Low Power Logic Obfuscation Through System Level Clock Gating

Daniel Xing, Yuntao Liu, Ankur Srivastava
University of Maryland, College Park, MD, USA
{dxing97, ytliu, ankurs}@umd.edu

2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) | 979-8-3503-1175-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISLPED58423.2023.10244561

***Abstract*—Logic locking methods such as Stripped Functionality Logic Locking (SFLL) tend to yield high overheads. SFLL only corrupts a small part of the input space by design in order to maintain good SAT resilience and in doing so selects high frequency inputs to corrupt (protect) and therefore increases locking's impact on system level error. This implies that much of the time stripped modules are doing unnecessary work while the restore units are correcting the computations. We propose taking advantage of this fact to selectively clock gate the modules when protected inputs are being processed. Under the highest possible level of attack resilience, this alone can yield up to 24.5% dynamic power savings when protected inputs are applied to synthesized MediaBench benchmarks. We also propose a system-level design approach that utilizes the data-flow graph to also gate operations that fully depend on other gated operations. In conjunction with modifying operation binding, this increases power savings to 32.9% under the same strict security constraints.**

***Index Terms*—clock gating, logic locking**

I. Introduction

As the cost of maintaining advanced technology IC foundries continues to rise, many chip designers have chosen to become fabless and rely on offshore foundries for fabrication. However, this outsourcing can jeopardize the security of the IC supply chain since the foundries are not controlled by the designer.

Logic locking has emerged as a protection of the intellectual properties in chip designs against untrusted fabs [1]. Logic locking involves a secret key input, known only to the designer, that must be correctly applied to the circuit for it to function properly. Various types of logic locking mechanisms have been proposed, starting with inserting XOR/XNOR gates in the design netlist [2] and progressing to more advanced techniques based on VLSI testing principles that produce high corruption at the output bits when an incorrect key is applied [3], [4].

The logic locking field was transformed by the Boolean satisfiability-based attack, also known as the SAT attack [5]. SAT offers a robust mathematical approach for identifying the correct locking key of a logic locked IC by progressively eliminating incorrect keys. In response, point function (PF)-based logic locking, such as SARLock [6] and Anti-SAT [7], limits the number of wrong keys pruned out in each iteration, resulting in an exponential increase in the number of SAT iterations required relative to the key size. However, such logic locking techniques were proven vulnerable to approximate SAT attacks [8], [9] and removal attacks [10].

In a more recent development, stripped functionality logic locking (SFLL) was proposed which empowers designers to choose a group of protected input patterns (PIPs) that are impacted by a significant proportion of incorrect keys, while other input patterns are affected by only a small percentage of incorrect keys [11]. Details of SFLL is introduced in Sec. II-A. Robust Strong Anti-SAT (RSAS) achieves the same level of security by also stripping the PIPs' functionality from the original circuit and using improved Anti-SAT infrastructure to restore the functionality when the correct key is applied [12].

Our proposed method gives SFLL the ability to lower power overhead. While this method can be applied to modules in isolation, we show that a system-level approach can enable even greater power savings for the same level of SAT attack resilience. Other system-level logic locking methods have been proposed in past works [13]–[17], including one work that proposes a system-level sharing method for reducing locking-related overhead [18]. However, our work is the first to propose clock gating entire functional modules locked with SFLL while incorporating a high level design perspective. Treating the look-up tables (LUTs) in SFLL as a high-level power-saving resource grants system designers greater design flexibility when considering power overheads while still maintaining granular control over SAT attack resilience.

A. Contributions

In this work, we propose utilizing SFLL's LUTs in a new way: enabling clock gating of entire modules by performing the lookup in an earlier clock cycle. While this technique can be readily applied at the module level, we demonstrate that a system-level design view enabled by high-level synthesis (HLS) can enable even greater power savings, all while keeping SAT attack resilience and LUT sizes in check. Our contributions can be summarized as follows:

1) We formally define finding the power-optimal operation binding and locking configuration as a 0-1 integer linear program (ILP). This formulation jointly determines which operations to protect (and therefore clock gate) with SFLL and operation-to-hardware binding so that power savings are maximized without compromising SAT resilience.
2) We also present a post-binding greedy heuristic for selecting protected operations and PIPs using system-level information, but implemented at a module level. This simplifies the control circuitry needed to implement clock gating at the expense of sub-optimal power reduction.
3) We evaluate our clock gating techniques on MediaBench benchmarks. Our heuristic and optimal methods on average reduce dynamic power by 24.5% and 32.9%,

This work was supported by the NSF under Grant 1953285.

respectively, while maintaining maximal SAT attack resilience. Further power reductions are possible under relaxed attack resilience requirements.

II. Preliminaries

A. SAT Attack and SFLL

The SAT attack provides a strong mathematical formulation and was able to defeat all the logic locking techniques that predated it. The potential adversary could be either an untrusted foundry or user with the capability to reverse engineer the produced chip and acquire the locked gate-level netlist. The SAT attack requires two resources: (1) the locked netlist and (2) an activated chip (one that has the correct key loaded) from which the adversary can query the correct output of selected input vectors. Details of the mathematical formulation of the SAT attack can be found in [5]. Simply put, in each iteration of the SAT attack, a Boolean satisfiability problem is solved, and a *distinguishing input (DI)* is found. The DI is capable of eliminating the set of wrong keys that produce incorrect output for the DI. When all the wrong keys are eliminated by DIs, a correct key will be found.

The advanced technique of Stripped Functionality Logic Locking (SFLL) comprises of two main components: a Functionality Stripped Circuit (FSC) and a Restore Unit (RU). The FSC is the original circuit, but with its functionality altered for a selected set of PIPs, rendering SFLL resistant to removal attack. Removing the RU renders the PIPs' functionality different from that of the original circuit, making the attack futile. The RU is often implemented with an LUT that stores the key, verifies the input of the circuit against the key, and produces a restore vector that is XOR-ed with the FSC output. If the key is correct, the restore vector will correct the FSC's output, resulting in a correct output for the circuit. SFLL comes in two types: SFLL-HD and SFLL-flex. SFLL-HD produces specific structural traces in the FSC, which can be captured through functional analysis-based attacks [19]. On the other hand, SFLL-flex leaves minimal structural traces in the FSC, thanks to a fault-injection-based approach used for stripping the functionality [20]. Of note is SFLL's inherent direct inverse relationship between the number of PIPs and the expected time required by SAT attack [21], [22]. More more PIPs lead to higher impact that SFLL has on the overall functionality of a locked circuit, doing so comes at the cost of decreased SAT attack resilience.

B. High-Level Synthesis

HLS, or high-level synthesis, is a process that involves transforming a high-level description of functionality, such as a behavioral description in a high-level language, e.g. C or SystemC, into a register-transfer level (RTL) design. During HLS, there are generally three main design optimizations: resource allocation, scheduling, and resource binding. Resource allocation involves determining the quantity and type of hardware resources, such as functional units (FUs), that are necessary for the design. Scheduling imposes clock-cycle boundaries on the target behavioral code to resolve data dependencies. This produces a scheduled data flow graph (DFG), a directed acyclic graph whose nodes and edges represent operations and dependencies between them, respectively. Resource binding maps operations in the scheduled DFG to the allocated FUs from resource allocation. The binding must meet the minimal timing and performance requirements of the design while trying to optimize for area or power. Common binding schemes aim to minimize area [23] or switching power [24], [25]. During binding, the expected input space for a circuit is generally known. This enables switching power estimation to inform power-aware binding decisions. Power-aware binding techniques aim to minimize the switching activity of FUs by selecting those that have a lower expected switching activity, thus reducing the overall power consumption of the design.

HLS-based technques have been exploited to strengthen SFLL. For example, the intermediate representation during the compilation of the high-level design code can be analyzed to identify suitable combinational logic cones to insert SFLL [26]. Furthermore, security-aware binding was proposed in [13] where the operations are selected to be bound to locked FUs in order to maximize the occurrence of PIPs in the locked FUs. In our work, we show that the binding step, i.e. assigning operations to SFLL-locked FUs, provides an opportunity to reduce the switching power for the entire design, and the security of logic locking and power savings can be achieved simultaneously without compromise.

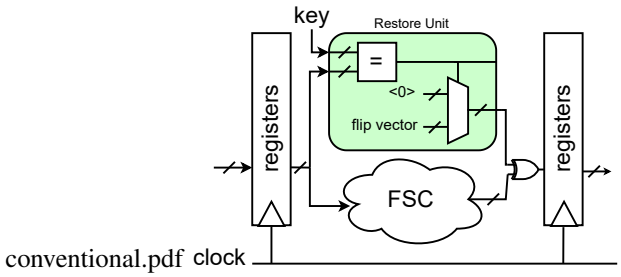
III. Gating, Graphs, and You: Saving Dynamic Power, Securely

Conventional logic locking methods always require some amount of additional circuitry to implement. SFLL-based methods in particular require a RU (typically implemented as a LUT) to correct stripped functionality when the correct key is applied. However, only a small number of protected inputs are chosen in order to strengthen SFLL against SAT attacks and to keep LUT sizes small. By moving LUT lookup into an earlier clock cycle (i.e. after module inputs are known but before inputs are applied), we can store precomputed output values and save them in the LUT, allowing the system to opportunistically clock gate entire locked modules.

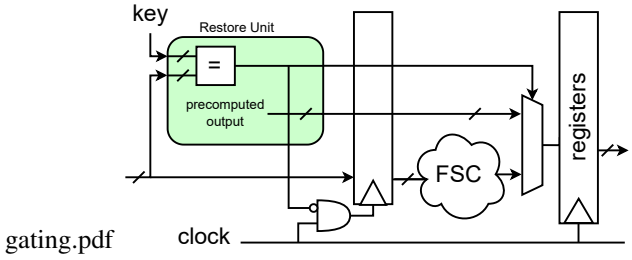
A. Module-Constrained Gating

To illustrate this idea, consider the locked single-module datapath shown in Fig. 1a. A subset of the locked module's input is compared against a stored key value that characterizes the PIP. If they match, then the output of the FSC will be inverted according to the stored flip vector. Since the functionality of SAT-secure stripped modules only differs from the original design for a few inputs, it is feasible to implement correction functionality using a small LUT, and indeed this practice is adopted by PIP-based locking schemes such as SFLL and RSAS. Note that during this operation, when the PIP are being processed by the LUT, the module itself is doing useless work and can therefore be gated.

Our proposed modification to the RU is illustrated in Fig. 1b. Instead of storing a flip vector, we store the module's output value corresponding to the protected input. Note that the RU is moved one clock step earlier and connected to the output of the module feeding the input FFs of the stripped module. Whenever the input to the RU matches the key, instead of correcting the corrupted output of the FSC, we clock gate the entire FSC and supply a precomputed output value to any modules or registers dependent on the FSC's output. Since the FSC's correct output value is completely known at lookup

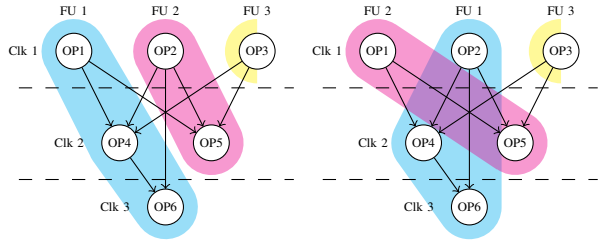


(a) Conventional restore unit architecture



(b) A clock-gating enabled restore unit

Fig. 1: Two single-module datapaths locked with SFLL. 1a places the RU in the conventional way, while 1b configures the RU to clock gate the FSC when protected inputs are applied.



(a) (b)

Fig. 2: Two different resource bindings for the same scheduled DFG.

time, the overall functionality of the system is not affected. We therefore perform the LUT lookup operation before the locked circuit is scheduled to operate, but after its input value is available. Since a SAT-secure locked circuit will protect only a few inputs, the LUT only needs to contain a few entries, minimally impacting timing. Provided the correct key values are loaded into the LUT, the locked module will operate correctly. Also, if the PIP are chosen such that they have high frequency of occurrence, the clock gating will be active often leading to large power savings.

B. System-Level Gating with DFGs

While this LUT look-ahead method can be readily applied within the confines of a single module on any synthesized design, we show that system-level modifications can enable more gating opportunities and provide the same guarantees against the SAT attack by considering data dependencies in the DFG and resource binding respectively.

Consider the scheduled DFG in Fig. 2a. We assume that a particular DFG input has been chosen for protection, so each operation in the DFG will already have a candidate protected input. Suppose the designer is limited to placing just two LUTs (e.g. due to area limitations). If PIPs for operations 2 and 4 are chosen to be placed in LUTs, then the system can clock gate up to *three* operations: operations 2, 4, and 6. The protected inputs of operations 2 and 4 are both stored by LUTs, and can be directly gated. Operation 6 can be gated if both operations 2 and 4 are also gated because the input of operation 6 is fully determined by the outputs of operations 2 and 4, which are also stored in our modified LUTs. Therefore, the output value for operation 6 when both operation 2 and 4 receive protected inputs is fully known at design time, and can be saved on-chip. On the other hand, if inputs of operations 1 and 2 are chosen for protection, then the number of gated operations drops from three to two when the protected DFG input is applied, since no operations in the DFG depend on only operations 1 and 2. Hence judicious allocation of limited restore LUT resources to operations impacts how many operations can be gated.

Binding of operations to hardware modules also affects the SAT-secureness of each locked module because each locked module's PIPs are chosen based on which operations are bound to each module. For example, consider the binding shown in Fig. 2a. Again assuming operations 2 and 4 are protected, modules FU1 and FU2 will only need to strip one input each, and therefore are maximally secure against SAT attack. If the binding is instead what's shown in Fig. 2b, then FU1 will need to strip the protected inputs of both operation 2 and 4, worsening its resilience against SAT attack (since we will need to protect two PIPs instead of one), while FU2 will not be locked at all. Therefore, binding should be carefully done to ensure that each module does not have too many protected inputs and remains sufficiently secure against SAT attacks.

This HLS-driven system-level approach is the one we take when clock gating modules. We select operations to protect and map protected operations to locked functional modules such we maximize the number of gated operations under protected inputs and ensure locked modules remain sufficiently secure against SAT attack.

IV. Finding Power-Optimal LUT and Binding Configurations with ILP

Finding locking configurations that maximize power saved in this way can be expressed as a 0-1 integer linear program (ILP). To help make our formulation easier to follow, Table I lists constants, variables, and some notation used in our formulation.

First, the constraints. Besides operations with LUT-protected inputs, we also want to gate operations that depend only on other gated operations. To start, we first describe operations that only depend on other operations and not DFG inputs ($NI(V)$). The two constraints calculates whether a node's predecessors are all gated and are not all gated respectively:

$$\prod_{i \in pre(k)} g_i \leq p_v \quad \forall v \in NI(V) \tag{1}$$

$$1 - p_v \leq n_v \quad \forall v \in NI(V) \tag{2}$$

Note that while constraint 1 is not linear, since all variables are constrainted to be binary, it can be rewritten as a set of linear constraints.

If every operation that v depends on is gated, then the inputs of v can be fully determined from stored LUT entries, and

TABLE I: ILP variables, constants, and definitions

Notation	Definition
$G(V, E)$	a DAG representing the scheduled DFG, where nodes represent operations and edges represent data dependencies
$pre(v)$	predecessors/data dependencies of a node/operation $v \in V$
$NI(V)$	Operations in G that depend only on other operations (i.e. don't depend on DFG inputs)
$start(v)$	The first clock cycle that operation v is active for
$end(v)$	The last clock cycle that operation v is active for
$w(v)$	Expected dynamic power used by operation v
M	set of functional modules
K	set of clock cycles in the schedule
l	User-defined limit on the total number of instantiated LUTs
C_m	User-defined limit on the number of LUTs allowed for each module m, this represents the number of PIPs stripped from m which is decided by the level of SAT attack resilience desired
g_v	Variable that encodes if an operation v is gated. g_v is 1 if operation v is gated, 0 otherwise.
p_v	Variable that encodes whether or not predecessors of v are gated. p_v is 1 if all predecessors are gated, 0 otherwise.
n_v	Logical inverse of p_v
L_v	Variable for keeping track of what operation a LUT is protecting. L_v is 1 if operation v's input is protected by a LUT, 0 otherwise.
$b_{v,m}$	Variable associated operation binding. $b_{v,m}$ is 1 if operation v is bound to functional module m, 0 otherwise.
$busy_{v,m,k}$	Variable for keeping track of the schedule during binding. $busy_{v,m,k}$ is 1 if operation v is scheduled during clock k and is bound to module m.

therefore v itself can also be gated. This can be expressed as the following constraint:

$$p_v \leq g_v \quad \forall v \in NI(V) \tag{3}$$

If not every parent operation of v is gated, then the output of v cannot be determined from LUTs assigned to parent operations alone. However, it can still be gated if a LUT is assigned to protect it. We can express this as the following constraint, again noting that it can be rewritten as a set of linear constraints:

$$g_v n_v \leq L_v \quad \forall v \in NI(V) \tag{4}$$

For operations v that do depend on DFG inputs (i.e. $V \setminus NI(V)$) then the inputs of v cannot be fully known by examining inputs and outputs of other operations alone. Therefore, v can only be gated if a LUT is assigned to it:

$$g_v \leq L_v \quad \forall v \in V \setminus NI(V) \tag{5}$$

For area-limited designs, we can constrain the total number of LUTs to some user-defined limit l with the following inequality:

$$\textstyle\sum_{v \in V} L_v \leq l \tag{6}$$

We use the following constraints to ensure that operations are properly bound to functional modules. Constraint 7 determines which which operations at what clock cycles each module is bound to:

$$b_{v,m} \leq busy_{v,m,k} \quad \forall v \in V, m \in M, k \in [start(v), end(v)] \tag{7}$$

Constraint 8 ensures that there can only be at most one operation bound to a module in any clock cycle:

$$\textstyle\sum_{v \in V} busy_{v,m,k} \leq 1 \quad \forall m \in M, k \in K \tag{8}$$

Constraint 9 ensures that every operation is bound to exactly one module:

$$\textstyle\sum_{m \in M} b_{v,m} = 1 \quad \forall v \in V \tag{9}$$

To ensure each locked functional module is still secure against SAT attack, we limit the number of LUTs (and therefore the number of PIPs per module) in each locked module. Since each LUT resource only protects a single input value, limiting the number of LUTs per module will also limit the number of PIPs, and therefore ensures locked modules will be sufficiently resilient against expected SAT attacks:

$$sum_{v \in V} b_{v,m} L_v \leq C_m \quad \forall m \in M \tag{10}$$

The overall optimization objective is to maximize expected dynamic power savings due to gated operations

$$\max_{g,p,n,L,b,busy} \sum_{i \in V} w_i g_i \tag{11}$$

subject to the constraints given.

A. ILP Usage

A system designer seeking to secure a design using lookahead LUT-based RUs will first need to perform the scheduling and resource allocation steps of HLS so that the scheduled DFG $G(V, E)$ and available functional modules M are known. One or more DFG inputs will need to be selected for protection, and should be propagated through the DFG so that each operation has one or more candidate protected inputs. The designer will also need to determine the maximum number of protected inputs that each module can protect (C_m) before SAT attack resilience degrades excessively. If the design is overhead constrained, then the maximum number of LUTs that can be instantiated l without exceeding area or static power limits will need to be determined.

Once these system parameters have been found, they can be encoded into the ILP described previously, and solved using one of the many available academic or commercial ILP solvers. Optimal solutions of g_v, L_v, and $b_{v,m}$ indicates which operations should be gated, which operations should have its input protected by an LUT, and which operations should be bound to which modules respectively.

V. Post-Binding Module-level Heuristic

While in practice ILP solvers can efficiently solve some kinds of large ILP instances, the worst-case runtime is still NP-hard. To avoid this worst-case time complexity and to simplify the control circuitry needed to implement optimal gating of downstream operations, we present a greedy PIP selection heuristic that can be implemented after operation binding has occurred.

While gating downstream operations increases the number of clock gating events without adding additional LUTs, proper implementation requires additional control circuitry to coordinate LUTs across different modules. As an example, again consider the scheduled and bound DFG shown in Fig. 2a. Suppose a particular DFG input is selected for obfuscation. While it may be the designer's intention to protect particular inputs of operation 2 and 4 (denoted as PIP_2 and PIP_4 respectively) calculated from the protected DFG input, in practice other DFG inputs may result in PIPs occuring at other clock cycles besides the ones operations 2 and 4 are scheduled for. For example, there may be a DFG input where PIP_2 occurs at FU2 in clock 1 but PIP_4 does *not* occur at FU1 in clock 2, or PIP_4 occurs in clock 1 of FU1 instead of clock 2. We cannot assume PIPs will occur at only in their intended scheduled operation, so some additional control logic is needed to make sure downstream operations with unprotected inputs

Algorithm 1 Greedy heuristic algorithm

Input: $G(V, E), w(\cdot), M, l, b, C$
Output: $L(v, m)$
Initialization:
1: $candidates = V$
2: $L(v, m)$ is initialized to 0 for all $v \in V$ and $m \in M$
3: $lcount = 0$
Loop:
4: **while** $lcount < l$ **do**
5: v_{sel} = highest weighted op in $candidates$
6: **if** $\sum_{w \in V} L(w, b(v_{sel})) < C_{b(v_{sel})}$ **then**
7: $L(v_{sel}, b(v_{sel})) = 1$
8: $lcount = lcount + 1$
9: **end if**
10: **end while**
11: **return** L

are only gated when all upstream operations are gated. Not gating downstream operations removes the need for control circuitry beyond what is already required by a single module at the expense of decreased power savings.

If operation binding is fixed and clock gating is confined to just the protected operation, then only protected inputs for each module need to be selected. This can be done efficiently with the greedy algorithm shown in Alg. 1. The algorithm first sets the pool of candidate gatable operations to V, (line 1) initializes the binding matrix $L(v, m)$ to 0 (line 2), and sets the total number of allocated LUTs to 0 (line 3). In each loop iteration, the operation v_{sel} with the highest dynamic power consumption is chosen (line 5). If the module that v_{sel} is bound to $b(v_{sel}) \in M$ has not exceeded its SAT-attack resiliency requirement $C_{b(v_{sel})}$ (line 6), then v_{sel} can be added to the set of operations protected by $b(v_{sel})$ (line 7). The loop terminates when all LUTs have been assigned (line 4).

The initialization requires sorting all operations in V by their dynamic power consumption, which takes $O(|V| \log |V|)$ time. The algorithm loop runs at most $|V|$ times (the highest value l can be set to) and the summation in line 6 can be implemented in $O(1)$ time if the length of each row of L is stored and updated separately from L. Therefore the heuristic's runtime is $O(|V| \log |V|)$.

VI. Results

To evaluate our optimal and heuristic clock gating approaches, we locked designs synthesized from C functions used in the MediaBench suite [27]. Each function's DFG was extracted using SUIF and scheduled using a path-based scheduler [28]. Up to three adders and three multipliers were allocated for each benchmark.

Locking was performed with SFLL-flex, although our method can be applied to any locking method that uses restore units. While SFLL PIPs can be of any bit width, we set the width to be the same as the input vector size of each functional module to maximize SAT attack resilience and simplify LUT design. For similar security reasons, we limited each locked module to strip at most one input (i.e. $C_m = 1 \forall m$). To evaluate the heuristic method, we used a security-aware binding method [13] to bind each operation to functional modules.

We implemented the ILP using Gurobi [29]. All experiments were performed on a desktop machine equipped with a 2.5 GHz Intel processor and 16 GB of system memory. Dynamic power was calculated for designs built using FreePDK45 [30] and synthesized with Cadence Genus tools.

To compare our techniques with the conventional no-gating approach, Fig. 4 shows dynamic power consumed by a locked datapath when a protected DFG input is applied to the system, normalized to the conventional no-gating method. Our optimal system-level approach reduces dynamic power consumed by the locked datapath under protected DFG inputs on average by 32.9% while ensuring that each locked module only strips one input, maximizing SAT attack resilience. Performing clock gating using our module-constrained heuristic at the same security level yields 24.5% dynamic power savings, although the advantage our heuristic holds over the ILP binding approach strongly depends on the structure of the DFG. For example, all operations in `jctrans2`'s DFG depend directly on a DFG input, so operations can only be gated via a LUT lookup, and cannot depend exclusively on preceding operations alone. However, both `fir` and `dct` have tree-like DFGs with multiple operations that do not directly depend on DFG inputs, so a system-level DFG-aware approach is able to save more power than a module-level one.

We can also explore the security-power tradeoff by varying C_m of each allocated module, shown in Fig. 3. Here, we assume that the system design allows enough area overhead to add any additional LUTs needed to implement restore units. When a small number of PIPs protects each module, the opportunities for clock gating decrease and so the dynamic power savings are less. Conversely, increasing the number of protected inputs per locked module increases the number of protected operations, and therefore the number of gated operations. At very high LUT allocation limits, every operation that depends on DFG inputs can be protected, and therefore gated, which in conjunction with gating downstream operations, brings dynamic power down to near-zero for protected inputs. Naturally, this is an extreme case since for such a scenario, much of the power will be dissipated in the LUTs themselves. For the heuristic module-limited gating method, this will happen when every operation in the DFG has a LUT protecting it. However, since there is a direct inverse relationship between the number of locked inputs and expected SAT iterations required, these additional power savings comes at the cost of decreased SAT attack resilience and increased LUT power dissipation.

VII. Conclusion

In this work we propose using SFLL restore unit lookup tables to store locked module outputs and therefore clock gate entire locked modules to reduce dynamic power consumption. We show that a system-level design approach guided by operation data dependencies can yield power savings beyond what a module-level approach can save. We demonstrate both a module-constrained and a system-level clock gating approach on synthesized designs from the MediaBench [27] suite. Our proposed module-constrained and system-level approach reduces dynamic power for protected inputs by 24.5% and 32.9% respectively compared to the non-gated locking method.

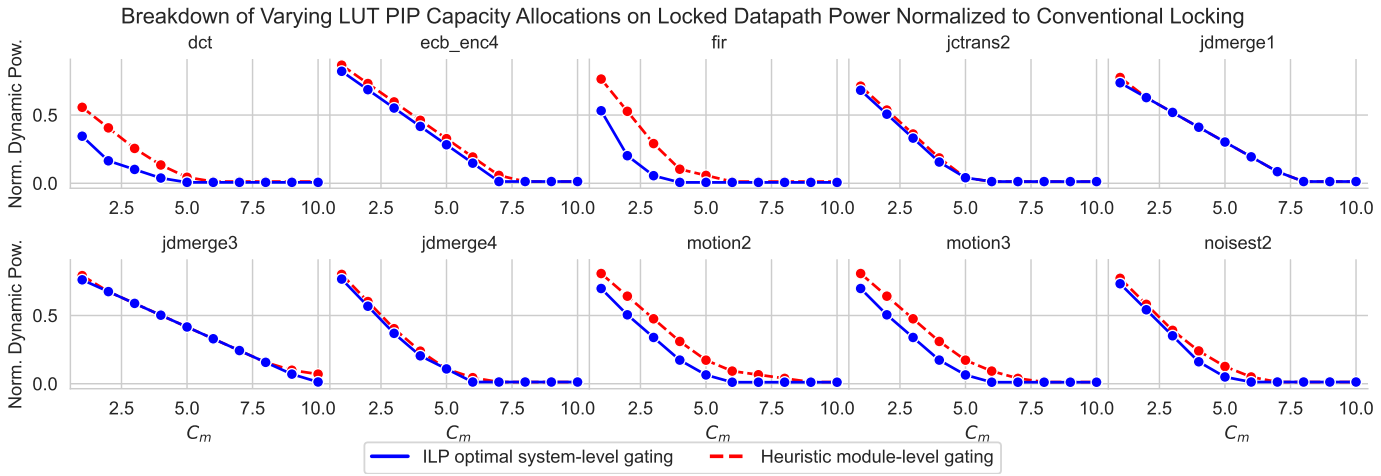


Fig. 3: Breakdown of how system dynamic power decreases as limits on the number of PIPs per locked module (C_m) are raised for both module-level heuristic and system-level ILP methods. The same C_m value is set for all allocated functional modules. Power is normalized to a system locked with a conventional no-gating appraoch.

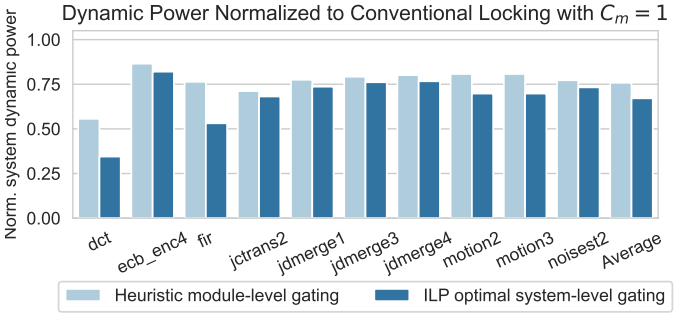


Fig. 4: Locked datapath dynamic power for benchmarks tested, normalized to a conventionally locked datapath. Data is shown for locking solutions found with $C_m = 1$ for all modules.

References

[1] A. Chakraborty *et al.*, "Keynote: A disquisition on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[2] J. A. Roy *et al.*, "Epic: Ending piracy of integrated circuits," in *Conference on Design, automation and test in Europe*, 2008.

[3] J. Rajendran *et al.*, "Security analysis of logic obfuscation," in *Proceedings of Design Automation Conference*, 2012.

[4] ——, "Fault analysis-based logic encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2015.

[5] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 137–143.

[6] M. Yasin *et al.*, "Sarlock: Sat attack resistant logic locking," in *Intl. Symposium on Hardware Oriented Security and Trust*, 2016.

[7] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.

[8] K. Shamsi *et al.*, "Appsat: Approximately deobfuscating integrated circuits," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 95–100.

[9] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Great Lakes Symposium on VLSI 2017*, 2017.

[10] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *Transactions on Emerging Topics in Computing*, 2017.

[11] ——, "Provably-secure logic locking: From theory to practice," in *Conference on Computer and Communications Security*, 2017.

[12] Y. Liu *et al.*, "Robust and attack resilient logic locking with a high application-level impact," *ACM Journal on Emerging Technologies in Computing Systems*, 2021.

[13] M. Zuzak *et al.*, "A resource binding approach to logic obfuscation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 235–240.

[14] C. Pilato *et al.*, "On the optimization of behavioral logic locking for high-level synthesis," *arXiv preprint arXiv:2105.09666*, 2021.

[15] M. R. Muttaki *et al.*, "Hlock: Locking ips at the high-level language," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 79–84.

[16] C. Pilato *et al.*, "Assure: Rtl locking against an untrusted foundry," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 7, pp. 1306–1318, 2021.

[17] N. Limaye *et al.*, "Fortifying rtl locking against oracle-less (untrusted foundry) and oracle-guided attacks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 91–96.

[18] D. Xing *et al.*, "Low overhead system-level obfuscation through hardware resource sharing," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023, pp. 1–8, ISSN: 1948-3295.

[19] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[20] A. Sengupta *et al.*, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[21] M. Zuzak *et al.*, "Trace logic locking: Improving the parametric space of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[22] H. Zhou *et al.*, "Resolving the trilemma in logic encryption," in *International Conference on Computer-Aided Design (ICCAD)*, 2019.

[23] C.-Y. Huang *et al.*, "Data path allocation based on bipartite weighted matching," in *Design Automation Conference*, 1991.

[24] J.-M. Chang and M. Pedram, "Register allocation and binding for low power," in *ACM/IEEE Design Automation Conference (DAC)*, 1995.

[25] A. Stammermann *et al.*, "Binding allocation and floorplanning in low power high-level synthesis," in *International Conference on Computer Aided Design*. IEEE, 2003.

[26] M. Yasin *et al.*, "Sfll-hls: Stripped-functionality logic locking meets high-level synthesis," in *Intl. Conf. on Computer-Aided Design*, 2019.

[27] C. Lee *et al.*, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *International Symposium on Microarchitecture*. IEEE, 1997.

[28] S. Ogrenci Memik *et al.*, "A super-scheduler for embedded reconfigurable systems," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, Nov. 2001, pp. 391–394, iSSN: 1092-3152.

[29] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[30] J. E. Stine *et al.*, "FreePDK: An Open-Source Variation-Aware Design Kit," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, Jun. 2007, pp. 173–174.