929.79	2190	2.71%
Obs 4	35775.82	2243	2.77%
RAM	1289972.77	80886	100.00%

Power & Frequency	**Max Freq [MHz]**	**Power [μW@10MHz]**
System	26.05	16065.6
Obs 4	46.68	1434.5
RAM	51.20	16652.8

the previously mentioned processors and buses, the SRAM with the obfuscator, and some of the SBOX obfuscated layout examples. The 4K SRAM is also presented in transistor view to illustrate the usage of the SRAM obfuscator implemented inside of the SRAM. This SRAM was generated using OpenRAM with unique configurations added to support the 0.2μm technology Guthaus et al. (2016). The obfuscated SBOX implemented in this design is the same as featured in figure 48 but pictured up to the upper-layer metals.

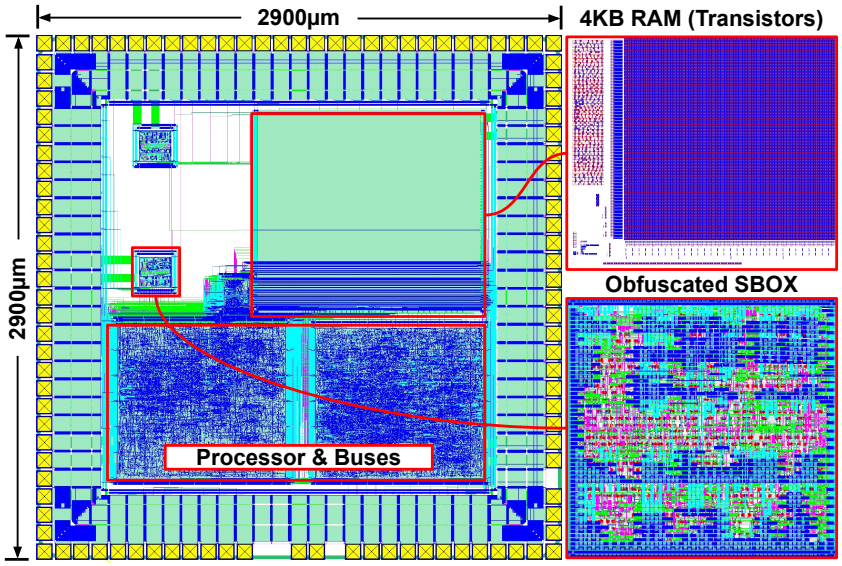

Figure 49. Chip layout capture.

3.5. Chapter Summary

We presented in this chapter a system-on-chip for low-power and low-area applications featuring security peripherals such as a TRNG, AES core, SBOX custom instruction, and obfuscated SRAM. The obfuscation of the SRAM scrambles the address of the transactions coming from an AHB bus. The obfuscation can be enabled by writing a random value from the TRNG in machine mode. Once the processor exits machine mode, the peripheral resets the value in the register used for scrambling. This design and some masked SBOX blocks were obfuscated in

layout. The layout obfuscation uses a series of generated standard cells from the same netlist. These layouts, named levels, are randomly replaced in the synthesis netlist of a digital circuit with low overhead in timing, power, or area. The success rate of detecting 4-level obfuscated circuits is 18.7% only for attackers with only knowledge of the base standard cell library. The SoC is implemented in a 0.2μm technology with 1.3 mm^2 of area and 16 mW of power at 10 MHz. The overhead of in power of the obfuscator represents 2.7% in area and 8% in power.

4. Summary and Conclusions

Several security concerns were analyzed in this book related to system-on-a-chip implementations, and solutions were proposed. The main problems were stated in the introduction where we specifically boarded security problems such as *imaging reversing*, *RAM access*, *code injection*, *firmware extraction*, and *channel capture*. The solution involves several algorithms and methodologies described in chapters 1, 2, and 3, which are addressed as the contributions of this dissertation. This final chapter will summarize all the findings and expose them to the reader to offer a clear perspective of the solutions described.

4.1. Compiled list of contributions

The key contributions of this dissertation are as follows:

- *An standard cell placer* which is optimized for horizontal routing span, and features two different algorithms for placement: Boolean Satisfiability and Graph Reversal.
- *An standard cell generator* by integrating the previous placement algorithms with two routing algorithms: Maze Router by Priorities and Maze Generator with Boolean Satisfiability.
- *An SBOX custom instruction for RISC-V* that is capable to run the AES algorithm for 32-bit simultaneous operation which saves load/save instructions.
- *An AES core for low-energy systems* which exploits the usage of a low-power DMA controller for a RISC-V microcontroller.
- *Olinguito, a chip generator* based on the Chisel Scala library for generation of low-power

systems for security, optimized for analog mixed-signal implementations and padring generation.

- *Functional and formal methodologies to verify processors* based on the Yosys synthesizer for the formal verification, and a general-purpose genetic algorithm (μGP) for the functional verification.

- *An oblivious memory obfuscation circuit* which can be integrated inside of a generated memory using OpenRAM.

- *A layout obfuscation methodology using the standard cell generator* which uses several alternatives of layouts of the same cell for 17 different types of circuits.

These contributions were reviewed by academic peers and their respective publications are listed at the end of this chapter.

4.2. Conclusions

Layout circuit obfuscation is critical to protect design information in security systems. This dissertation has proposed circuit generators that can create security systems and the standard cells that integrate them. The cells can output a range of different layout results that can be used to protect low-overhead circuits. The target security system can handle AES cryptography, whose data in memory is secured through a low-overhead oblivious memory obfuscator. This low-overhead circuit can be obfuscated in layout to prevent circuit extraction using imaging tools and reverse engineering software.

The standard cell generation is fundamental for the layout circuit obfuscation. EDA tools do the integration of these circuits by using standard cells. In chapter 1, we introduced two

standard cell generation procedures with different placement algorithms. The first algorithm is based on boolean satisfiability (SAT) to find the positions of individual P-N transistors in a double-row arrange Duran and Roa (2021b). A series of clauses are introduced into the SAT solver to guarantee transistor abutment, common vertical gates, and unique positioning. This alternative supports optimization for routing by using pseudo-boolean satisfiability (PB-SAT). The algorithm introduces an optimization function that minimizes the horizontal span of the placed transistors and the common vertical gates and diffusions within a cell. The second algorithm presents a graph-reversal with a best-search implementation, explained in section 1.2. The program detects clusters in the netlist, then pairs P-N transistors based on a list of priorities or a smaller aided placement implementation of the PB-SAT algorithm. Finally, the chaining procedure reverses the graph constructed with the P-N transistor pairs with a best-search algorithm. The graph-based alternative shows better routing congestion because of the horizontal net optimizations implemented in the PB-SAT alternative as the aided placement for pairing.

This dissertation proposes a custom instruction SBOX and an AES core for security in low-power designs Duran et al. (2021); Duran and Roa (2021a). The custom instruction SBOX can operate over 32-bit registers in a RISC-V processor. This instruction can be adapted into a lightweight AES software implementation. The energy consumption decreases between 100 and 400 times when using the instruction. Regardless, the quantity of instructions in the software is focused on store/load instructions due to the data fetching for encryption. An AES core is implemented to solve this problem in push-pull and DMA automaton versions. The push-pull version requires the processor to move the data, while the DMA automaton can automatically

fetch and save the data. Using the DMA, the whole system-on-chip can go as low as 9.7 pJ/cyc of energy consumption.

A system for security requires to be reliable. This dissertation proposes Olinguito, a chip generator designed for low-power systems. This generator can output several types of system-on-chips with RISC-V processor, system memory, common peripherals (GPIO, UART, SPI), security peripherals (DMA, AES), and the power management unit (PMU). This management is handled in a separate always-on domain (AON), which contains programmable state machines for sleep and wake-up routines of the other domains. The generator can manage analog instantiations and configurations of signal interactions for the PMU. Additional to this management, Olinguito can also generate padring instantiations which are aware of both the system signals and the chip packaging attached. This work demonstrates the concept of generating low-power SoC by showing two chips in 180nm and 65nm technologies.

For a system to be reliable, the circuits must be verified using different methods. The previous chip generator implements a RISC-V processor, verified using two different methods shown in this dissertation. The first is a functional approach that utilizes genetic algorithms (μGP) to generate tests programs that the processor will execute under verification (PUV). The execution of the instructions is analyzed by a coverage tool, whose metrics are used as the optimization value of the genetic algorithm. By optimizing this way, this functional approach can encounter errors in executing most of the possible covered scenarios for the implemented RISC-V instructions. The second is a formal approach that introduces clauses and assertions to the signals inside the processor. Implementing the RISC-V formal interface evaluates register statuses, instruction decoding, program counter, and memory interactions. The formal approach

encounters errors according to the formal deck, which contains constraints and assertions of each of the RISC-V instructions and some additional keep-alive tests.

Low-overhead memory security needs layout obfuscation. A low-overhead oblivious memory obfuscator is presented. This circuit scrambles the address of the system inside of an SRAM module to obfuscate the contents. An oblivious register and SBOX units are included inside the peripheral to perform a 1-to-1 address translation as a demonstration circuit. This secure memory can protect critical information such as the key and algorithm critical execution of cryptography such as AES. The user-mode program can request execution of such algorithm in machine mode, and then a random value is fetched and written into the write-only oblivious register. When the execution ends, or the processor exits machine mode, the oblivious register is reset, leaving the memory contents obfuscated in position.

Low-overhead security circuits can be easily identifiable in layout because of the size. The circuit behavior can be extracted and further reversed using imaging tools and the standard cell library knowledge. This dissertation presents a layout obfuscation technique using the standard cell generators previously discussed. The generators can output several solutions of the placement by modifying the algorithms. The layout of the standard cells changes drastically if the solution is different for the same netlist. Each of the layouts is named a level. The digital circuit can be scrambled from the synthesis phase of the EDA implementation by replacing the cells with several levels of the same cell. The circuit is then obfuscated in the layout by performing place and route with minimal overhead for timing, power, and area. First iterations of reverse engineering using imaging tools show under 20% of success of identification for the cells implemented in the low-overhead security circuits.

4.3. Future work

Although the placement algorithms were presented in this dissertation, the routing algorithms are not adequately explored. In section 1.1, the PB-SAT algorithm optimizes according to the horizontal routing span, but the vertical routing and subsequent vertical congestion are not taken into the formulation. Detail placements in the vertical axis are also not formulated in this dissertation. A similar localized version of the optimizer can be applied to aid the routing algorithms further.

The power management unit inside the always-on domain can compromise the system's security. Because the design relies on external analog interfaces, the events handled by the PMU can be induced by an attacker. These events interrupt the execution of the protected algorithm. Some external resources can be leaked into non-secured memory or processor registers if timed correctly. An interaction between the low-energy peripherals and the cryptography peripherals is necessary to solve this issue.

The RISC-V processor is verified using two different procedures. Further exploration of these approaches can be applied to AES and arithmetic operations. Currently, the verification over operations only includes a list of test vectors. The functional genetic algorithm case can create random test vectors but will not cover the whole range of possibilities in a single run. In the same way, the formal verification of operations is not possible. The creation of assumptions and assertions for an operation comes down to the same tests vectors. According to the current calculation round, there is a possibility to specify the behavior of certain aspects of AES.

Contribution List

4.4. Conference papers

1. **C. Duran**, H. Gomez and E. Roa, "AES Sbox Acceleration Schemes for Low-Cost SoCs," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, pp. 1-5, doi: https://doi.org/10.1109/ISCAS51556.2021.9401539.

2. **C. Duran** and E. Roa, "Routing-Aware Standard Cell Placement Algorithm Applying Boolean Satisfiability," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, pp. 1-5, doi: https://doi.org/10.1109/ISCAS51556.2021.9401098.

3. **C. Duran** et al., "An Energy-Efficient RISC-V RV32IMAC Microcontroller for Periodical-Driven Sensing Applications," 2020 IEEE Custom Integrated Circuits Conference (CICC), 2020, pp. 1-4, doi: https://doi.org/10.1109/CICC48029.2020.9075877.

4. **C. Duran**, H. Morales, C. Rojas, A. Ruospo, E. Sanchez and E. Roa, "Simulation and Formal: The Best of Both Domains for Instruction Set Verification of RISC-V Based Processors," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1-4, doi: https://doi.org/10.1109/ISCAS45731.2020.9180589.

5. H. Gomez, **C. Duran** and E. Roa, "Standard cell camouflage method to counter silicon reverse engineering," 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1-4, doi: https://doi.org/10.1109/ICCE.2018.8326300.

4.5. Journal Papers

1. **C. Duran** and E. Roa, "A 10pJ/bit 256b AES-SoC Exploiting Memory Access Acceleration," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 3, pp. 1612-1616, March 2022, doi: https://doi.org/10.1109/TCSII.2021.3126984.

2. H. Gomez, **C. Duran** and E. Roa, "Defeating Silicon Reverse Engineering Using a Layout-Level Standard Cell Camouflage," in IEEE Transactions on Consumer Electronics, vol. 65, no. 1, pp. 109-118, Feb. 2019, doi: https://doi.org/10.1109/TCE.2018.2890616.

4.6. Papers in Submission/Revision Process

1. **C. Duran**, C-K. Pham and E. Roa, "Exploiting Graph-based Standard Cell Placement Algorithm by Combining Boolean Satisfiability", to submit in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

2. **C. Duran**, C-K. Pham and E. Roa, "Layout Obfuscation for Protecting Low-Overhead Oblivious Memory Security in Microcontrollers" to submit in IEEE Transactions on Circuits and Systems II: Express Briefs.

4.7. Other Publications

1. H. Morales, **C. Duran** and E. Roa, "A Low-Area Direct Memory Access Controller Architecture for a RISC-V Based Low-Power Microcontroller," 2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS), 2019, pp. 97-100, doi: https://doi.org/10.1109/LASCAS.2019.8667579.

Bibliographic References

Awad, A., Wang, Y., Shands, D., and Solihin, Y. (2017). ObfusMem: A low-overhead access obfuscation for trusted memories. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 107–119.

Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., and Asanović, K. (2012). Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1216–1225, New York, NY, USA. ACM.

Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., and Asanović, K. (June 2012). Chisel: Constructing Hardware in a Scala Embedded Language. In *Design Automation Conf. (DAC)*, pages 1212–1221.

Banerjee, U. and Chandrakasan, A. P. (2021). A Low-Power Elliptic Curve Pairing Crypto-Processor for Secure Embedded Blockchain and Functional Encryption. In *2021 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–2.

Banerjee, U., Das, S., and Chandrakasan, A. P. (2020). Accelerating post-quantum cryptography using an energy-efficient tls crypto-processor. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Banerjee, U., Juvekar, C., Wright, A., Arvind, and Chandrakasan, A. P. (2018). An Energy-Efficient Reconfigurable DTLS Cryptographic Engine for End-to-End Security in IoT Applications. In *2018 IEEE Int. Solid-State Circuits Conf. - (ISSCC)*, pages 42–44.

Bar-Yehuda, R., Feldman, J. A., Pinter, R. Y., and Wimer, S. (1989). Depth-first-search and dynamic programming algorithms for efficient CMOS cell generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):737–743.

Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., and Rijmen, V. (2015). Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200.

Bovet, D. and Cesati, M. (2005). *Understanding The Linux Kernel*. Oreilly & Associates Inc.

Branco, R. and Gueron, S. (2016). Blinded random corruption attacks. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 85–90.

Brist, G. and Park, J. (2015). A Novel Approach to IC, Package and Board Co-Optimization. In *Sixteenth International Symposium on Quality Electronic Design*, pages 512–518.

Bykov, S., Ryzhenko, N., and Sorokin, A. (2016). Automated solution for preventing design rules violations at abutment stage for standard cells synthesis flow. In *2016 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–4.

Canright, D. (2005). A Very Compact S-Box for AES. In Rao, J. R. and Sunar, B., editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg. Springer Berlin Heidelberg.

Chang, E., Han, J., Bae, W., Wang, Z., Narevsky, N., NikoliC, B., and Alon, E. (2018). BAG2: A process-portable framework for generator-based AMS circuit design. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8.

Chi Yi Hwang, Yung Ching Hsieh, Youn-Long Lin, and Yu-Chin Hsu (1993). An efficient layout style for two-metal CMOS leaf cells and its automatic synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(3):410–424.

Choi, Y., Sim, J., and Kim, L. (2020). CREMON: Cryptography Embedded on the Convolutional Neural Network Accelerator. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 1–1.

Corno, F., Sanchez, E., and Squillero, G. (2005). Evolving Assembly Programs: How Games Help Microprocessor Validation. *IEEE Transactions on Evolutionary Computation*, 9(6):695–706.

Cortadella, J. (2013). Area-Optimal Transistor Folding for 1-D Gridded Cell Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(11):1708–1721.

de Dood, P., Frederick, M. W., Wang, J. C., CLINE, B. T., Xu, X., CHEN, A. W., Chong, Y. K., Shore, T., Thyagarajan, S., Yeung, G., Albers, D. J., and Granda, D. W. (2019). Computer Implemented System and Method for Generating a Layout of a Cell Defining a Circuit Component.

Degate Community (2022). Degate. `https://github.com/DegateCommunity/Degate`.

Deng, C., Wang, B., Liu, L., Zhu, M., Wu, Y., Li, H., Yin, S., and Wei, S. (2020). A 60 Gb/s-Level Coarse-Grained Reconfigurable Cryptographic Processor With Less Than 1-W Power. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(2):375–379.

Duran, C., Gomez, H., and Roa, E. (2021). AES Sbox Acceleration Schemes for Low-Cost SoCs. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Duran, C. and Roa, E. (2021a). A 10pJ/bit 256b AES-SoC Exploiting Memory Access Acceleration. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 1–1.

Duran, C. and Roa, E. (2021b). Routing-Aware Standard Cell Placement Algorithm Applying Boolean Satisfiability. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Duran, C., Wachs, M., Rueda G., L. E., Huntington, A., Ardila, J., Kang, J., Amaya, A., Gomez, H., Romero, J., Fernandez, L., Flechas, F., Torres, R., Moya, J., Ramirez, W., Arenas, J., Gomez, J., Morales, H., Rojas, C., Mantilla, A., Roa, E., and Asanovic, K. (2020). An Energy-Efficient RISC-V RV32IMAC Microcontroller for Periodical-Driven Sensing Applications. In *2020 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4.

Gomez, H., Duran, C., and Roa, E. (2019). Defeating Silicon Reverse Engineering Using a Layout-Level Standard Cell Camouflage. *IEEE Transactions on Consumer Electronics*, 65(1):109–118.

Gupta, A. and Hayes, J. P. (1996). Width Minimization of Two-Dimensional CMOS Cells Using Integer Programming. In *Proceedings of International Conference on Computer Aided Design*, pages 660–667.

Gupta, A. and Hayes, J. P. (1998). Optimal 2-D cell Layout with Integrated Transistor Fold-

ing. In *1998 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (IEEE Cat. No.98CB36287)*, pages 128–135.

Gupta, A., The, S.-C., and Hayes, J. P. (1996). XPRESS: a Cell Layout Generator with Integrated Transistor Folding. In *Proceedings ED TC European Design and Test Conference*, pages 393–400.

Guruswamy, M., Maziasz, R. L., Dulitz, D., Raman, S., Chiluvuri, V., Fernandez, A., and Jones, L. G. (1997). CELLERITY: A Fully Automatic Layout Synthesis System For Standard Cell Libraries. In *Proceedings of the 34th Design Automation Conference*, pages 327–332.

Guthaus, M. R., Stine, J. E., Ataei, S., Chen, B., Wu, B., and Sarwar, M. (2016). OpenRAM: An Open-Source Memory Compiler. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, New York, NY, USA. Association for Computing Machinery.

Hankerson, D., Menezes, A., and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer-Verlag, New York.

Herdt, V., Große, D., Le, H. M., and Drechsler, R. (2019). Verifying Instruction Set Simulators using Coverage-guided Fuzzing*. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 360–365.

Hoang, T. T., Duran, C., Tsukamoto, A., Suzaki, K., and Pham, C. K. (2020). Cryptographic Accelerators for Trusted Execution Environment in RISC-V Processors. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4.

Horowitz, M. (2014). 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14.

Hsich, Y.-C., Hwang, C.-Y., Lin, Y.-L., and Hsu, Y.-C. (1991). LiB: a CMOS Cell Compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(8):994–1005.

Hutter, M., Feldhofer, M., and Wolkerstorfer, J. (2011). A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES Like Sardines. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Comm.*, pages 144–159. Springer.

Hwang, C. Y., Hsieh, Y. C., Lin, Y. L., and Hsu, Y. C. (1990). A Fast Transistor-Chaining Algorithm for CMOS Cell Layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(7):781–786.

Iizuka, T., Ikeda, M., and Asada, K. (2006). Exact Minimum-Width Multi-Row Transistor Placement for Dual and Non-dual CMOS Cells. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4 pp.–.

Jo, K., Ahn, S., Do, J., Song, T., Kim, T., and Choi, K. (2019). Design Rule Evaluation Framework Using Automatic Cell Layout Generator for Design Technology Co-Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1933–1946.

Jo, K., Ahn, S., Kim, T., and Choi, K. (2018). Cohesive Techniques for Cell Layout Opti-

mization Supporting 2D Metal-1 Routing Completion. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 500–506.

Krishnakumar, G., SLPSK, P., Vairam, P. K., Rebeiro, C., and Veezhinathan, K. (2018). GANDALF: A Fine-Grained Hardware–Software Co-Design for Preventing Memory Attacks. *IEEE Embedded Systems Letters*, 10(3):83–86.

Kumar, R., Kohler, E., and Srivastava, M. (2007a). Harbor: Software-based Memory Protection For Sensor Nodes. In *2007 6th International Symposium on Information Processing in Sensor Networks*, pages 340–349.

Kumar, R., Singhania, A., Castner, A., Kohler, E., and Srivastava, M. (2007b). A System For Coarse Grained Memory Protection In Tiny Embedded Processors. In *2007 44th ACM/IEEE Design Automation Conference*, pages 218–223.

Kundi, D., Khalid, A., Aziz, A., Wang, C., O'Neill, M., and Liu, W. (2020). Resource-Shared Crypto-Coprocessor of AES Enc/Dec With SHA-3. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–14.

Lazzari, C., Anghel, L., and Reis, R. (2007). *"A Transistor Placement Technique Using Genetic Algorithm and Analytical Programming"*, pages 331–344. Springer US, Boston, MA.

Lee, D., Kohlbrenner, D., Shinde, S., Asanovic, K., and Song, D. (2020). Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20.

Lee, H. and Chang, Y. (2012). A Chip-Package-Board Co-Design Methodology. In *DAC Design Automation Conference 2012*, pages 1082–1087.

Liang, K., Feng, Y., Wei, J., and Guo, W. (2016). SecPage - A Lightweight Memory Protection Architecture. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1917–1922.

Lu, A., Lu, H. J., Jang, E. J., Lin, Y. P., Hung, C. H., Chuang, C. C., and Lin, R. B. (2015a). Simultaneous Transistor Pairing and Placement for CMOS Standard Cells. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1647–1652.

Lu, Z., Xing, X., Tong, Q., and Liu, Z. (2015b). Efficient Off-Chip Memory Protection Mechanism for Embedded Computing Systems Using AES-GCM. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pages 236–237.

Marshall, B., Newell, G. R., Page, D., Saarinen, M.-J. O., and Wolf, C. (2020). The design of scalar AES instruction set extensions for RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):109–136. Artifact available at `https://artifacts.iacr.org/tches/2021/a3`.

Mathew, S., Satpathy, S., Suresh, V., Kaul, H., Anders, M., Chen, G., Agarwal, A., Hsu, S., and Krishnamurthy, R. (2014). 340mV–1.1V, 289Gbps/W, 2090-gate NanoAES Hardware Accelerator with Area-Optimized Encrypt/Decrypt GF(24)2 Polynomials in 22nm Tri-Gate CMOS. In *2014 Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2.

Maziasz, R. L. and Hayes, J. P. (1991). Exact width and height minimization of CMOS cells. In *28th ACM/IEEE Design Automation Conference*, pages 487–493.

Morales, H., Duran, C., and Roa, E. (2019). A Low-Area Direct Memory Access Controller Architecture for a RISC-V Based Low-Power Microcontroller. In *2019 IEEE 10th Latin American Symposium on Circuits Systems (LASCAS)*, pages 97–100.

Nikolic, B., Alon, E., and Asanovic, K. (2018). Generating the Next Wave of Custom Silicon. In *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, pages 6–11.

NIST (2001). Advanced Encryption Standard. *NIST FIPS PUB 197*.

Park, J. F. (2010). Board Driven I/O Planning and Optimization. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 395–397.

Philipp, T. and Steinke, P. (2015). PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF. In Heule, M. and Weaver, S., editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, volume 9340 of *Lecture Notes in Computer Science*, pages 9–16. Springer International Publishing.

Rajendran, J., Sinanoglu, O., and Karri, R. (2013). VLSI Testing Based Security Metric for IC Camouflaging. In *2013 IEEE International Test Conference (ITC)*, pages 1–4.

Real Time Statistics Project (2022). Internet Live Stats. `https://www.internetlivestats.com/`.

Reid, A., Chen, R., Deligiannis, A., Gilday, D., Hoyes, D., Keen, W., Pathirane, A., Shepherd, E., Vrabel, P., and Zaidi, A. (2016). End-to-End Verification of Arm Processors with Isa-Formal. In Chaudhuri, S. and Farzan, A., editors, *Proceedings of the 2016 International Conference on Computer Aided Verification (CAV'16)*, volume 9780 of *Lecture Notes in Computer Science*, pages 42–58. Springer Verlag.

RISC-V Foundation (2019a). Rocket Chip Generator. `https://github.com/freechipsproject/rocket-chip`.

RISC-V Foundation (2019b). Spike RISC-V ISA Simulator. `https://github.com/riscv/riscv-isa-sim`.

Ryzhenko, N. and Burns, S. (2012). Standard Cell Routing Via Boolean Satisfiability. In *DAC Design Automation Conference 2012*, pages 603–612.

Sadakane, T., Nakao, H., and Terai, M. (1995). A New Hierarchical Algorithm for Transistor Placement in CMOS Macro Cell Design. In *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*, pages 461–464.

Sanchez, E., Schillaci, M., and Squillero, G. (2011). *Evolutionary Optimization: the μGP toolkit*. Springer Science & Business Media.

Sayilar, G. and Chiou, D. (2014). Cryptoraptor: High Throughput Reconfigurable Cryptographic Processor. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 155–161.

Schiavone, P. D., Sanchez, E., Ruospo, A., Minervini, F., Zaruba, F., Haugou, G., and Benini,

L. (2018). An Open-Source Verification Framework for Open-Source Cores: A RISC-V Case Study. In *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 43–48.

Serrano, R., Duran, C., Hoang, T.-T., Sarmiento, M., Nguyen, K.-D., Tsukamoto, A., Suzaki, K., and Pham, C.-K. (2021). A fully digital true random number generator with entropy source based in frequency collapse. *IEEE Access*, 9:105748–105755.

Sifive, Inc. (2021). *Sifive FE310-G000 Preliminary Datasheet*. Sifive, Inc. Ver. 1.0.6.

Stillmaker, A. and Baas, B. (2017). Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm. *Integration*, 58:74–81.

Sung Mo Kang (1987). Metal–Metal Matrix (M /sup 3/) for High-Speed MOS VLSI Layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):886–891.

Symbiotic EDA (2019). RISC-V Formal Verification Framework. `https://github.com/SymbioticEDA/riscv-formal`.

Uehara and Vancleemput (1981). Optimal Layout of CMOS Functional Arrays. *IEEE Transactions on Computers*, C-30(5):305–312.

Wang, L.-T., Chang, Y.-W., and Cheng, K.-T. T., editors (2009). *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Wang, Y. and Ha, Y. (2013). FPGA-Based 40.9-Gbits/s Masked AES With Area Optimization for Storage Area Network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(1):36–40.

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2014). The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley.

Wolf, C. (2022). Yosys open synthesis suite. `http://www.clifford.at/yosys/`.

Yang, J., Gao, L., and Zhang, Y. (2005). Improving memory encryption performance in secure processors. *IEEE Transactions on Computers*, 54(5):630–640.

Yu, W. and Köse, S. (2016). A Voltage Regulator-Assisted Lightweight AES Implementation Against DPA Attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(8):1152–1163.

Zhang, Y., Xu, L., Dong, Q., Wang, J., Blaauw, D., and Sylvester, D. (2018). Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security. *IEEE Journal of Solid-State Circuits*, 53(4):995–1005.

Zhao, W., Ha, Y., and Alioto, M. (2015). Novel Self-Body-Biasing and Statistical Design for Near-Threshold Circuits With Ultra Energy-Efficient AES as Case Study. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(8):1390–1401.

Zheng, R. and Huang, M. C. (2017). Redundant memory array architecture for efficient se-

lective protection. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 214–227.