TIPLock: Key-Compressed Logic Locking using Through-Input-Programmable Lookup-Tables

1st Kaveh Shamsi
Electrical and Computer Engineering Department
University of Texas at Dallas, Richardson, Texas, USA
kaveh.shamsi@utdallas.edu

2nd Rajesh Kumar Datta
Electrical and Computer Engineering Department
University of Texas at Dallas, Richardson, Texas, USA
rajesh.datta@utdallas.edu

***Abstract*—Herein we explore using logic elements that can be programmed through their inputs for logic locking. For this purpose, we design a novel through-input-programmable (TIP) lookup-table (LUT) element and develop algorithms to find cuts in the circuit that can be mapped to such elements while maintaining programmability. Our proposed TIPLock flow achieves area savings of 50-70% compared to the traditional approach of using a key-vector-long scan-chain.**

***Index Terms*—Logic locking, circuit obfuscation, Hardware security, Through-input programmable**

I. Methodology

Logic Locking involves making an original design semi-programmable to hide its precise functionality from an untrusted foundry [1]. This locked design is then configured/unlocked by a secret key post-fabrication.

TIP-LUT. Fig. 1b shows our proposed locking TIP-LUT. Two programming transistors with programming bits (PBs) L_p/L_n are added to the tip of the multiplexer (MUX) tree. By turning one of the programming transistors on, and configuring the MUX-tree we can program one of the 2^n key bits at a time. Turning both off allows for normal post-programming operation. Spice simulations of this modified LUT showed only a slight ($<$10%) degradation in delay/power compared to the scan-chain (SC)-LUT in the FreePDK45nm model library.

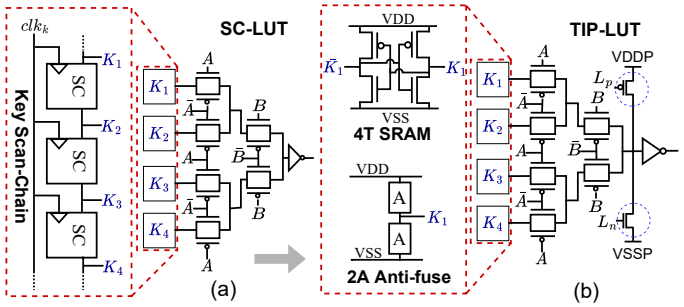

Fig. 1. a) Scan-chain 2-input (4-configuration bits) LUT with a transmission-gate MUX-tree where key bits are shifted in through scan-cells SCs (e.g. DFF with non-volatile storage). b) The TIP-LUT which adds two programming transistors (L_p/L_n gate signals) to program the key bits. It can be implemented with SRAM or anti-fuse saving area compared to DFF scan-cells.

Finding TIP-Eligible Cuts. To lock the circuit using the above TIP-LUTs, fanout-free cones with n inputs (i.e. isolated n-cuts) in the circuit need to be picked and replaced with n-input TIP-LUTs. The inputs of these cuts must be able to take on all 2^n different combinations to guarantee full programmability of internal TIP-LUTs via the circuit's primary inputs. Such a cut is called a TIP-eligible cut.

Given a current cut partitioning/assignment for the circuit, we identify TIP-eligible cuts using several techniques: 1) Simulating random patterns on the circuit and identifying cuts whose inputs take on all possible patterns during this process. 2) Using a SAT solver to check if all 2^n input combinations of a cut can be asserted via the primary inputs. 3) A hybrid approach in which an initial round of random simulation is followed by the SAT-based procedure.

Sharing Programming Bits. Given two TIP-LUTs in the circuit, if they can be fully programmed using a programming vector sequence, such that at each step, the programming bits (L_p/L_n) of the two LUTs happen to be the same, then their programming bits can be merged, saving additional programming structure area. If two TIP-LUTs cannot be programmed this way, we say that they conflict. We can add all the conflicts among the TIP-LUTs to a so-called conflict graph. It is possible to show that the number of distinct programming bit pairs (called programming domains) that are needed to fully program all LUTs is the minimum number of colors needed to color this conflict graph. This is the famous minimum graph coloring problem which is NP-complete, but for which good heuristic non-optimal algorithms exist that we utilize here.

As for building the conflict graph, we use two approaches. First, is a simple structural test: if the two cuts depend on non-intersecting sets of primary inputs, then they can be controlled independently, and hence do not conflict.

SAT-based Simultaneous Conflict and Eligibility Mining. Second, which is a formal approach, is to construct a SAT problem, that captures attempting to program a target TIP-LUT in the circuit to a one/zero while landing every other TIP-LUT in the circuit into a similar one/zero-programming address. If this problem is not satisfiable, its UNSAT core can be studied to find the offending cuts, adding them as conflicts, and repeating the process. If the UNSAT core shows that the current cut itself is the source of the problem, this suggests that the cut itself may not be TIP-eligible. Therefore, this SAT-based procedure can simultaneously identify both conflicts and TIP eligibility.

See Fig. 2 for the overall TIPLock flow.

II. Experiments

We implemented our algorithms in C++20 using `g++` with `O3` optimization. Tests were run on a 128-thread Threadripper 3990X with 256GB of RAM, with a 2GB-per-thread memory limit. DFF/MUX cells from the Nangate45nm library were used for SC/TIP-LUT area calculations. Independent L_p/L_n were assumed to take up two scan cells (DFFs). ISCAS combinational circuits resynthesized to Nangate45nm were used as benchmarks. `neos` [2] was used for deobfuscation attacks.

978-3-9819263-7-8/DATE23/© 2023 EDAA

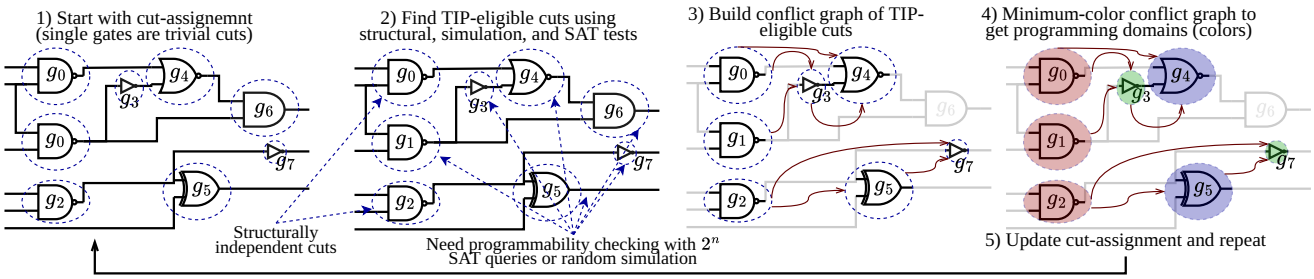

Fig. 2. TIPLock procedure. Identify gates/cuts that are TIP eligible, find conflicts, and minimum color the conflict graph to assign programming domains. The process can be repeated by making incremental updates to the cut assignment. The simultaneous SAT-based approach merges steps 2&3.

TABLE I

TIP-eligible cut-finding. rt: runtime (seconds), #tip: TIP-eligible cut count and fraction of total gates(=trivial cuts), #pd: number of programming domains.

			RandomSim+Structural			SAT+Structural			Simultanous SAT-based		
bench	#i/#o	#g	rt(s)	#tip	#pd	rt(s)	#tip	#pd	rt(s)	#tip	#pd
c432	36/7	161	0.01	151 (93.8%)	111 (73.5%)	0.03	160 (99.4%)	120 (75.0%)	0.17	160 (99.4%)	58 (36.2%)
c880	60/26	315	0.02	258 (81.9%)	174 (67.4%)	0.08	287 (91.1%)	203 (70.7%)	1.02	287 (91.1%)	118 (41.1%)
c1908	33/25	395	0.04	247 (62.5%)	166 (67.2%)	0.16	283 (71.6%)	202 (71.4%)	1.53	283 (71.6%)	142 (50.2%)
c1355	41/32	452	0.05	246 (54.4%)	132 (53.7%)	0.21	291 (64.4%)	177 (60.8%)	2.06	291 (64.4%)	111 (38.1%)
c499	41/32	464	0.05	258 (55.6%)	154 (59.7%)	0.22	290 (62.5%)	186 (64.1%)	2.02	290 (62.5%)	117 (40.3%)
c2670	157/63	637	0.08	459 (72.1%)	184 (40.1%)	0.31	462 (72.5%)	187 (40.5%)	7.02	462 (72.5%)	109 (23.6%)
c3540	50/22	910	0.29	634 (69.7%)	463 (73.0%)	0.72	634 (69.7%)	463 (73.0%)	15.30	634 (69.7%)	369 (58.2%)
c5315	178/123	1272	0.37	892 (70.1%)	232 (26.0%)	1.18	893 (70.2%)	233 (26.1%)	47.18	893 (70.2%)	163 (18.3%)
c7552	206/107	1532	1.01	1075 (70.2%)	936 (87.1%)	2.17	1081 (70.6%)	942 (87.1%)	46.60	1081 (70.6%)	612 (56.6%)
c6288	32/32	1881	1.73	636 (33.8%)	393 (61.8%)	3.75	636 (33.8%)	393 (61.8%)	62.10	636 (33.8%)	306 (48.1%)

TABLE II

TIPLock area overhead and deobfuscation time. TIP-x% means x percent of TIP eligible cuts (those from the largest programming domains first) were mapped to TIP-LUTs. #pd/#rc: final programming domain and replacement counts. ao-sc/tip: area overhead (in times) for the SC/TIP-LUTs respectively. dt: deobfuscation time, #di: number of OG queries. #k: number of keys. kerr: best key error rate (E: correct key via equivalence checking)

perc		20%							40%							80%						
bench	tip/g	#pd/rc	#k	ao-sc(x)	ao-tip(x)	dt(s)	#di	kerr	#pd/rc	#k	ao-sc(x)	ao-tip(x)	dt(s)	#di	kerr	#pd/rc	#k	ao-sc(x)	ao-tip(x)	dt(s)	#di	kerr
c432	160/161	3/32	78	4.44	1.32	2.83	228	E	11/64	222	13.1	4.46	2.79	118	E	36/128	580	34.9	13.1	TO	90	0.22
c880	287/315	7/57	182	5.12	1.6	1.86	66	E	24/114	444	12.8	4.49	15.2	99	E	76/229	1000	29.0	11.5	TO	33	0.19
c1908	283/395	7/56	168	3.67	1.16	0.38	42	E	27/113	440	9.82	3.59	77.7	197	E	86/226	998	22.5	9.26	TO	33	0.13
c1355	291/452	6/58	170	3.29	1.0	14.9	105	E	15/116	368	7.18	2.29	338.4	132	E	64/232	1028	20.6	7.69	TO	76	0.01
c499	290/464	4/58	162	3.09	0.87	44.8	131	E	15/116	406	7.91	2.52	TO	215	E	67/232	1064	21.1	7.96	TO	76	0.01
c2670	462/637	4/92	252	3.52	0.93	7.6	146	E	15/184	630	9.01	2.64	TO	1633	0.0	65/369	1372	19.8	6.7	TO	191	0.01
c3540	634/910	25/126	462	4.51	1.57	17.4	153	E	71/253	1004	9.88	3.77	150.1	273	E	243/507	2010	19.8	9.23	TO	27	0.34
c5315	893/1272	5/178	462	3.11	0.78	8.03	104	E	20/357	1168	8.06	2.25	88.3	199	E	85/714	2648	18.5	5.76	TO	198	0.0
c7552	1081/1532	30/216	754	4.26	1.37	211.9	398	E	121/432	1698	9.69	3.69	TO	153	0.01	396/864	3410	19.5	8.88	TO	39	0.06
c6288	636/1881	10/127	496	2.16	0.64	TO	39	0.01	27/254	992	4.32	1.33	TO	47	0.04	179/508	2728	12.1	4.68	TO	29	0.4

Locking Performance. Per Table I SAT-based procedures found the maximum number of TIP-eligible trivial cuts, while simple random testing surprisingly found on average more than 95% of the same. The simultaneous SAT-based procedure takes the most amount of time but achieves the lowest number of programming domains. All locking tasks were completed in less than two minutes and within the memory limit.

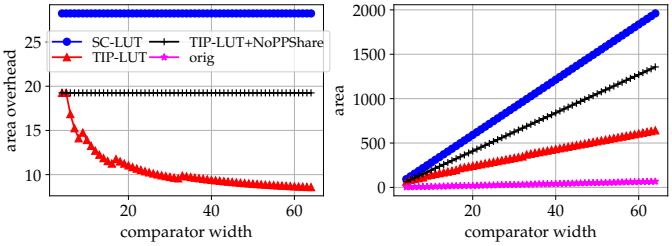

Fig. 3. Area overhead (in times) and area (in NanGate45 library units) of comparators of size 2 to 64 mapped to TIP and SC LUTs. The SC-LUT is dramatically larger. Program pin sharing ($\log n$ domains for n-bit comparators) has a big impact on area overhead as well.

Deobfuscation and Overhead. Table II shows area overheads and oracle-guided (OG) SAT attack times [3] for circuits locked with varying percentages of TIP-eligible cuts mapped to TIP-LUTs, with those in the largest programming domains going first. We can see an overall 50-70% reduction in area overhead for the TIP-LUT compared to the SC-LUT. Even small ISCAS benchmarks can overwhelm the SAT attack when mapping large portions of their gates to (TIP)-LUTs.

Per Fig. 3 we locked point-functions (PF, comparators) [1] with TIPLock as well. All primitive cuts in a binary comparator AND-tree are TIP-eligible, and the number of programming domains is always $\log n$ for n-input comparators. When at least the first XOR layer of the PF is mapped to TIP-LUTs, reliable exponential query counts (2^n for n-bit comparators) plus error rates higher than single-point PFs were observed under the oracle-guided SAT attack.

References

[1] M. Yasin, A. Sengupta, M. Ashraf, M. Nabeel, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM Conf. on Computer & Communications Security*, 2017.

[2] neos. http://www.bitbucket.com/kavehshm/neos.

[3] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pp. 137–143, IEEE, 2015.