Low Overhead Logic Locking for System-Level Security: A Design Space Modeling Approach

Long Lam
Rochester Institute of Technology
ll5530@rit.edu

Maksym Melnyk
Rochester Institute of Technology
mm6878@rit.edu

Michael Zuzak
Rochester Institute of Technology
mjzeec@rit.edu

Abstract

Integrated circuits are often fabricated in untrusted facilities, making intellectual property privacy a concern. This prompted the development of logic locking, a security technique that corrupts the functionality of a design without a correct secret key. Prior work has shown that system-level phenomena can degrade the security of locking, highlighting the importance of configuring locking in a system. In this work, we propose a design space modeling framework to generate system-level models of the logic locking design space in arbitrary ICs by simulating a small, carefully-selected portion of the design space. These models are used to automatically identify near-optimal locking configurations in a system that achieve security goals with minimal power/area overhead. We evaluate our framework with two experiments. 1) We evaluate the quality of modeling-produced solutions by exhaustively simulating locking in a RISC-V ALU. The models produced by our algorithm had an average $R^2 > 0.99$ for all design objectives and identified a locking configuration within 96% of the globally optimal solution after simulating < 3.6% of the design space. 2) We compare our model-based locking to conventional module-level locking in a RISC-V processor. The locking configuration from our model-based approach required 29.5% less power on average than conventional approaches and was the only method to identify a solution meeting all design objectives.

CCS Concepts

• **Security and privacy → Hardware security implementation**; *Hardware attacks and countermeasures.*

Keywords

Design Space Modeling, Untrusted Foundry Problem, Logic Locking

ACM Reference Format:

Long Lam, Maksym Melnyk, and Michael Zuzak. 2024. Low Overhead Logic Locking for System-Level Security: A Design Space Modeling Approach. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24), August 5–7, 2024, Newport Beach, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665314.3670833

1 Introduction

The high cost and complexity of integrated circuit (IC) fabrication has driven the widespread use of untrusted facilities to manufacture and test ICs. The GDSII files provided for fabrication can be reverse-engineered into complete design netlists, leaking high-value intellectual property (IP) and enabling piracy, counterfeiting, and malicious modification [9]. This set of supply-chain security challenges is known as the *untrusted foundry problem.*

Logic locking was developed to mitigate the security concerns from untrusted fabrication by locking the function of specific modules in an IC behind a secret key [2]. Without the key, error is injected in the locked module to cause failures and derail unauthorized use. A large body of work on locking explores these schemes through a module-level lens, relying on the assumption that any errant functionality in a module is sufficient to protect an IC from unauthorized use [16]. However, recent research has shown that system-level phenomena (e.g. error resilience, input space utilization, etc.) degrade or even eliminate the security of logic locking when an IC is viewed as a whole [13, 20]. This prompted a shift towards logic locking for high-level security (i.e., architecture or system), including gate-level constructions with architectural evaluations [18, 22], and HLS strategies [18, 21]. Despite varied high-level locking approaches, most research relies on the assumption that a designer has pre-configured locking in a design and must only consider optimizing security and attack resilience. This is a non-trivial assumption that substantially impacts design goals (e.g., power).

In this work, we aim to address this problem, exploring how to best configure logic locking in an IC to meet system objectives. Specifically, we address the following design problem. *Given a list of modules containing critical IP that must be protected and an arbitrary set of design objectives (e.g., power budget, attack time, etc.), identify a set of locking techniques, their size, and the modules they are implemented in to optimize system design goals.* To achieve this, a strong understanding of the system design space of logic locking is necessary. Unfortunately, this design space is not intuitive due to the many variables involved and the complex interaction between these variables. Moreover, exhaustive simulation or ad-hoc approaches to explore this design space are infeasible. To overcome this, we propose design space modeling (DSM) to model the locking design space. DSM has been successfully applied to varied design space exploration (DSE) problems with complex, multi-variate design spaces [5, 11]. This leads to our primary goal: to develop a DSM framework to model the system-level design space for logic locking and employ these models to identify optimal logic locking configurations in arbitrary ICs sufficient for system-wide security with minimal power and area overhead.

1.1 Contributions

We propose DSM to configure logic locking to achieve designer-specified security requirements with minimal corresponding power and area overhead in arbitrary ICs. To do so, small portions of the design space are iteratively identified and simulated to produce

ISLPED '24, August 5–7, 2024, Newport Beach, CA, USA

ACM ISBN 979-8-4007-0688-2/24/08.
https://doi.org/10.1145/3665314.3670833

mathematical models for design goals (i.e., power overhead, security, etc.). These models are then used to identify near-optimal configurations of logic locking in arbitrary ICs based on designer-specified goals. We summarize our contributions as follows:

- A quantifiable, system-level security metric for logic locking that quantifies both the frequency and severity of errors for an unauthorized user of a logic-locked system.
- An algorithm for guided DSM to automatically configure logic locking in an IC to meet arbitrary, designer-specified goals with a minimal power and area overhead.
- An open-source DSM framework that applies our algorithms to configure logic locking in arbitrary ICs. This can be found at https://github.com/mzuzak/DSM-for-Logic-Locking.

To evaluate the proposed DSM framework, we applied it to lock a RISC-V processor [15]. The design space models produced by the framework were highly predictive, modeling the power, area, security, and SAT attack runtime of arbitrary locking solutions with an average $R^2 > 0.99$. Moreover, our DSM framework identified system-level locking solutions within 96% of the globally optimal solution after simulating only 3.6% of the design space. Compared to conventional methods that configure logic locking at the module level, the locking solution identified by our model-based approach required 29.5% less power on average and was the only method to identify a solution meeting all design objectives.

2 Preliminaries

2.1 Logic Locking

Logic locking addresses the untrusted foundry problem by rendering IC function dependent on a locking key. This is done through adding extra primary inputs, known as key inputs, to drive combinational logic in a target module in a design. The security of locking is often evaluated by two goals: 1) error severity; and 2) attack resilience [21]. Error severity is the ability of logic locking to derail device function, often quantified by the percent of a locked module's input space producing corrupt output for wrong keys. Attack resilience is the ability of locking to resist attack. The SAT attack is commonly used to assess attack resilience [10, 12, 16, 20]. State-of-the-art locking includes Stripped Functionality Logic Locking (SFLL) [10], CASLock [12], and others [2].

2.2 Design Space Exploration and Modeling

Design space exploration (DSE) is used to identify favorable system configurations from a vast and complex design space. It is common in the area of processor [8] and 3D IC [11] design. Due to continued increases in the number of design space variables and the complexity of evaluating design decisions, DSM was developed to perform DSE by generating mathematical models for the design space that are used to predict favorable design configurations [5, 11].

2.3 Related Work

Prior work has adopted a high-level (i.e., system or architecture) approach to the untrusted foundry problem, including secure scan-chains [1] and behavioral locking during high-level synthesis or at the RT-Level [3, 6, 7]. While these approaches consider high-level security, they do not use logic locking. This overlooks the body of work developing provably secure and low-overhead logic locking. For example, the behavioral locking used by TAO [7] can be unlocked by an SAT-based attacker that is provably resisted by most recent logic locking [10, 12, 22]. Several works have explored locking for system security [18, 21]. However, these works assume that a designer has already selected locking techniques and implemented them in a design, aiming to tune locking in a module to optimize security goals. These approaches differ from this work where we assume the designer has only specified the modules with critical IP to be locked and provided a cost function.

Table 1: Example system-level design space for logic locking.

Variable:	Possible Variable Configurations
Lockable Module:	{ALU, Decoder, Branch Predictor}
Locking Techniques:	{SFLL [10], CASLock [12], SLL [17]}
Key Length (Bits):	{0,16,32,48,64}

2.4 Threat Model

We consider an untrusted foundry adversary who aims to use the IC in an unauthorized manner. They can take any strategy using:

(1) A locked IC netlist from GDSII reverse-engineering [9].
(2) An activated, black-box oracle IC from testing facilities or the open market that can be queried with arbitrary inputs.

A defense must 1) resist attacks against locking (e.g., SAT attack [14]); and 2) inject enough error to stop unauthorized IP use for a wrong key. This model is consistent with prior work [10, 12, 21, 22].

3 Motivation and Problem Formulation

System-level locking configuration has sizable design implications [20]. Therefore, a method to understand the system design space of locking is crucial to produce effective locked ICs. This work proposes a method to do so. We begin by formalizing the system-level logic locking DSE problem addressed in this work.

DSE poses the problem: given a set of decision variables (degrees-of-freedom), m, identify a variable configuration that optimizes a set of objective values, n [8]. DSM solves the DSE problem by developing a fitness function, f(m), that translates a point in the design space (m variable configuration) to a point in the solution space (n objective values). This fitness function is used to identify configurations in the design space as candidate solutions.

For the system-level logic locking configuration problem, the m-variable design space contains a set of variables for each module with critical IP that must be locked. These variables correspond to each candidate locking technique that could be used to lock the module and the length of the key for the locking technique in each module. Tbl. 1 contains an example of the design space for a processor IC with 3 modules that must be locked and 3 possible locking schemes considered that use $|k| = \{0, 16, 32, 48, 64\}$ key bits. Even this small sample design space includes 125^3 possible design configurations (i.e., 125 locking combinations per module with 3 independently-locked modules). Hence, any attempt to simulate more than a tiny fraction of the $> 10^6$ locking solutions is infeasible.

There are 4 design objectives[1]: 1) system security (see Sec. 4); 2) SAT attack runtime; 3) power overhead; and 4) area overhead. An arbitrary cost function will mix these objective values into a quantifiable goal (e.g., minimize power subject to a security constraint).

[1]We assume the designer is unwilling to degrade clock frequency for locking. Hence, all designs are subject to a fixed clock constraint and timing is excluded as an objective.

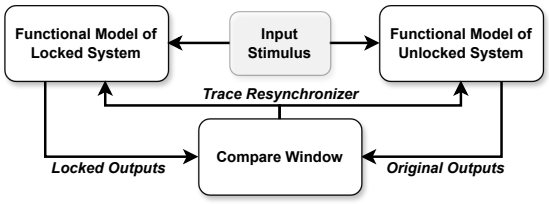

Figure 1: Method to quantify proposed security metric (S).

4 System-Level Security Metric (S)

Principled system design requires quantifiable design objectives. In the case of logic locking, prior work relied on metrics expressed qualitatively [18, 21] (e.g., "restrict unauthorized use") or at the module level [12]. To enable system-level design, we propose a security-metric, S, equal to the ratio of *inputs per observed error* and *inputs until recovery*. Inputs per observed error measures the average number of characteristic inputs that must be applied to the IC until an error is observed on output pins. This quantifies the rate that locking injects errors, taking into account system-level phenomena that mask error, such as error resilience. Inputs until recovery measures the average number of errant outputs until the IC returns to correct output. This measures the severity of locking-induced error, taking into account system-level phenomena that limit error impact, including module importance and redundancy. **Smaller S values indicate higher security.**

4.1 Measuring the System Security Metric (S)

The proposed S metric is dependent on the specific inputs applied to the IC, the locking configuration, and the characteristics of the locked system. Thus, we must quantify it per design. To do so, we propose the following measurement framework. Two functional simulations for an IC are run simultaneously: one with a random incorrect key and one with the correct key. At each time, the simulations (i.e., locked and unlocked) are compared. We depict this measurement setup in Fig. 1. Note that this approach is similar to the system-level security metrics proposed in [20], however, in this work we produce a single, unified security metric for locking.

Both functional simulations (i.e., locked and unlocked) are run on a common input trace. As the simulation runs, locking-induced errors are identified by comparing the output values produced by both functional simulations over a large output window, called the compare window. Any divergence of the locked IC output is classified as a critical error and used to calculate inputs per observed error. The duration of this error is measured by tracking how long it takes for outputs from the locked simulation instance to once again match the outputs from the other simulation instance. The number of cycles until this occurs is used to measure inputs until recovery. When this recovery occurs, the two output traces are re-synchronized (to reset the compare window) and functional simulation continues. This allows system-wide security (S) to be measured for arbitrary locking configurations in arbitrary ICs.

5 Design Space Modeling Algorithm

In this section, we formalize our DSM algorithm to produce the models used to identify system-level locking configurations that optimize designer cost functions in arbitrary ICs. A block diagram for the DSM algorithm is in Fig. 2. The details of each component of the DSM algorithm are in the sections below. The complete code can be found at https://github.com/mzuzak/DSM-for-Logic-Locking.

5.1 Overview of DSM Algorithm

As input, the m-variable system-level locking design space and a user design objective (i.e., cost function) is provided. The locking design space consists of: 1) the modules with critical IP to be locked; 2) the locking schemes that can be used in each locked module; and 3) the possible size of each locking scheme in key bits. See Tbl. 1 for a small example design space. The design objective formalizes the system-level designer goals. For example, a design objective could be to identify a locking solution with minimum power overhead subject to an area, SAT runtime, and S-metric constraint. Upon termination, the framework returns: 1) a design space model, namely a fitness function, f(m), that estimates the four objective values (S-metric, SAT runtime, power, and area) for any m-variable locking solution in the target IC; and 2) a predicted optimal locking solution generated from these models based on the design objective.

To generate these outputs, the proposed DSM framework first performs a sampling of η uniformly distributed design space points (see Sec. 5.2). Regression functions are then used to generate a model from this data. We outline the specific regression functions and how they were selected in Sec. 5.3. The regression model will be used to predict objective values for locking solutions, allowing simulation results to be compared to model estimates to assess model accuracy. The stopping criteria are then evaluated to terminate modeling if model accuracy is above a threshold or if the design space becomes oversampled, risking overfitting (see Sec. 5.5). If termination does not occur, a region of interest (ROI) is defined based on the model, highlighting regions where optimal design configurations are likely to occur (see Sec. 5.4). The proposed DSM framework will select the next simulation from the ROI. This configuration will then be simulated to quantify each design objective (see Sec. 6.1 for the simulation framework). The model is then re-generated with these added data points and the process continues until the stopping criteria are met. Upon termination, the design space model, f(m), and the expected optimal configuration is returned. Key components of the DSM algorithm are green-shaded in Fig. 2 and described below.

5.2 Initial Sampling of Design Space

Creating the initial model requires balance between over-sampling, which risks long runtimes and overfitting, and under-sampling, which may yield a poor model to guide simulation. To generate the initial model, a set of points for simulation are selected uniformly (i.e., equally spaced) in the design space. This ensures that the initial model will not be biased towards a specific region of the design

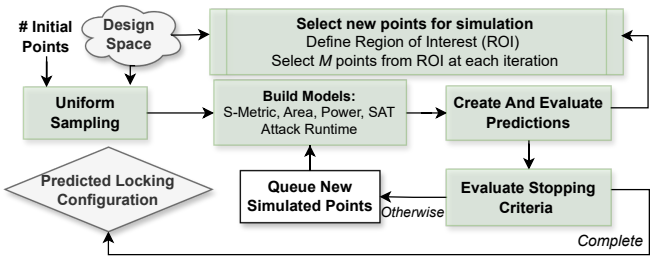

Figure 2: Overview of proposed DSM algorithm.

space. This helps minimize the risk of the model producing an artificial maxima in a large, unsampled portion of the design space that could potentially arise from non-uniform initial sampling approaches. Such an artificial maxima would discourage any further simulation in that region of the design space during the directed simulation phase described in the following section, failing to identify any high-quality solutions that may exist in this region. The number of simulated points used to generate initial models, defined as η, is left to the user as design space complexity may vary widely by design, locking scheme, or other application-specific context.

5.3 Building Models and Regression Functions

A suitable regression function is necessary to accurately model the design space. To select the regression function for each design objective, we evaluated several candidates by building models on a small subset of design points and assessing their efficacy using an exhaustive characterization of the locking design space of an ALU in a RISC-V processor (see Sec. 6.2). Evaluated regression functions included SS-ANOVA [4], linear, logistic, polynomial, and exponential. We omit detailed results for brevity, however, we discuss the selected regression function for each design objective below.

5.3.1 Power, Area, and S-Metric For power, area, and S-metric objectives, SS-ANOVA outperformed all other evaluated regression functions in the RISC-V ALU, achieving an average $R^2 > 0.99$ (see Sec. 6.2). Thus, we have adopted SS-ANOVA [4] to produce models for these 3 design objectives. SS-ANOVA fits smoothing splines (piece-wise cubic functions) to a dataset using the analysis of variance (ANOVA) method [4]. ANOVA is a statistical technique that aims to analyze the underlying source of variance in a population and generate a model as a function of descriptive properties. An SS-ANOVA model, f, is represented as a function of independent variables $\mathbf{m} = \{m_1, m_2, ..., m_n\}$ as depicted in Eqn. 1 [4]. Each unique subset of independent variables is defined as a term. The order of the term is defined as the number of independent variables in the subset. SS-ANOVA models each term as a smoothing spline, $\{f_1, ..., f_{1,2,...,n}\}$. The final model, f, is the sum of all smoothing splines.

$$f(\mathbf{m}) = c + \sum_{i=1}^{n} f_i(m_i) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} f_{i,j}(m_i, m_j) + ...+ f_{1,2,...,n}(m_1, m_2, ..., m_n) + \epsilon \quad (1)$$

To create S-metric, area, and power models for a system, an SS-ANOVA model for each design objective is generated for locking in one module at a time. This constitutes a greedy algorithm that assumes each design objective is separable per module. For power and area, this is largely true. However, for a system with multiple locked modules, the locking in each module may interact, impacting the system S-metric. To model this, we adopt an iterative approach, constructing models for one module at a time and freezing the locking configuration before considering the next locked module. For example, consider a system with L locked modules. In this case, we randomly select the first module and apply the DSM algorithm to generate S-metric, area, and power models with SS-ANOVA. After the locking configuration is selected for that module, it is frozen. Models for the next locked module are generated assuming all prior locking is already in place. This process proceeds iteratively until a locking solution is found for all L modules. For each SS-ANOVA model, we include all first and second order terms. This produced the best models in our exhaustively simulated ALU (see Sec. 6.2).

5.3.2 SAT Attack Runtime Many locking schemes are designed to resist SAT attacks [10, 12]. This makes quantifying SAT runtime infeasible for many locking solutions, limiting the regions of the design space that can be used to build models. As a result, we adopt a different approach for runtime modeling, relying on prior theoretical derivations of SAT attack complexity for model generation [19, 22]. Specifically, we use theoretically derived SAT attack complexity to select a regression function for each locking scheme. For example, the Anti-SAT locking scheme [16] proved that SAT attack iteration count, which is proportional to runtime, grows exponentially in key length. Hence, an exponential regression function is selected to model SAT runtime for Anti-SAT.

If a module has multiple locking schemes, the model for each scheme is summed to predict total SAT runtime for the locked module[2]. Consider the following example without the loss of generality. To produce a SAT runtime model for a module being locked with 3 locking schemes (SFLL [10], CASLock [12], and SLL [17]) with key, $\mathbf{k} = (k_{SFLL}, k_{CASLock}, k_{SLL})$, we define the regression function in Eqn. 2. In Eqn. 2, each locking scheme has its own term, whose form (e.g., exponential, linear, etc.) is determined by [19, 22], with added constants $(A, ..., F)$ that are determined through regression.

$$SAT(\mathbf{k}) = A * 2^{B*|k_{SFLL}|-C} + D * 2^{E*|k_{CASLock}|} + F * |k_{SLL}| \quad (2)$$

To construct a SAT runtime model using the regression function in Eqn. 2, locking configurations using the γ smallest key lengths for each locking scheme are generated and SAT attacked. γ is a user-defined parameter. The SAT runtime model is then fit based on these simulated points. Despite only simulating low SAT complexity solutions, accurate models can still be produced because the theoretical results ensure similar growth/tail behavior. This enables the prediction of SAT runtime in regions of the design space where simulation is infeasible. The accuracy of these SAT runtime models is empirically demonstrated using exhaustive simulation in Sec. 6.2.

5.4 Iterative Selection of Simulation Points

Near-optimal regions in the design space based on design objectives are more important to a designer than those farther away, which are unlikely to produce a high-quality solution. As a result, directed simulation is performed by drawing a region of interest (ROI) in the design space from which future simulations are selected to update the model. This ROI-based approach mirrors those used by prior work exploring DSM for 3D ICs [11]. By generating more data points in these near-optimal regions of the design space, the model becomes more accurate in these regions.

The data points used for model update are selected as follows. From the set of all points in the design space, Ω, the designer aims to discover an optimal configuration based on user-specified objectives. Without loss of generality, we assume the design objectives are to satisfy an S-metric, SAT runtime, and area constraint while minimizing power overhead. To select the data points for directed simulation, we define an ROI with Eqn. 3. This ROI is the set of design points, $i \in \Omega$, with S-metric, SAT attack runtime, power, and area $(\{S_i, T_i, P_i, A_i\})$ within distance $\Phi = \{\phi_S, \phi_T, \phi_P, \phi_A\}$ of the

[2]For compound locking, prior work showed that keys for SAT-weak locking could be found quickly and independently from other locking [1], indicating additive behavior.

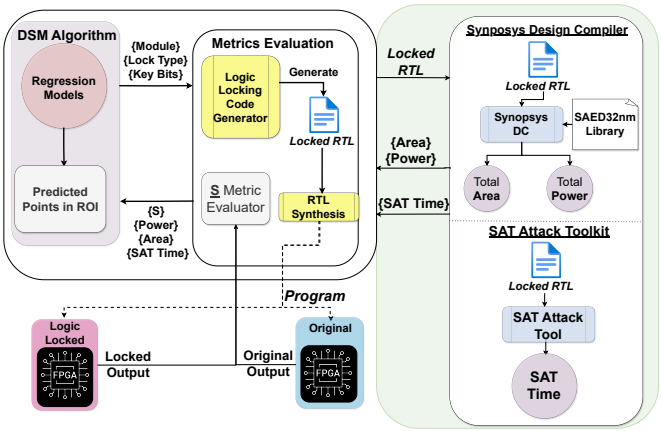

Figure 3: Overview of DSM and simulation framework developed to evaluate DSM for system-level locking configuration.

optimal point predicted by the model, d. Up to M points with the best-predicted quality are selected from the ROI at each iteration. These points are simulated and used to update the model.

$$ROI = \left\{ i \in \Omega \;\middle|\; |S_d - S_i| \leq \phi_S \wedge \left|\frac{P_d - P_i}{P_d}\right| \leq \phi_P \wedge \left|\frac{A_d - A_i}{A_d}\right| \leq \phi_A \wedge \left|\frac{T_i - T_d}{T_d}\right| \leq \phi_T \right\} \quad (3)$$

5.5 Stopping Criteria

Stopping criteria determine when modeling halts. We consider 2 criteria. 1) A sufficient portion of the design space was sampled to avoid model overfitting. This is user-defined, however, for the evaluation (Sec. 6) it is defined as 5% of the design space. 2) The ROI converges (i.e., the same M points are selected as the ROI in consecutive iterations). This indicates that the model has converged to a local minima. Once a stopping criterion is met, modeling is halted and the predicted optimal system locking solution is returned.

6 Experimental Evaluation

We evaluate our DSM framework with 2 experiments using a RISC-V processor benchmark [15]. First, we perform an exhaustive simulation of the locking design space in the ALU to evaluate the accuracy of our design space models. Second, we apply our DSM framework to lock the RISC-V core using an arbitrary cost function. We then compare the locking solution identified by our model-based approach to a conventional module-level locking approach.

6.1 Experimental Setup and Simulation Flow

The DSM framework used for the experimental evaluation is outlined in Fig. 3. Key aspects of this setup are summarized below.

Design Objective (Cost Function): The design objective (cost function) is defined by Eqn. 4. The goal is to identify a locking solution, i, from the design space, Ω, with the lowest power overhead, P, subject to a constraint on 1) area overhead, $A_o = A_{lock}/A_{no\_lock} \leq 2\%$, 2) S-metric, $S \leq 10^{-4}$, and 3) SAT attack runtime, $T_i > 7$ days.

$$\{i \in \Omega | A_{o_i} \leq 2\% \wedge S_i \leq 10^{-4} \wedge T_i \geq 7days \wedge min(P_i)\} \quad (4)$$

This cost function is defined without the loss of generality as our DSM framework can be used with any desired objective.

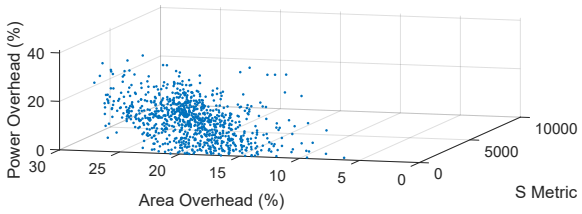

Figure 4: Exhaustive characterization of the logic locking design space for a single locked module (RISC-V ALU).

DSM Parameters: The initial models are created from $\eta = 25$ uniformly selected points. During each iteration, the $M = 5$ points with the lowest predicted cost are selected from the ROI to update the model. The ROI is defined using the radii: $\Phi = \{\phi_S, \phi_T, \phi_P, \phi_A\} = \{3 \times 10^{-5}, 2\%, 10\%, 5\%\}$.

Logic Locking Configuration: Locked modules are produced using a custom logic-locking code generator. While any locking scheme could be considered, we selected SFLL [10], CASLock [12], and SLL [17] to provide a cross-section of locking approaches.

System Security Metric (S): A 50,000 input trace was applied to the processor to estimate S using the simulator outlined in Sec. 4. This trace was the set of all unit tests for the RISC-V core [15].

SAT Runtime: Estimates are made using the SAT attack in [14].

Power/Area Overhead: Estimates are produced using the Synopsys Design Compiler with the Synopsys 32nm SAED library. All evaluated designs were subject to a 100MHz clock constraint.

6.2 Experiment 1: Exhaustive Characterization

To evaluate the accuracy of the design space models and locking solutions produced by our DSM algorithm, we performed an exhaustive simulation of a small logic locking configuration problem. Specifically, we considered the locking design space of a RISC-V ALU incorporating a 64-bit secret key (i.e., the sum of the key length for each locking technique in the ALU must be 64-bits). This constrains the design space to 905 possible locking configurations, making exhaustive simulation feasible. However, we note that even for this small example the simulation time exceeded 100 hours. This highlights the infeasibility of simulating large portions of the design space and motivates our modeling-based approach.

Logic Locking Design Space: Fig. 4 depicts the S-metric, power, and area overhead for all possible locking configurations in the ALU with a total key size of 64 bits. We make 3 observations based on these results. 1) Fig. 4 demonstrates the non-trivial nature of the locking design space, even with only a single locked module, highlighting the challenge of identifying optimal design configurations. 2) The overwhelming majority of points in the design space are far from an optimal solution, highlighting the importance of the system locking configuration problem addressed in this work. 3) The correlation between each design objective is weak, affirming our decision to model design objectives separately.

DSM-Predicted Solution Quality: Fig. 5 compares the *optimality* of the DSM-predicted solution to the percent of the design space that was simulated to produce it. We define the solution *optimality* as p_o/p_d, where p_o is the power consumption of the globally optimal solution found by exhaustive simulation and p_d is the power consumption of the DSM-predicted optimal solution. Note that because all other design objectives are fixed constraints that must be

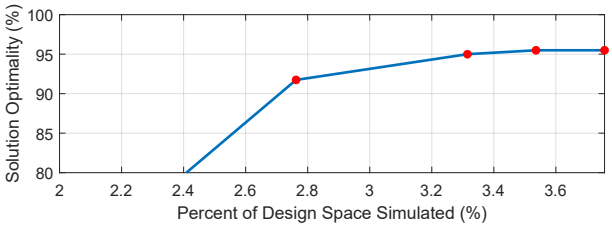

Figure 5: Percent of design space simulated vs. the optimality of DSM-identified locking solution in RISC-V ALU.

satisfied, only power consumption influences solution optimality. Unsurprisingly, solution quality improves as more points are simulated. When the DSM algorithm stopping criteria were met, only 3.6% of the design space was simulated. The final locking solution exhibited a power consumption within 96% of the globally optimal solution and met all other design constraints. This supports the claim that our DSM algorithm can produce high-quality solutions after simulating only a small fraction of the design space.

Accuracy/Fit of Models: To assess model fit, we calculated the coefficient of determination (R^2) for the models produced by the DSM framework upon halting (i.e., after simulating 3.6% of the design space). R^2 was calculated using all points in the design space from the exhaustive simulation. The final DSM models exhibited a $R^2 > 0.99$ for S-metric, power, and SAT runtime, and a $R^2 > 0.98$ for area. These high R^2 values indicate that the DSM models are highly predictive of the locking design space they model.

6.3 Experiment 2: Full-System Evaluation

Next, we consider a complete system-level locking configuration problem. Specifically, we consider locking the complete RISC-V core [15]. In the RISC-V core, we have designated three combinational modules as critical IP that must be locked: 1) ALU, 2) branch predictor, and 3) decoder. We have allocated a 64-bit key budget for both the ALU and the branch predictor, and a 32-bit key for the decoder, resulting in a 160-bit key in the full system.

We applied our proposed DSM framework to identify a locking configuration for this RISC-V core. For comparison, we consider conventional, module-level locking where the 3 candidate logic locking schemes (SFLL [10], CASLock [12], and SLL [17]) are selected and implemented in each module as described by the authors. To assess solution quality, we compare each locking solution using our four design metrics and the cost function outlined in Sec. 6.1.

Results and Analysis: Tbl. 2 shows the S-metric, predicted SAT runtime, total design area, and locked module power consumption for the locking configuration produced by each approach. Based on the cost function (see Sec. 6.1), satisfied design constraints are shaded in green and failed design constraints are shaded in red. We make two observations based on these results. 1) Only the proposed DSM framework identified a locking configuration that met all design constraints. This is unsurprising given that conventional locking approaches consider only module criteria during configuration. 2) The DSM-identified locking configuration required 29.5% less power on average than the solutions identified by the conventional locking approaches. These results indicate that our DSM framework outperforms prior module-level locking strategies by identifying locking solutions that achieve system-level design objectives and substantially reduce power overhead.

Table 2: Design metrics for RISC-V locking configured by DSM and conventional approaches. Red cells fail design goals.

	DSM	SFLL [10]	CASLock [12]	SLL [17]
S-Metric	$2 * 10^{-5}$	10000	10000	$2 * 10^{-5}$
SAT Runtime	6.5 (yrs)	$6 * 10^{23}$ (yrs)	120 (yrs)	72 (s)
Area (μm^2)	6235.56	5967.71	6369.88	6325.62
Power (μW)	769.77	1270.37	890.38	829.32

7 Conclusion

We propose a DSM framework to generate system-level models of the logic locking design space in an IC by simulating a subset of the design space. These models can be used to automatically identify near-optimal locking solutions in an IC that achieve arbitrary design goals. To evaluate this framework, we applied it to a RISC-V core. The generated design space models were highly predictive, modeling the power, area, security, and SAT runtime of locking solutions with an average $R^2 > 0.99$. Moreover, the DSM framework identified system-level locking solutions consuming 29.5% less power on average than those from conventional approaches.

Acknowledgments

This work was supported by the National Science Foundation (NSF) under Grant 2245573.

References

[1] K. Z. Azar et al. 2021. From cryptography to logic locking: A survey on the architecture evolution of secure scan chains. IEEE.
[2] A. Chakraborty et al. 2020. Keynote: A Disquisition on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*
[3] L. Collini et al. 2022. A composable design space exploration framework to optimize behavioral locking. In *Design, Automation & Test in Europe.* IEEE.
[4] C. Gu. 2013. *Smoothing Spline ANOVA Models.* Vol. 297. Springer Science & Business Media.
[5] W. Jia et al. 2012. Stargazer: Automated regression-based GPU design space exploration. In *Symposium on Performance Analysis of Systems & Software.*
[6] C. Pilato et al. 2022. Optimizing the use of behavioral locking for high-level synthesis. *IEEE Transactions on CAD of Integrated Circuits and Systems.*
[7] L. Pilato et al. 2018. TAO: Techniques for algorithm-level obfuscation during high-level synthesis. In *Proceedings of the 55th Annual Design Automation Conference.*
[8] A. D. Pimentel et al. 2016. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Design & Test.*
[9] M. Rostami et al. 2014. A Primer on Hardware Security: Models, Methods, and Metrics. *Proc. IEEE* (2014).
[10] A. Sengupta et al. 2020. Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective. *IEEE TCAD* (2020).
[11] C. Serafy et al. 2017. Design Space Modeling and Simulation for Physically Constrained 3D CPUs. Association for Computing Machinery.
[12] B. Shakya et al. 2019. CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme. *Trans. on Cryptographic Hardware and Embedded Systems.*
[13] K. Shamsi et al. 2019. On the Impossibility of Approximation-Resilient Circuit Locking. In *Symposium on Hardware Oriented Security and Trust (HOST).*
[14] P. Subramanyan et al. 2015. Evaluating the security of logic encryption algorithms. In *Symposium on Hardware Oriented Security and Trust (HOST).* IEEE.
[15] ultraembedded. 2020. BiRISC-V: A Binary Translation RISC-V Core. https://github.com/ultraembedded/biriscv. Accessed: Feb. 2024.
[16] Y. Xie et al. 2019. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).
[17] M. Yasin et al. 2016. On Improving the Security of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2016).
[18] M. Yasin et al. 2019. SFLL-HLS: Stripped-Functionality Logic Locking Meets High-Level Synthesis. In *IEEE/ACM International Conference on CAD (ICCAD).*
[19] H. Zhou et al. 2019. Resolving the Trilemma in Logic Encryption. In *IEEE/ACM International Conference on CAD (ICCAD).*
[20] M. Zuzak et al. 2021. ObfusGEM: Enhancing Processor Design Obfuscation Through Security-Aware On-Chip Memory and Data Path Design. In *MEMSYS.*
[21] M. Zuzak et al. 2021. A Resource Binding Approach to Logic Obfuscation. In *58th ACM/IEEE Design Automation Conference (DAC).*
[22] M. Zuzak et al. 2021. Trace Logic Locking: Improving the Parametric Space of Logic Locking. *Transactions on CAD of Integrated Circuits and Systems* (2021).