Yoshan: Algebraic-Circuit Co-Synthesis and Search Methodology of Hardware-Efficient S-Boxes

Robert Hanhan, Binyamin Alony and Itamar Levi

Bar-Ilan University, Ramat Gan, Israel
{robert.hanhan, binyamin.alony, itamar.levi}@biu.ac.il

Abstract. Choosing a hardware S-box means balancing cryptanalytic strength (differential uniformity δ, linearity L, and algebraic degree) against circuit cost (AND count and AND depth D). Under masking, non-linear gates dominate silicon area, registers, and fresh randomness, while D sets pipeline latency. As a result, automated searches have largely retreated to quadratic permutations ($D = 1$), whose slow degree growth forces high cipher round counts.

We present YOSHAN, a co-synthesis engine that optimizes an S-box permutation and its gate-level circuit together across $n \in \{5, 6, 7, 8\}$. Inside a quality-diversity archive, YOSHAN seeds search with subfield towers, Feistel/Misty/$\mathcal{MM}$ networks, affine wrappers, CCZ graph automorphisms, and bilinear or shear constructions. It then evolves circuits via cryptographically guided DAG rewrites and exploratory differentiable Piling-Up seeding and crossover with Hungarian LSAP projection, before compressing them with SMT sub-DAG shaving, stage-preserving CSE, exact depth-1 SAT, and four-tier inverse compilation. Every archived circuit is exported gate for gate to SystemVerilog and certified by exhaustive simulation and a Z3 miter. Across 31,337 certified netlists (30,986 distinct permutations) in 56 $(n, \delta, L, \deg)$ profiles, EA invariants and an exact affine-equivalence solver against a 712-entry catalogue (power maps, 5- and 6-bit quadratic/quartic classifications, Feistel/MISTY/Bridge structures, two-point-swapped power maps, and cipher S-boxes) partition the archive into known catalogue classes (Category A, 2,681 permutations) and permutations certified EA-separated from it (Category B, 28,305 permutations; 12 benchmark classes $\mathsf{YSH}_{n,i}$). All 30 benchmark S-boxes and their explicit inverses are synthesized gate-exactly in 45 nm CMOS (unmasked and pipelined $d = 2$ DOM) and traced from seed to final netlist. For known classes, we obtain a gate-exact depth-3 AES-class pair (42 AND, 101 XOR; inverse 42 AND, 106 XOR at $D = 3$) and a provably AND-optimal 5-AND, $D = 1$ circuit for the 5-bit Q_{71} class ($\delta = 4, L = 16$). When the lookup table can vary, we find a 12-AND 6-bit degree-5 S-box $\mathsf{YSH}_{6,5}$ ($\delta = 6, L = 24, \beta = 14$), a 5-AND, $D = 2$ 5-bit quartic S-box $\mathsf{YSH}_{5,2}$ in both directions, and 28- and 18-AND 8-bit degree-7 S-boxes, all compared against the best published circuits.

Keywords: Substitution Boxes · Multiplicative Complexity · Algebraic Degree · Quality-Diversity Search · Hardware Synthesis · Masking

1 Introduction

Symmetric-key ciphers, hash functions, and cryptographic permutations rely on Substitution Boxes (S-boxes) $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ as their core source of non-linearity [Sha49]. A strong S-box combines low differential uniformity (δ) [BS91] and low linearity (L) [Mat94] with high algebraic degree ($\deg(S)$) to resist higher-order differential [Lai94, Knu95], division-property [Tod15], zero-sum [BCDC11], and cube attacks [DS09].

In side-channel-protected hardware and bitsliced software, S-boxes are Boolean circuits. Under t-th order masking (Threshold Implementations (TI) [NRR06], Domain-Oriented Masking (DOM) [GMK16], and Hardware Private Circuits (HPC) [CGLS21, FGMDP+18]), each wire splits into $d \geq t+1$ shares. Linear gates (XOR, NOT) act share-wise without randomness or registers, whereas each 2-input AND gate needs d^2 refreshed cross-share products and a register stage [GMK16, DCBG+17, LBS19]. Thus AND count drives masked area, randomness, and power, while **multiplicative depth** D sets cycle latency (Section 5.2).

1.1 The Search Problem and the Algebraic-Degree Compromise

Ideally, a designer would pick an S-box with both high algebraic degree and a shallow, low-AND circuit. Jointly optimizing $(\delta, L, \deg)$ and masked hardware cost (AND, D) faces three obstacles:

1. **Combinatorial explosion.** Beyond 4-bit S-boxes (302 affine classes [LP07]), widths $n = 5$ ($32! \approx 2.6\times10^{35}$) to $n = 8$ ($256! \approx 8.6\times10^{506}$) defy enumeration, and bijections occupy a vanishing fraction $2^n!/(2^n)^{2^n}$ of all functions.
2. **Fixed-target minimization.** Existing tools minimize a fixed table via Boolean Satisfiability (SAT) [Sto16], A* Multiplicative Complexity (MC) search [JBK24], whole-S-box Boyar–Peralta heuristics [JBKK25], segmented low-depth search [KBJ+26], or top-down polynomial division [BDM26a, BDM26b]. Yet division tables grow as $2^{\binom{n}{5}+\binom{n}{4}}$ and stop at degree 3 for $n = 7$ [BDM26b, p. 619], and joint SAT co-synthesis stops at $n = 4$ [JCL+25]. Crucially, fixed-target tools cannot trade a costly table for a cheaper permutation with the same or better cryptographic profile.
3. **The quadratic compromise.** To keep $D = 1$, automated searches [BDMD+20, BDM26a] restrict attention to quadratic S-boxes. Their slow degree growth forces many cipher rounds (24 in Keccak-f [BDPVA11] and 12 in Ascon [DEMS21], versus 5–7 in degree-5 SPEEDY [LMMR21]). We instead seek *low-depth, low-AND circuits for* $\deg \in \{3, \ldots, 7\}$ *S-boxes that cut total masked latency (rounds* $\times$ *D).*

1.2 Contributions

We introduce **Yoshan**, a co-synthesis engine uniting constructive algebra, Directed Acyclic Graph (DAG) rewriting, gradient relaxation, and formal synthesis to co-evolve an S-box S and its circuit DAG G:

1. **Algebraic-circuit co-synthesis (Section 4).** Yoshan searches in circuit rather than table space, starting from library circuits, subfield towers, or structural seeds (Feistel/Misty/$\mathcal{MM}$ networks, affine wrappers, Carlet–Charpin–Zinoviev (CCZ) automorphisms [CCZ98], and bilinear/shear seeds). Circuits evolve in a MAP-Elites archive [MC15] via guided DAG rewrites (AND-tree rotations, distributive folding, AND erasers) and auxiliary Piling-Up relaxation with Linear Sum Assignment Problem (LSAP) projection, then compress via Satisfiability Modulo Theories (SMT) cut-6 shaving, Paar Common-Subexpression Elimination (CSE) [Paa97], wire reuse [BP10], and depth-1 SAT.
2. **Certified export, inverse synthesis, and 45 nm silicon evaluation (Sections 5, 6.3, and 7).** Every archived DAG is exported gate for gate to SystemVerilog and certified by 2^n-input simulation and a Z3 miter [dMB08]. For all 30 benchmark S-boxes of Table 2, we compile explicit inverses via a four-tier algebraic and structural pipeline and print all 60 lookup tables and netlists in Supplementary Material (SM) C. We map all 60 circuits 1-to-1 onto NanGate 45 nm in Yosys (both unmasked and as D-stage pipelined $d = 2$ `DOM-indep` modules) and verify gate equivalence and timing in Berkeley ABC.

3. **Exact affine-equivalence classification (Section 6.1).** We classify every archived S-box using Extended Affine (EA) invariants and a Biryukov-style exact affine solver [BDCBP03] against a 712-entry catalogue. This catalogue covers power maps, the 75 quadratic 5-bit classes of [BBŞ17], the 5-bit quartic and 6-bit quadratic classes of [DMB19], 3-round Feistel/MISTY/Bridge structures [CDL16, BDMD+20], two-point-swapped power maps [BCC10, BKRS22], and cipher S-boxes. It partitions all 30,986 distinct archived permutations into certified catalogue members (**Category A**: 2,681 permutations with verified equality or (A, a, B, b) witnesses and zero unresolved collisions) and permutations certified EA-separated from it (**Category B**: 28,305 permutations, 12 benchmark classes $\mathsf{YSH}_{n,i}$). We also show that boomerang uniformity β separates our non-Feistel subspace-partition S-boxes ($\beta \in [12, 20]$) from Feistel-type S-boxes ($\beta = 2^n$).
4. **Benchmarking across $n \in \{5, 6, 7, 8\}$ (Section 6).** Across 31,337 certified netlists (30,986 permutations) in 56 profiles (traced from Stage 1 to Stage 4 in Table 10), we improve both known classes and flexible-table profiles. In Category A, we obtain a $D = 3$ AES-class pair (42/101/4 forward, 42/106/4 inverse, equal to FIPS-197 AES up to a wire permutation and realizing Hadžić and Bloem's depth-3 inverter [HB25]) that saves 66/60 `DOM-indep` flip-flops and 11.5%/8.9% area at 25% lower latency than our $D = 4$ FIPS-197 circuit, plus a provably AND-optimal 5/12/3 ($D = 1$) circuit for Q_{71} [BBŞ17]. In Category B, $\mathsf{YSH}_{6,5}$ (12/31/0, $D = 4$, $(\delta, L, \beta) = (6, 24, 14)$) improves on SPEEDY's $(8, 24, 36)$ while matching its native 12-gate NAND/NOR count ($D = 3$) [LMMR21] in (AND, XOR) and saving 8 AND and 20 XOR gates over its 20/51/3 ($D = 3$) circuit [BDM26b] at +1 level. We also find the 5-bit quartic $\mathsf{YSH}_{5,2}$ (5 AND, $D = 2$ both ways), 9-AND 6-bit quartic and cubic S-boxes with symmetric S/S^{-1} cost, and 8-bit degree-7 S-boxes $\mathsf{YSH}_{8,2}$ and $\mathsf{YSH}_{8,3}$ with 28 and 18 AND gates.

2 Related Work

Yoshan bridges three lines of research: fixed-target circuit minimization, joint S-box and circuit search, and affine/CCZ equivalence classification.

2.1 Circuit Minimization for a Fixed S-box

For a fixed S-box S, circuit synthesizers minimize MC (AND count), linear gate count (XOR and XNOR), combinational depth (longest topological path, LTP), or AND depth $D \geq \lceil \log_2 \deg(S) \rceil$ [BDM26b]. Early work exploited composite fields: Canright [Can05] decomposed the $\mathbb{F}_{2^8}$ AES inverter over $\mathbb{F}_{((2^2)^2)^2}$, and Boyar and Peralta paired MC reduction with linear heuristics (RNBP/BPD) to reach 32 AND gates [BP10] and depth 16 [BP12], later refined in [ME19]. Subsequent tools targeted general S-boxes via SAT [Sto16], cut-based XOR-AND graph (XAG) rewriting [TSADM19], gate-latency optimization [Ras22], low-D and shared S/S^{-1} templates [BDMD+20], and A* optimal-MC search for $n \leq 5$ and selected 6-bit targets [JBK24]. Most recently, Jeon et al. [JBKK25] extended Boyar–Peralta heuristics to whole S-boxes (eRNBP/eBPD) by rebuilding node-level XOR expressions over non-linear outputs at fixed $(\#\text{NL}, D)$, optimizing 8-bit AES (32–34 AND at $D \in \{4, 5, 6\}$), SNOW3G S_Q, and Saturnin's 16-bit super S-box. Kim et al. [KBJ+26] introduced SLICE, combining bitsliced subcircuit search over 2-fanin $\{\text{AND}, \text{OR}, \text{NAND}, \text{NOR}\}$ and $\{\text{XOR}, \text{XNOR}\}$ gates with bit-width segmenting, boundary merging, and refined eBPD to set low-depth records on AES (LTP $\in \{12, 13\}$ at $D = 3$, LTP $= 14$ at $D = 4$), 5-bit Ascon, Dillon's Almost Perfect Nonlinear (APN) map, and 7-bit x^3 (Table 4).

At the minimum AND depth $D = \lceil \log_2 \deg(S) \rceil$, Bolzer, Duval, and Minier [BDM26a, BDM26b] establish the (AND, XOR) baseline via depth-1 quadratic synthesis [BDM26a]

and top-down polynomial division $P = Q_1Q_2 + R$ [BDM26b]. Their tool covers 5-bit ($\deg \leq 4$), 6-bit ($\deg \leq 5$), 7-bit ($\deg = 3$), and 8-bit AES ($33/84, D = 4$) targets, but hits two structural walls [BDM26b, Sections 4.1.2, 4.2, 6, 7.2]: precomputation grows as $2^{\binom{n}{5}+\binom{n}{4}}$ while factorable quadratics drop from ≈ 0.50 to ≈ 0.15 at $n = 7$ (restricting $n = 7$ to $\deg \leq 3$), and 6-bit inversion costs 29/109 at $D = 3$ by division versus 21/63 by tower fields. For 8-bit inversion, Hadžić and Bloem [HB25] built the first depth-3 $\mathbb{F}_{2^8}$ inverter ($9 + 21 + 12 = 42$ AND) and a 139-gate AES S-box. While fixed-target division, EBPD, and SLICE lead in linear gates or combinational depth on several fixed classical targets, YOSHAN scales to $n = 7, \deg = 6$ and $n = 8, \deg = 7$ without division tables, cuts AND counts by up to 40% when the lookup table can vary across a profile, and reproduces the depth-3 AES architecture of [HB25] gate-exactly with its inverse (Table 4).

2.2 SAT Co-Synthesis, Metaheuristic Search, and Equivalence Classification

Jointly searching for an S-box and its circuit is much harder than minimizing a fixed table. At $n = 4$, Jia et al. [JCL+25] combined Prim-like upper bounds, balanced-Boolean lower bounds, and a unified SAT model of $(\delta, L, \deg)$, fixed points (FP), and UMC 180 nm area/depth to prove minimum areas in gate equivalents (GE) for optimal 4-bit S-boxes ($\delta = 4, L = 8$): 12 GE (8 gates, depth 4) with 2/3-input gates (12.33 GE at FP $= 0$), 11 GE with 4-input gates, and 14 GE at $\deg_{\min} = 3$, while also optimizing fixed 5-bit χ_5 (17.00 GE) and 8-bit SKINNY-128 (26.67 GE); without bound pruning, joint SAT fails even at $n = 4$ [JCL+25, Section 1]. At larger widths, structural Feistel/Misty templates [LW14, CDL16, BPU16] and 6-bit cipher designs (SC2000 [SYY+02], FIDES [BBK+13], SPEEDY [LMMR21], BIPBIP [BDD+23]) lock in a fixed network topology, whereas evolutionary searches [PČR16, KPF+24] and MAP-Elites [MC15] optimize truth-table non-linearity without tracking gate cost. YOSHAN instead uses the Boolean DAG itself as the genotype, jointly evolving $(\delta, L, \deg)$ and (AND, D, XOR) across $n \in \{5, 6, 7, 8\}$. Once the truth table is free to move, rigorous equivalence checking becomes essential: because exhaustive affine classification is known only for $n = 4$ (302 classes [LP07]) and quadratic $n \in \{5, 6\}$ (75 and 2,263 classes [BBŞ17, DMB19]), we pair EA/CCZ invariants [CCZ98, Kal21] with an exact affine-equivalence solver [BDCBP03] (Section 6.1) against a 712-entry catalogue spanning cyclotomic power maps, published classifications, Feistel/MISTY/Bridge structures, swapped power maps, and cipher S-boxes (KECCAK [BDPVA11], ASCON [DEMS21], SPEEDY [LMMR21], AES [Nat23], Dillon's APN [BDMW10]).

3 Algebraic Foundations and Cipher-Level Security Criteria

We focus on n-bit bijective S-boxes, modeled as permutations $S = (f_0, \ldots, f_{n-1}) \in \mathcal{S}_{2^n}$ over $\mathbb{F}_2^n$. This section reviews the cryptographic and algebraic properties that guide YOSHAN's search.

3.1 Differential Uniformity, Linearity, and Algebraic Degree

An S-box $S \in \mathcal{S}_{2^n}$ resists differential [BS91] and linear cryptanalysis [Mat94] through its Difference Distribution Table (DDT) for $\Delta x \in \mathbb{F}_2^n \setminus \{0\}$, $\Delta y \in \mathbb{F}_2^n$ and Walsh–Hadamard Transform for masks $\alpha \in \mathbb{F}_2^n$, $v \in \mathbb{F}_2^n \setminus \{0\}$ ($\langle u, v\rangle = \bigoplus_i u_i v_i$):

$$\mathrm{DDT}_S(\Delta x, \Delta y) = \#\{x \in \mathbb{F}_2^n : S(x) \oplus S(x \oplus \Delta x) = \Delta y\},\ W_S(\alpha, v) = \sum_x (-1)^{\langle \alpha, x\rangle \oplus \langle v, S(x)\rangle}. \tag{1}$$

The **differential uniformity** $\delta(S) = \max_{\Delta x \neq 0, \Delta y} \mathrm{DDT}_S(\Delta x, \Delta y)$ bounds single-S-box differential transitions by $DP_{\max} = \delta(S)/2^n$. Its optimal value $\delta(S) = 2$ defines **APN** functions [Nyb94]. Among bijections, APN permutations exist for odd $n \in \{5, 7\}$ via Gold (x^{2^k+1}) and Kasami ($x^{2^{2k}-2^k+1}$) power maps ($\gcd(k, n) = 1$) and at $n = 6$ via Dillon's CCZ class [BDMW10], while even $n \geq 8$ (the Big APN Problem) remains open. Similarly, the **linearity** $L(S) = \max_{\alpha, v \neq 0} |W_S(\alpha, v)| = 2 \max_{\alpha, v \neq 0} |\mathrm{LAT}_S(\alpha, v)|$ (for Linear Approximation Table LAT) sets maximum correlation $C_{\max} = L(S)/2^n$, linear potential $LP_{\max} = C_{\max}^2$, and non-linearity $N_L(S) = 2^{n-1} - L(S)/2$. For odd n, Almost Bent (AB) permutations meet the Sidelnikov–Chabaud–Vaudenay bound [CV95] $L(S) = 2^{(n+1)/2}$ ($L = 8$ at $n = 5$, $L = 16$ at $n = 7$). For even n, where bijections are never bent ($L(S) > 2^{n/2}$, so $N_L \leq 2^{n-1} - 2^{n/2-1} - 2$ [Car21]), the best known bijective values are $L = 16$ at $n = 6$ [BDMW10] and $L = 32$ ($N_L = 112$) at $n = 8$. Resistance to boomerang and rectangle attacks is governed by the Boomerang Connectivity Table $\mathrm{BCT}_S(a, b) = \#\{x : S^{-1}(S(x) \oplus b) \oplus S^{-1}(S(x \oplus a) \oplus b) = a\}$ and **boomerang uniformity** $\beta(S) = \max_{a, b \neq 0} \mathrm{BCT}_S(a, b)$ [CHP+18], where $\delta(S) \leq \beta(S) \leq 2^n$ and $\beta(S) = 2$ iff S is APN [BC18].

Every coordinate f_i has a unique Algebraic Normal Form (ANF) $f_i(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^{(i)} x^u$ ($x^u = \prod_j x_j^{u_j}$), computed in $O(n\, 2^n)$ via the binary Möbius transform. The **algebraic degree** $\deg(S) = \max_i \deg(f_i)$ satisfies $\deg(S) \leq n - 1$ ($n \geq 2$) and $\deg(S) \leq (n+1)/2$ for odd-n AB functions [CCZ98], while $\deg_{\min}(S) = \min_{v \neq 0} \deg\langle v, S(x)\rangle \geq 1$. To track degree across all output masks, the **component degree spectrum** $\mathcal{D}(S) = \{\deg\langle v, S(x)\rangle : v \neq 0\}$, written $[d_1^{m_1}, d_2^{m_2}, \dots]$, records all $2^n - 1$ non-zero component degrees. Because $V_{\leq d} = \{v : \deg\langle v, S(x)\rangle \leq d\}$ forms a nested subspace flag $0 = V_{\leq 0} \subset V_{\leq d_1} \subset \cdots \subset \mathbb{F}_2^n$ with $k_j = \dim V_{\leq d_j}$ and $m_j = 2^{k_j} - 2^{k_{j-1}}$, any two-level spectrum takes the subspace-partition form $\mathcal{D}(S) = [d_1^{2^k-1}, d_2^{2^n-2^k}]$ with $k = \dim V_{\leq d_1}$ (or $[d_2^{2^n-1}]$ when $d_1 = d_2$).

3.2 Algebraic Degree, Higher-Order Derivatives, and Masked Total Complexity

While Wide-Trail bounds [DR01] control single-trail $(DP_{\max})^{A_r}$ and $(LP_{\max})^{A_r}$, resistance to boomerang [CHP+18], integral and division-property [Tod15], zero-sum [BCDC11], cube [DS09], and interpolation attacks [JK97] hinges on how fast algebraic degree grows across rounds. Following Lai [Lai94] and Knudsen [Knu95], the r-th order derivative along linearly independent differences $a_1, \dots, a_r$ sums S over an r-dimensional cube and lowers the degree by at least r:

$$D_{a_1} \cdots D_{a_r} S(x) = \bigoplus_{I \subseteq \{1, \dots, r\}} S\left(x \oplus \bigoplus_{i \in I} a_i\right), \quad r > \deg(S) \implies D_{a_1} \cdots D_{a_r} S(x) \equiv 0. \tag{2}$$

When $\deg(S) = 2$, third derivatives vanish after one round and r-round degree grows at most as 2^r [BCDC11], forcing high round counts (Ascon [DEMS21], Keccak-f [BDPVA11]) compared with degree-5 designs (SPEEDY [LMMR21]). In hardware, multiplicative complexity and depth obey $MC(S) \geq \deg(S) - 1$ and $MD(S) \geq \lceil \log_2 \deg(S) \rceil$ [BFP16, GR16]. Realizing degree-3 to 7 S-boxes at $D \in \{2, 3, 4\}$ with near-quadratic AND counts therefore yields faster degree saturation, fewer cipher rounds, and lower total masked latency ($R_{\text{rounds}} \times D$).

3.3 Tower Fields, Affine Equivalence, and Forward/Inverse Duality

Composite tower fields. Evaluating $x \mapsto x^{2^n-2}$ directly over $\mathbb{F}_{2^n}$ yields deep circuits. By contrast, decomposing $\mathbb{F}_{2^8} \cong \mathbb{F}_{((2^2)^2)^2}$ via $M \in \mathrm{GL}(8, \mathbb{F}_2)$ reduces $\mathbb{F}_{2^8}$ inversion to a norm $\Delta \in \mathbb{F}_{2^4}$, one $\mathbb{F}_{2^4}$ inversion, and two $\mathbb{F}_{2^4}$ products in the polynomial basis of Satoh

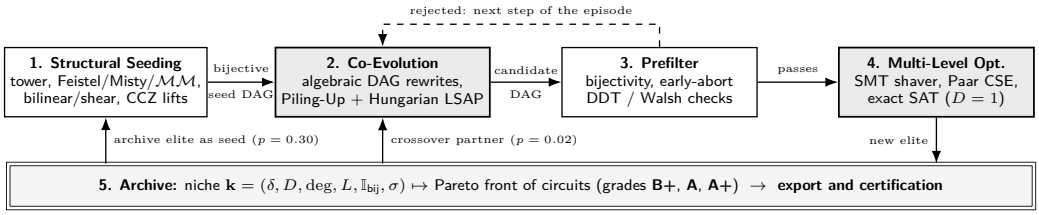


Figure 1: One search episode of YOSHAN. Stages 2–4 repeat until the episode stops; the archive is shared by all workers. σ is a signature of the DDT and |LAT| value histograms.

et al. [SMTM01] or Canright's normal basis [Can05, BP10]. Recursing to $\mathbb{F}_{2^2}$ (where inversion is a free wire swap and multiplication takes 3 AND gates at $D = 1$) gives the compact AES circuits of [Can05, BP10, BP12, BDM26b, HB25]; similarly, $\mathbb{F}_{2^6} \cong \mathbb{F}_{(2^3)^2}$ yields the 21-AND, 63-XOR ($D = 3$) 6-bit inverter of [BDM26b, Table 5].

Affine equivalence and forward/inverse duality. Two S-boxes are **affine equivalent** ($S_1 \sim_{\text{aff}} S_2$) if $S_2(x) = B \cdot S_1(Ax \oplus c) \oplus d$ for $A, B \in \mathrm{GL}(n, \mathbb{F}_2)$ and $c, d \in \mathbb{F}_2^n$. While δ, L, deg, $\mathcal{D}(S)$, $MC(S)$, and $MD(S)$ are affine invariants on $\mathcal{S}_{2^n}$ (and shared across each cyclotomic coset $C_e = \{e \cdot 2^k \bmod (2^n - 1)\}$), fixed points $\mathrm{FP}(S)$, the Strict Avalanche Criterion (SAC), and XOR counts vary across representatives. For decryption, involutions and balanced Feistel networks reuse or reverse the forward core at identical (AND, D), whereas quadratic S-boxes have $\deg(S^{-1}) \in \{3, 4\}$ and require $D^{-1} \geq 2$ (Section 6.3). Because $(\delta, L, \deg)$ depend on the permutation S while masked area, randomness (N_{AND}), and registers ($4N_{\text{AND}} + 2N_{\text{sync}}$, Section 5.2) depend on the circuit DAG G, YOSHAN co-optimizes S and G together.

4 The Yoshan Search Engine

Instead of picking a truth table for $(\delta(S), L(S))$ and then minimizing its gates, YOSHAN searches directly in circuit space so that AND depth D and AND count (Section 5.2) are optimized alongside cryptographic strength. Each candidate is a DAG G of 2-input AND, 2-input XOR, and NOT gates (plus small lookup-table (LUT) nodes during search) computing $S = \mathrm{LUT}(G)$. Parallel workers share elites through a common archive across five stages (Figure 1): (1) **seeding** (Section 4.1) from a bijective circuit; (2) **circuit evolution** (Section 4.2) under annealed cryptographic and cost rules; (3) a fast **prefilter and grade gate** (Section 4.3); (4) **compression** (Section 4.3) via sub-DAG shaving, CSE, and SAT; and (5) the **quality-diversity archive** (Section 4.4), followed by gate-exact export and formal certification (Sections 5 and 7).

4.1 Stage 1: Seeding

Every episode starts from a bijective circuit, since random lookup tables are rarely bijective and random gate graphs are deep. Finite-field seeds use $\mathbb{F}_{2^n} \cong \mathbb{F}_2[x]/(p_n(x))$ with $p_5 = x^5 + x^2 + 1$, $p_6 = x^6 + x + 1$, $p_7 = x^7 + x + 1$, and $p_8 = x^8 + x^4 + x^3 + x + 1$ (changing the polynomial basis is a linear isomorphism that preserves affine classes). Each worker draws its starting circuit from one of three sources. First, once the archive is non-empty, it picks an **archive elite** with probability 0.30. Otherwise, with conditional probability ρ_g (0.7 in optimal, 0.6 in relaxed/lightweight campaigns), it picks a **published or algebraic circuit**: the cipher S-boxes of Table 6, Gold and inverse-Gold circuits, DB_1 [AL26], Q_{2256}/Q_{2258} [BDM26a] and their inverses, fractional subfield towers, $\mathbb{F}_{(2^3)^2}$ inversion (21 AND, $D = 3$), and for $n = 8$ Camellia, Gold, $\mathbb{F}_{((2^2)^2)^2}$ AES (36 AND, $D = 4$ [Can05]), and

the 42-AND, $D = 3$ inverter of Hadžić and Bloem [HB25]. With the remaining probability $1 - \rho_g$, it builds a **structural seed**. These include **Feistel**, **Misty**, and **product-round Feistel ($\mathcal{MM}$-type)** networks $S(x_L, x_R) = (x_R, x_L \oplus \pi(x_R) \odot \phi(x_R))$ (3-round balanced Feistel seeds carry a structural probability-one boomerang, Section 6.6), sparse affine wrappers $S(x) = A_1 R(A_2 x \oplus c_1) \oplus c_2$ with A_1, A_2 formed by $\lfloor n/2 \rfloor$ to $2n$ row additions from I_n, and **linear/CCZ graph automorphisms** $\mathcal{L}\binom{x}{F(x)} = \left(\begin{smallmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{smallmatrix}\right)\binom{x}{F(x)}$ with $\mathcal{L} \in \mathrm{GL}(2n, \mathbb{F}_2)$. Whenever $\pi_1(x) = L_{11}x \oplus L_{12}F(x)$ and $\pi_2(x) = L_{21}x \oplus L_{22}F(x)$ are both bijective, this graph transform yields the CCZ-equivalent permutation $S_{\text{lift}} = \pi_2 \circ \pi_1^{-1}$ (also used as a Stage 2 mutation). Structural seeds also include the differentiable templates of Section 4.2, single lookup-table nodes holding a power map (mutated by LUT swaps until lowered to gates, Section 5.1), and, at $n = 5$, a 5-AND, $D = 1$ quadratic template $y_k = M_k x \oplus (\alpha_k \cdot x \oplus a_k)(\beta_k \cdot x \oplus b_k) \oplus c_k$ sampled via bounded-random SAT. Two additional seed techniques target quadratic cores directly:

Bilinear and shear seeds. For a quadratic target, a randomized bilinear solver builds $D = 1$ seeds by extracting $c_k = S_k(0)$, $L_{k,i} = S_k(e_i) \oplus c_k$, and $Q_{k,i,j} = S_k(e_i \oplus e_j) \oplus c_k \oplus L_{k,i} \oplus L_{k,j}$ $(i < j)$ via finite differences and sampling R products of linear forms $a_r(x) = (u_r^\top x) \wedge (v_r^\top x)$:

$$a_r(x) = \bigoplus_{0 \le i < j < n} (u_{r,i} v_{r,j} \oplus v_{r,i} u_{r,j})\, x_i x_j \oplus \bigoplus_{j=0}^{n-1} (u_{r,j} \wedge v_{r,j})\, x_j, \tag{3}$$

$$L_{k,j}^{\text{req}} = L_{k,j} \oplus \bigoplus_{r=1}^{R} W_{k,r}(u_{r,j} \wedge v_{r,j}), \qquad S_k(x) = c_k \oplus \bigoplus_{j=0}^{n-1} L_{k,j}^{\text{req}} x_j \oplus \bigoplus_{r=1}^{R} W_{k,r}\, a_r(x). \tag{4}$$

Whenever each Q_k lies in the $\mathbb{F}_2$-span of $(u_{r,i} v_{r,j} \oplus v_{r,i} u_{r,j})_{i<j}$, Gaussian elimination yields $W_{k,r}$ and $L_{k,j}^{\text{req}}$ cancels the diagonal $x_j^2 = x_j$ terms, producing a $D = 1$ circuit with R AND gates (e.g., $R = 7$ for 5-bit Gold x^5). Complementing this, a *shear* sets $y_{n-1} \leftarrow y_{n-1} \oplus u(y) \wedge v(y)$ with linear forms u, v independent of y_{n-1}. This preserves bijectivity while adding one AND gate and raising a quadratic core up to degree 4 (Section 6.6).

4.2 Stage 2: Circuit Evolution

During an episode, a worker evolves G over many steps (Algorithm 1) using three kinds of moves. First, **crossover** (prob. 0.02) pairs G with an elite from another niche, blending template weights via homotopy when both parents are template-based or otherwise sharing inputs and combining outputs via XOR or AND (Stage 3 drops non-bijective children). Second, **function-preserving rewrites** apply CSE, distributive folding $(u_1 \wedge v) \oplus (u_2 \wedge v) \to (u_1 \oplus u_2) \wedge v$, and (every 25 steps) AND-tree rotation and factoring (Figure 2). Third, **function-changing mutations** reshape the permutation itself: before (δ, L) targets are met, workers apply LUT swaps and Feistel, Misty, or Lai–Massey moves; once targets are met, they apply an *AND eraser* (replacing one or two AND gates by XORs), *depth collapse* (rewiring a deepest AND input to a wire ≥ 2 levels shallower), and affine or CCZ graph automorphisms. A proposed G' is kept only if it is acyclic, bijective, and (for erasers and collapses) preserves the (δ, L) targets. Candidates are ranked in two phases: before (δ, L) targets are met, by target distance, maximal DDT count, deg, and SAC deviation; once targets are met, by higher deg and lower hierarchical masked-cost score $10^5\, N_{\text{AND}} + 1.5 \cdot 10^4\, D + 100\, N_{\text{XOR}} + 10\, N_{\text{NOT}}$. The decade weights enforce $(N_{\text{AND}}, D) \succ N_{\text{XOR}} \succ N_{\text{NOT}}$ so non-linear gates and stages never trade for linear gates. Acceptance follows cosine simulated annealing (period 2000, $T \in [0.05, 5]$), restarting from an elite after 50 duplicates and stopping after $\max(P, 1500(n-3))$ non-improving steps. LUT nodes carry zero gate cost during search and appear only before (δ, L) targets are met;

Algorithm 1: One YOSHAN search episode (per worker)

Input: width n, targets (δ_t, L_t), shared archive $\mathcal{A}$ **Ensure:** new elites inserted into $\mathcal{A}$

1: $G \leftarrow$ seed from $\mathcal{A}$ ($p = 0.30$); otherwise from the circuit library ($p = \rho_g$) or a structural construction ($p = 1 - \rho_g$) (Section 4.1)

2: **repeat**

3: $G' \leftarrow$ crossover with an elite ($p = 0.02$); otherwise a function-preserving rewrite if one applies (CSE or distributive folding; AND-tree rotation and factoring every 25th step); otherwise a function-changing mutation of G

4: **if** G' is cyclic, or fails the prefilter, or LUT(G') is not a bijection **then continue**

5: **end if**

6: compute $\delta, L, \deg$, SAC deviation, and gate counts of G'; **if** G' is better than G or accepted by annealing **then** $G \leftarrow G'$

7: **if** G has grade $\geq$ **B+** and is non-dominated in its niche **then** compress G (CSE) and insert into $\mathcal{A}$

8: **end if**

9: **until** targets met and no improvement for $\max(P, 1500(n-3))$ steps

10: final compression of G (sub-DAG shaver, CSE, and exact $D = 1$ SAT for quadratic $n \leq 7$); insert G into $\mathcal{A}$ if non-dominated

every candidate is lowered to logic gates before compression, certification, and archiving (Section 5.1).

Differentiable Piling-Up relaxation and Hungarian LSAP projection. As an exploratory gradient-guided module, YOSHAN parameterizes multi-round networks with learned wire permutations: a Keccak-like Substitution-Permutation Network (SPN) and an unbalanced Feistel network at $n = 7$, and dual-branch Misty networks for $n \in \{5, 6, 7, 8\}$. Wires are relaxed to probabilities $p_i \in [0, 1]$, AND gates to $p_i p_j$, and k-input XOR sums to Matsui's multilinear Piling-Up relaxation $\widetilde{\mathrm{XOR}}(p_1, \ldots, p_k) = \frac{1}{2} - \frac{1}{2}\prod_{m=1}^{k}(1 - 2p_m)$ [Mat94], which is exact on $\{0, 1\}^k$ and under independent inputs. Multiplying by the Hadamard matrix H_{2^n} gives the relaxed Walsh spectrum $\widetilde{W}_S(\alpha, v) = H_{2^n}\big(1 - 2\widetilde{S}_v(x)\big)$ for every component on all 2^n inputs. We then minimize the multi-objective Adam loss

$$\mathcal{L}(\Theta; \tau) = 20\Big(\sum_{\alpha \neq 0, v \neq 0} |\widetilde{W}_S(\alpha, v)|^{32}\Big)^{1/32} + 10\,\mathcal{L}_{\mathrm{SAC}} + 15\,\mathcal{L}_{\mathrm{sym}} + \lambda(\tau)\big(\mathcal{L}_{\mathrm{ent}} + \mathcal{L}_{\ell_1}\big), \quad (5)$$

which jointly penalizes peak Walsh magnitude, SAC deviation, wire-permutation fixed points, and (with annealed weight $\lambda(\tau)$) non-binary wires and dense polynomials. After gradient descent, the Hungarian algorithm [Kuh55] projects the continuous weights to a discrete circuit using one LSAP per round permutation and K masked rectangular LSAP passes per branch polynomial. Bijectivity holds structurally for the Feistel/Misty branch update $L \leftarrow L \oplus F(R)$ and is checked by Stage 3 for SPN projections. This module runs as a Stage-1 structural seeder (prob. 0.18, 0.18–0.75 s per seed across $n \in \{5, \ldots, 8\}$) and as the homotopy crossover of Algorithm 1. SM A.2 details its terms, schedule, and projection; Section 6.5 quantifies its archive yield (26 direct-graduate netlists, zero benchmark circuits), and Section 8 analyzes its depth attenuation.

4.3 Stages 3 and 4: Prefilter, Grade Gate, and Circuit Compression

Every candidate first passes a bijectivity check and early-abort DDT/Walsh scans against a cap above the target. The archive accepts only circuits with $\deg \geq 2$ that meet a grade threshold (**A+**, **A**, or **B+** in Table 1). Beyond the Stage-2 rewrites (Figure 2), accepted circuits are compressed without changing S by three passes. First, **Paar CSE and Boyar–Peralta wire reuse** [Paa97, BP10] merge identical gates, factor shared XOR operand pairs, and synthesize intra-stage linear blocks via distance-1/2 wire reuse. Second, the

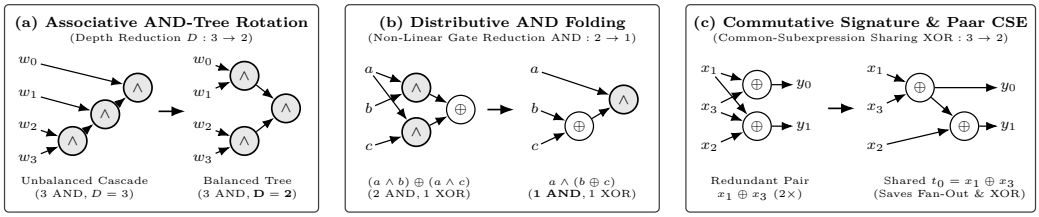


Figure 2: Examples of the circuit rewrites used in Stages 2 and 4 (Sections 4.2 and 4.3): associative AND-tree rotation, distributive folding, and common-subexpression elimination. Shaded nodes ($\wedge$) are AND gates, white nodes ($\oplus$) are XOR gates.

Table 1: Grade thresholds of the archive gate (all grades also require $\deg \geq 2$).

n	**A+**	**A**	**B+**
5	$\delta \leq 2, L \leq 8$	$\delta \leq 2, L \leq 12$; or $\delta \leq 4, L \leq 8$; or $\delta \leq 4, L \leq 12, \deg \geq 3$	$\delta \leq 8, L \leq 16$
6	$\delta \leq 2, L \leq 16$	$\delta \leq 4, L \leq 24$	$\delta \leq 6, L \leq 24$; or $\delta \leq 8, L \leq 28$
7	$\delta \leq 2, L \leq 16$	$\delta \leq 4, L \leq 32$	$\delta \leq 6, L \leq 40$; or $\delta \leq 8, L \leq 44$
8	$\delta \leq 4, L \leq 32$	$\delta \leq 8, L \leq 48$	$\delta \leq 8, L \leq 64$; or $\delta \leq 16, L \leq 80$

sub-DAG shaver checks whether two AND gates (g_1, g_2) supported on a cut C ($|C| \leq 6$) can be replaced by a single product $a(z) = (u^\top z) \wedge (v^\top z)$ plus affine functions of $z \in \mathbb{F}_2^{|C|}$, meaning some $w_k \in \{0, 1\}$ makes $g_k \oplus w_k a$ affine on C for $k \in \{1, 2\}$. This condition is decided either exactly by Z3 [dMB08] over all $2^{|C|}$ cut assignments or by a fast filter testing whether some $f \in \{g_k, g_k \oplus a\}$ satisfies $f(e_{b_1} \oplus e_{b_2}) \oplus f(e_{b_1}) \oplus f(e_{b_2}) \oplus f(0) = 0$ for all $b_1 < b_2$ (complete when $\deg_z g_k \leq 2$); every rewrite is re-simulated on all 2^n inputs and reverted if $\mathrm{LUT}(G)$ changes, keeping the shaver sound regardless of the filter. Third, **exact SAT re-synthesis** searches for a $D = 1$ circuit with $\leq 8, 13, 14$ AND gates on quadratic results for $n = 5, 6, 7$ (under a per-call timeout). Running the sub-DAG shaver and stage-preserving Paar CSE once more before printing forms the *Stage-4 pass* of Section 6.5.

4.4 Stage 5: Quality-Diversity Archive

Without diversity maintenance, cheap quadratic circuits ($\deg = 2$, $D = 1$) would crowd out higher-degree candidates before compression can shrink them. Following MAP-Elites [MC15], each candidate competes only inside its own niche

$$\mathbf{k}(S) = \Big(\delta(S),\ D(G),\ \deg(S),\ L(S),\ \mathbb{I}_{\mathrm{bij}}(S),\ \sigma(S)\Big), \tag{6}$$

where $D(G)$ is circuit depth, $\mathbb{I}_{\mathrm{bij}}(S) = 1$ for archived circuits (Stage 3 rejects non-bijections), and $\sigma(S)$ hashes the DDT and |LAT| histograms to keep inequivalent S-boxes with equal $(\delta, L, \deg)$ in separate cells. A grade $\geq$ **B+** candidate enters $\mathcal{A}[\mathbf{k}]$ if it is non-dominated in $(N_{\mathrm{AND}}, D, N_{\mathrm{AND}} + N_{\mathrm{XOR}}, c)$ with $c = 1000\, D + 0.8\, N_{\mathrm{AND}} + 1.6\, N_{\mathrm{XOR}} + 0.5\, N_{\mathrm{NOT}}$. Because D is fixed inside each niche, dominance reduces to $(N_{\mathrm{AND}}, N_{\mathrm{AND}} + N_{\mathrm{XOR}}, 0.8\, N_{\mathrm{AND}} + 1.6\, N_{\mathrm{XOR}} + 0.5\, N_{\mathrm{NOT}})$, weighting gates by their unmasked CMOS area ratios $1.064 : 1.596 : 0.532\,\mu\mathrm{m}^2$ (Eq. (7)). Dominated entries are pruned to at most eight circuits per niche (Figure 4 in SM A).

5 Netlist Export and Gate-Exact Hardware Synthesis

Standard logic optimizers such as Berkeley ABC [BM10] and Yosys [WG13] operate on And-Inverter Graphs (AIGs), expanding each 2-input XOR into three AND gates and inverters. Their algebraic rewrites (`strash`, `rewrite`, `refactor`) mix linear and non-linear

cones and trade XOR gates for AND gates or compound cells (`AOI21`, `OAI21`). Under Boolean masking, however, every non-linear gate costs cross-share products, registers, and fresh randomness while linear gates act share-wise. YOSHAN therefore never applies AIG restructuring: it emits each DAG gate for gate and maps every operator 1-to-1 to its standard cell.

5.1 Gate-for-Gate Export and Technology Mapping

Each archived DAG is emitted as structural SystemVerilog with one continuous assignment per AND, XOR, and NOT gate. Any remaining lookup-table node is expanded first via sub-circuit synthesis or ANF, mapping each degree-d monomial to a balanced AND tree of depth $\lceil \log_2 d \rceil$ with shared sub-products and XOR trees. Before printing in SM C, identical gates are merged, dead gates are pruned, and the netlist is exhaustively re-verified; all reported gate counts refer to these printed netlists.

To measure area and timing without altering a single gate, we synthesize all 60 forward and inverse netlists in Yosys against the NanGate 45 nm Open Cell Library (`Nangate45_typ.lib`) [Nan11] via 1-to-1 mapping. Specifically, `simplemap` lowers operators to primitives, `techmap` and `dfflibmap` bind each primitive to its unit-drive cell (`AND2_X1`, `XOR2_X1`, `INV_X1`, `DFF_X1`) without ABC's AIG converter, `stat -liberty` reports cell area in μm^2 and gate equivalents (1 GE = Area(`NAND2_X1`) = 0.798 μm^2), and `ltp` extracts the longest topological path. Berkeley ABC (`read_verilog -m; topo; stime`) then computes the Non-Linear Delay Model (NLDM) critical-path delay in picoseconds, capturing slew and fan-out load on the untouched graph.

5.2 Unmasked and Pipelined First-Order Masked Silicon Synthesis

Under 1-to-1 NanGate 45 nm mapping, the unmasked combinational cell area for N_{AND}, N_{XOR}, and N_{NOT} gates is

$$\text{Area}_{\text{unmasked}}\ (\mu\text{m}^2) = 1.064 \cdot N_{\text{AND}} + 1.596 \cdot N_{\text{XOR}} + 0.532 \cdot N_{\text{NOT}} \tag{7}$$

(1.333, 2.0, and 0.667 GE per cell). For first-order Boolean masking across all 60 circuits, we compile each depth-D DAG into a pipelined D-stage, two-share ($t = 1, d = 2$) `DOM-indep` [GMK16] SystemVerilog module (Figure 3 in SM A) and synthesize it 1-to-1 in Yosys:

1. **Stage assignment and synchronization.** Each wire w gets its as-soon-as-possible (ASAP) stage $\ell(w) \in \{0, \ldots, D\}$ ($\ell = 0$ for inputs, $\max(\ell(a), \ell(b))$ for linear gates, and $\max(\ell(a), \ell(b)) + 1$ for AND gates). A wire from stage ℓ_0 feeding $\ell_1 > \ell_0$ (or an output at D) traverses a deduplicated chain of $\ell_1 - \ell_0$ flip-flop pairs (2 `DFF_X1` per stage crossed, 4.522 μm^2 per `DFF_X1`). Because XOR trees compress wires before crossing stages, ASAP uses no more sync flip-flops than as-late-as-possible (ALAP) on 23 of the 30 forward circuits (138 vs. 144 on $D = 4$ AES; on $D = 3$ AES ALAP uses 42 vs. 48).
2. **Share-domain and `DOM-indep` gadgets.** Each 2-input XOR is duplicated across the two share domains (2 `XOR2_X1`), and each NOT inverts share 0 only (1 `INV_X1`). Each stage-ℓ AND gate becomes a `DOM-indep` gadget with two inner-domain and two cross-domain products (4 `AND2_X1`), two resharing XORs with one fresh random bit (2 `XOR2_X1`), four stage-ℓ registers (4 `DFF_X1`), and two domain-compression XORs (2 `XOR2_X1`).

A compiled $d = 2$ `DOM-indep` module with N_{sync} synchronized wire-stage crossings thus instantiates

$$N_{\text{AND2}} = 4N_{\text{AND}},\ N_{\text{XOR2}} = 2N_{\text{XOR}} + 4N_{\text{AND}},\ N_{\text{INV}} = N_{\text{NOT}},\ N_{\text{DFF}} = 4N_{\text{AND}} + 2N_{\text{sync}}, \tag{8}$$

uses N_{AND} fresh random bits per evaluation, and achieves throughput 1/cycle at latency D. Here `DOM-indep` serves as our architectural register and area baseline: at $D \geq 2$, its inner-domain products $(a_0 \wedge b_0, a_1 \wedge b_1)$ are unregistered before compression and lack multi-stage Probe-Isolating Non-Interference (PINI) composability [CGLS21, MMSS19]. Tables 3 and 9 thus report intra-stage fan-in reconvergence N_{reconv} (Section 7.3), while SM A.1 gives exact $d = 2$ cell/register costs and AES comparisons under `DOM-dep` [GMK16], two-cycle PINI `HPC2` [CGLS21], and single-cycle `HPC3`/`HPC3.1` [KM22, HB25]. Every `DOM-indep` netlist is verified on all 2^n inputs ($y_0 \oplus y_1 = S(x_0 \oplus x_1)$ and $x_0 \oplus x_1 = S^{-1}(y_0 \oplus y_1)$), and all 60 unmasked and 60 `DOM-indep` post-synthesis netlists are proven equivalent to their SystemVerilog modules in Yosys with cell counts matching Eqs. (7)–(8).

6 Results

We ran eight search campaigns across $n \in \{5, 6, 7, 8\}$ (Table 5 in SM B), pairing one near-optimal campaign with one relaxed/lightweight campaign at each width. Table 1 sets every archived circuit's grade regardless of which campaign found it. In total, the archive holds 31,337 certified netlists spanning 30,986 distinct permutations across 56 $(n, \delta, L, \deg)$ profiles (Tables 7 and 11), all verified by exhaustive simulation and the Z3 check of Section 7. At $n = 7$, where optimal runs start from APN power maps, 865 of 895 distinct permutations lie in APN power-map classes, while the lightweight campaign adds compact quartic and degree-6 permutations with $\delta = 4$. At $n = 6$ and $n = 8$, the relaxed campaigns fill large **B+** niches where high degree meets low gate count: at $n = 6$, 19,380 netlists sit in $(\delta, L, \deg) = (6, 24, 5)$ (down to 12 AND gates at $D = 4$), with 1,729 and 1,611 more in $(8, 24, 5)$ and $(8, 28, 5)$ (down to 12 and 11 AND gates). At $n = 8$, 3,925 netlists populate $\delta \in \{10, 12\}$, $L \in \{64, 68, 72\}$, where direct ANF synthesis needs 239–246 AND gates compared with 24–28 for the $\mathsf{YSH}_{8,2}$ Feistel family. Hardware-efficient candidates lie along the 104 non-dominated $(\text{AND}, \text{XOR}, D)$ Pareto points of Table 11. Table 2 summarizes the 30 benchmark S-boxes printed with their inverses in SM C, and Table 10 traces each from seed to final netlist (Section 6.5).

6.1 Equivalence Classification

Because co-synthesis modifies the truth table, we test whether each archived permutation S belongs to a known class using four lookup-table invariants: the forward and inverse degree spectra $\mathcal{D}(S), \mathcal{D}(S^{-1})$ (Section 3.1), the differential spectrum (multiset of $\text{DDT}(a, b)$, $a \neq 0$), and the extended Walsh spectrum (multiset of $|W_S(\alpha, v)|$, $v \neq 0$). The last two are invariant under EA ($S' = A_2 \circ S \circ A_1 \oplus A_3$) and CCZ equivalence [CCZ98, Car21], while $\mathcal{D}(S)$ is affine-invariant on $\mathcal{S}_{2^n}$ and EA-invariant whenever $\deg_{\min}(S) \geq 2$. We screen every archived permutation against a reference catalogue of 712 unique verified lookup tables across $n \in \{5, 6, 7, 8\}$ (Table 6). The catalogue spans six families: (i) all bijective cyclotomic power maps x^d with $\gcd(d, 2^n - 1) = 1$; (ii) all 75 quadratic 5-bit classes $Q_1, \ldots, Q_{75}$ of Božilov, Bilgin, and Şahin [BBŞ17] and their inverses; (iii) the 17 strong quartic 5-bit classes (Table 7) and the 8 optimal $(\delta, L) = (4, 16)$ 6-bit quadratic classes $Q_{2256}, \ldots, Q_{2263}$ (Table 15) of De Meyer and Bilgin [DMB19] and their inverses; (iv) 3-round Feistel, MISTY, Bridge, Open Butterfly, and unbalanced $\text{MISTY}_{5,3,5}$ maps [CDL16, BPU16, BDMD+20], including the 6-bit 3-round χ_3-Feistel cubic involution $\text{Feistel}_{3\text{R}}(\chi_3)$; (v) all canonical Frobenius and inversion linear-equivalence orbits of two-point-swapped power maps [BCC10, BKRS22] across $n \in \{5, 6, 7, 8\}$ and their inverses; and (vi) deployed and literature S-boxes (Keccak χ_5, Ascon, Fides-5/6 [BBK+13], SC2000-5/6 [SYY+02], DryGASCON128, Shamash, SPEEDY, Dillon's APN, BipBip-6 [BDD+23], AES, and Skinny-8) and their inverses.

Matching invariants alone do not prove affine equivalence, so an exact solver following Biryukov et al. [BDCBP03] (combining bidirectional linear-equivalence closure between

Table 2: Cryptographic properties and gate counts of the 30 forward (S) and inverse (S^{-1}) circuits (SM C). δ: differential uniformity; L: linearity (NL $= 2^{n-1} - L/2$); β: boomerang uniformity (†Feistel-type with $\beta = 2^n$: 3-round Feistel circuits and 2-round unbalanced $\mathsf{YSH}_{5,2}$, Section 6.6); d/d^{-1}: degrees of S, S^{-1}; $\mathcal{D}(S)$: degree spectrum; FP: fixed points; SAC: max avalanche bias $|\Pr_x[S_j(x) \neq S_j(x \oplus e_i)] - \frac{1}{2}|$; A/X/N, D: AND/XOR/NOT counts and AND depth. Cat. A: equal (=) or affine-equivalent ($\sim_{\text{aff}}$) to Table 6 (712 entries); Cat. B ($\mathsf{YSH}_{n,i}$): EA-separated from Table 6. *3-round χ_3-Feistel involution ($S^{-1} = S$, $\mathcal{D}(S^{-1}) = [3^{63}]$; EA-inequivalent to x^{13}).

									Forward (S)		**Inverse (S^{-1})**		
S-box (SM)	n	δ	L	β	d/d^{-1}	$\mathcal{D}(S)$	FP	SAC	A/X/N	D	A/X/N	D^{-1}	Cat. / Class
$x^3_{\text{swap}(1,3)}$ class (C.1)	5	4	12	6	4/4	$[2^{15}, 4^{16}]$	0	.125	11/20/3	2	13/38/6	2	**A** ($\sim_{\text{aff}} x^3_{\text{swap}(1,3)}$)
Q_{71} class (C.2)	5	4	16	12	2/3	$[2^{31}]$	0	.5	5/12/3	1	9/32/2	2	**A** ($\sim_{\text{aff}} Q_{71}$)
$\mathsf{YSH}_{5,1}$: cubic, δ=8 (C.3)	5	8	16	20	3/3	$[2^{15}, 3^{16}]$	2	.5	6/11/0	2	9/22/2	2	**B** ($\mathsf{YSH}_{5,1}$)
$\mathsf{YSH}_{5,2}$: quartic, δ=8 (C.4)	5	8	16	$32^\dagger$	4/4	$[2^7, 3^8, 4^{16}]$	2	.5	5/12/1	2	5/11/2	2	**B** ($\mathsf{YSH}_{5,2}$)
Gold x^5 (C.5)	5	2	8	2	2/3	$[2^{31}]$	2	.5	7/22/0	1	10/37/4	2	**A** ($= x^5$)
Gold class, SAC-ideal (C.6)	5	2	8	2	2/3	$[2^{31}]$	1	0	7/21/4	1	10/41/10	2	**A** ($\sim_{\text{aff}} x^5$)
χ_5 class (C.7)	5	8	16	16	2/3	$[2^{31}]$	5	.5	5/8/1	1	9/21/3	2	**A** ($\sim_{\text{aff}} \chi_5$)
inverse-Gold (C.8)	5	2	8	2	3/2	$[3^{31}]$	1	.125	14/31/2	2	7/26/5	1	**A** ($\sim_{\text{aff}} x^{25}$)
x^{15} (C.9)	5	2	12	2	4/4	$[4^{31}]$	2	.125	14/45/0	2	14/57/0	2	**A** ($= x^{15}$)
$\mathsf{YSH}_{6,1}$: quartic, δ=4 (C.10)	6	4	16	$64^\dagger$	4/4	$[3^7, 4^{56}]$	0	.125	9/27/1	3	9/27/1	3	**B** ($\mathsf{YSH}_{6,1}$)
$\mathsf{YSH}_{6,2}$: quartic, k=5 (C.11)	6	4	16	12	4/3	$[3^{31}, 4^{32}]$	2	.25	27/64/0	3	9/37/0	2	**B** ($\mathsf{YSH}_{6,2}$)
$\mathsf{YSH}_{6,3}$: cubic, δ=4 (C.12)	6	4	16	$64^\dagger$	3/4	$[3^{63}]$	2	.5	9/17/9	3	9/17/8	3	**B** ($\mathsf{YSH}_{6,3}$)
$\mathsf{YSH}_{6,4}$: deg. 5, δ=8 (C.13)	6	8	28	18	5/5	$[2^{31}, 5^{32}]$	2	.5	11/22/1	3	22/49/1	3	**B** ($\mathsf{YSH}_{6,4}$)
$\mathsf{YSH}_{6,5}$: deg. 5, δ=6 (C.14)	6	6	24	14	5/5	$[2^{31}, 5^{32}]$	1	.125	12/31/0	4	21/57/0	3	**B** ($\mathsf{YSH}_{6,5}$)
$\mathsf{YSH}_{6,6}$: quartic, δ=8 (C.15)	6	8	24	16	4/4	$[2^{31}, 4^{32}]$	2	.5	10/22/0	2	27/50/3	2	**B** ($\mathsf{YSH}_{6,6}$)
Q_{2258} class (C.16)	6	4	16	4	2/3	$[2^{63}]$	2	.5	8/34/0	1	18/37/0	2	**A** ($\sim_{\text{aff}} x^5$)
χ_3-Feistel invol. (C.17)	6	4	16	$64^\dagger$	3/3	$[3^{63}]$	8	.5	9/15/9	3	9/15/9	3	**A** ($\sim_{\text{aff}} \text{Feistel}_{3\text{R}}(\chi_3)$)*
inversion (C.18)	6	4	16	4	5/5	$[5^{63}]$	1	.125	21/48/0	3	21/50/0	3	**A** ($\sim_{\text{aff}} x^{62}$)
Dillon APN (C.19)	6	2	16	2	4/4	$[3^7, 4^{56}]$	1	.5	28/68/0	2	28/78/7	2	**A** (= Dillon)
$\mathsf{YSH}_{7,1}$: quartic, δ=4 (C.20)	7	4	32	12	4/5	$[2^{63}, 4^{64}]$	1	.125	15/70/10	2	30/158/12	3	**B** ($\mathsf{YSH}_{7,1}$)
$x^5_{\text{swap}(1,25)}$ class (C.21)	7	4	20	10	6/6	$[2^{63}, 6^{64}]$	2	.0312	51/76/1	3	34/164/9	3	**A** ($\sim_{\text{aff}} x^5_{\text{swap}(1,25)}$)
Gold x^3 (C.22)	7	2	16	2	2/4	$[2^{127}]$	2	.5	14/57/0	1	29/149/15	2	**A** ($= x^3$)
Kasami x^{13} (C.23)	7	2	16	2	3/3	$[3^{127}]$	2	.0625	30/114/0	2	37/139/0	2	**A** ($= x^{13}$)
inversion (C.24)	7	2	20	2	6/6	$[6^{127}]$	0	.0938	42/182/6	3	42/229/6	3	**A** ($\sim_{\text{aff}} x^{126}$)
power map x^7 (C.25)	7	6	40	10	3/5	$[3^{127}]$	2	.125	56/98/0	2	51/220/15	3	**A** ($= x^7$)
$\mathsf{YSH}_{8,1}$: deg. 7, δ=6 (C.26)	8	6	56	12	7/7	$[6^1, 7^{254}]$	1	.0781	43/107/7	4	43/109/10	4	**B** ($\mathsf{YSH}_{8,1}$)
$\mathsf{YSH}_{8,2}$: deg. 7, δ=10 (C.27)	8	10	72	22	7/7	$[6^3, 7^{252}]$	0	.1094	28/61/1	6	28/84/2	6	**B** ($\mathsf{YSH}_{8,2}$)
$\mathsf{YSH}_{8,3}$: deg. 7, δ=14 (C.28)	8	14	72	38	7/7	$[4^1, 5^2, 6^{60}, 7^{192}]$	0	.1094	18/71/2	5	18/77/8	6	**B** ($\mathsf{YSH}_{8,3}$)
AES, D=3 (C.29)	8	4	32	6	7/7	$[7^{255}]$	3	.0625	42/101/4	3	42/106/4	3	**A** ($\sim_{\text{aff}}$ AES)
AES FIPS-197 (C.30)	8	4	32	6	7/7	$[7^{255}]$	0	.0625	36/99/4	4	36/95/4	4	**A** (= AES)

A and B with finite-field automorphisms and second-derivative conjugate signatures) tests every invariant match for $S(x) = B \cdot S_{\text{ref}}(Ax \oplus a) \oplus b$ with invertible A, B. Every invariant collision resolves with **zero unresolved entries** (Category U = 0), partitioning the archive into two certified categories. **Category A** (2,681 archive permutations; 18 of 30 benchmark circuits in Table 2) covers **exact lookup-table equality** ('='; 15 archive permutations, 7 benchmark circuits) and **certified affine equivalence** ('$\sim_{\text{aff}}$'; 2,666 archive permutations, 11 benchmark circuits, with (A, a, B, b) witnesses checked on all 2^n inputs in SM C and orbit counts in SM B.1). **Category B** (28,305 archive permutations; 12 benchmark classes $\mathsf{YSH}_{5,1}$, $\mathsf{YSH}_{5,2}$, $\mathsf{YSH}_{6,1}$–$\mathsf{YSH}_{6,6}$, $\mathsf{YSH}_{7,1}$, and $\mathsf{YSH}_{8,1}$–$\mathsf{YSH}_{8,3}$ in Table 2) holds permutations certified EA-separated from the 712-entry catalogue (and from all 2,263 quadratic 6-bit classes of [DMB19] and their inverses, since every 6-bit Category B entry has forward and inverse algebraic degree at least three). Within Category A, two-point-swapped power maps [BKRS22] and quadratic classes [BBŞ17] account for 904 archive permutations (899 at $n = 5$, 5 at $n = 7$) and three benchmark circuits: the 5-bit quartic SM C.1 ($\sim_{\text{aff}} x^3_{\text{swap}(1,3)}$), the 5-bit quadratic SM C.2 ($\sim_{\text{aff}} Q_{71}$; since $\mathcal{D}(Q_{71}) = [2^{31}]$, its five coordinates are linearly independent modulo affine functions RM$(1, 5)$, making our 5-AND, $D = 1$ circuit provably AND-optimal), and the 7-bit degree-6 SM C.21 ($\sim_{\text{aff}} x^5_{\text{swap}(1,25)}$). Likewise, our 9-AND 6-bit cubic involution (SM C.17) is certified $\sim_{\text{aff}}$ Feistel$_{3\text{R}}(\chi_3)$ [CDL16, BDMD+20]; although it shares all forward invariants with x^{13}, it is proven EA-inequivalent to x^{13} by $\mathcal{D}(S^{-1}) = [3^{63}]$ vs. $[2^{63}]$, $\beta = 64$ vs. 4, and its third-derivative rank multiset $[2^{273}, 3^{840}, 4^{840}]$ vs. $[3^{189}, 4^{1764}]$. Finally, for our 8-bit $D = 3$ AES-class circuit (SM C.29, 42/101/4), the witness to FIPS-197 AES has $A = I_8$,

Table 3: Gate-exact Yosys synthesis and ABC NLDM timing of the 30 forward (S) circuits on NanGate 45 nm (S^{-1}: Table 9 in SM B; GE in SM C). **Unmasked**: `AND2_X1/XOR2_X1/INV_X1` (A/X/N), longest topological path (LTP), area (μm^2), and delay (ps). **Masked** ($d = 2$ `DOM-indep`): latency D (cycles), random bits $R = N_{\text{AND}}$, cells (A/X/N), `DFF_X1` flip-flops FF (gadget G + sync S), area (μm^2), stage delay (ps), and N_{reconv} (AND gates with intra-stage fan-in reconvergence, Sections 5.2 and 7.3).

		Unmasked Combinational (S)				**Masked ($d = 2$ `DOM-indep`, S)**					
S-box (SM)	n	A/X/N	LTP	μm^2	Delay (ps)	D/R	A/X/N	FF (G+S)	μm^2	Stage (ps)	N_{reconv}
$x^3_{\text{swap}(1,3)}$ class (C.1)	5	11/20/3	7	45.22	334.4	2/11	44/84/3	66 (44+22)	480.93	310.0	0
Q_{71} class (C.2)	5	5/12/3	5	26.07	198.3	1/5	20/44/3	30 (20+10)	228.76	202.1	4
$\mathsf{YSH}_{5,1}$: cubic, δ=8 (C.3)	5	6/11/0	5	23.94	231.6	2/6	24/46/0	48 (24+24)	316.01	187.6	0
$\mathsf{YSH}_{5,2}$: quartic, δ=8 (C.4)	5	5/12/1	7	25.00	295.4	2/5	20/44/1	40 (20+20)	272.92	246.2	1
Gold x^5 (C.5)	5	7/22/0	6	42.56	311.2	1/7	28/72/0	38 (28+10)	316.54	288.5	2
Gold class, SAC-ideal (C.6)	5	7/21/4	6	43.09	282.0	1/7	28/70/4	38 (28+10)	315.48	243.3	0
χ_5 class (C.7)	5	5/8/1	4	18.62	141.2	1/5	20/36/1	28 (20+8)	205.88	139.0	1
inverse-Gold (C.8)	5	14/31/2	8	65.44	439.6	2/14	56/118/2	88 (56+32)	646.91	346.2	0
x^{15} (C.9)	5	14/45/0	13	86.72	710.9	2/14	56/146/0	76 (56+20)	636.27	518.9	7
$\mathsf{YSH}_{6,1}$: quartic, δ=4 (C.10)	6	9/27/1	16	53.20	916.7	3/9	36/90/1	74 (36+38)	517.10	561.1	3
$\mathsf{YSH}_{6,2}$: quartic, k=5 (C.11)	6	27/64/0	10	130.87	502.4	3/27	108/236/0	172 (108+64)	1269.35	434.1	1
$\mathsf{YSH}_{6,3}$: cubic, δ=4 (C.12)	6	9/17/9	11	41.50	411.0	3/9	36/70/9	72 (36+36)	480.40	254.4	0
$\mathsf{YSH}_{6,4}$: deg. 5, δ=8 (C.13)	6	11/22/1	9	47.35	427.0	3/11	44/88/1	80 (44+36)	549.56	358.1	0
$\mathsf{YSH}_{6,5}$: deg. 5, δ=6 (C.14)	6	12/31/0	10	62.24	453.9	4/12	48/110/0	98 (48+50)	669.79	368.8	1
$\mathsf{YSH}_{6,6}$: quartic, δ=8 (C.15)	6	10/22/0	7	45.75	362.8	2/10	40/84/0	66 (40+26)	475.08	317.9	1
Q_{2258} class (C.16)	6	8/34/0	9	62.78	527.6	1/8	32/100/0	44 (32+12)	392.62	354.6	6
χ_3-Feistel invol. (C.17)	6	9/15/9	10	38.30	336.8	3/9	36/66/9	72 (36+36)	474.01	254.4	0
inversion (C.18)	6	21/48/0	15	98.95	797.5	3/21	84/180/0	152 (84+68)	1064.00	367.4	3
Dillon APN (C.19)	6	28/68/0	10	138.32	538.6	2/28	112/248/0	154 (112+42)	1211.36	460.5	0
$\mathsf{YSH}_{7,1}$: quartic, δ=4 (C.20)	7	15/70/10	14	133.00	750.9	2/15	60/200/10	88 (60+28)	786.30	478.9	14
$x^5_{\text{swap}(1,25)}$ class (C.21)	7	51/76/1	16	176.09	840.4	3/51	204/356/1	284 (204+80)	2070.01	802.7	0
Gold x^3 (C.22)	7	14/57/0	9	105.87	558.8	1/14	56/170/0	72 (56+16)	656.49	409.5	5
Kasami x^{13} (C.23)	7	30/114/0	16	213.86	1007.6	2/30	120/348/0	148 (120+28)	1352.34	644.6	19
inversion (C.24)	7	42/182/6	33	338.35	2043.5	3/42	168/532/6	212 (168+44)	1989.68	718.5	42
power map x^7 (C.25)	7	56/98/0	14	215.99	772.4	2/56	224/420/0	280 (224+56)	2174.82	765.9	0
$\mathsf{YSH}_{8,1}$: deg. 7, δ=6 (C.26)	8	43/107/7	25	220.25	1529.9	4/43	172/386/7	236 (172+64)	1869.98	574.7	13
$\mathsf{YSH}_{8,2}$: deg. 7, δ=10 (C.27)	8	28/61/1	17	127.68	969.0	6/28	112/234/1	208 (112+96)	1433.74	275.1	2
$\mathsf{YSH}_{8,3}$: deg. 7, δ=14 (C.28)	8	18/71/2	18	133.53	1084.7	5/18	72/214/2	160 (72+88)	1142.74	431.6	3
AES, D=3 (C.29)	8	42/101/4	24	208.01	1516.4	3/42	168/370/4	216 (168+48)	1748.15	574.7	12
AES FIPS-197 (C.30)	8	36/99/4	21	198.44	1148.6	4/36	144/342/4	282 (144+138)	1976.38	371.0	12

$a = 0$, B a pure wire permutation reversing bit order within each 4-bit output nibble, and $b =$ 0x0F: moving two of its four output NOT gates and reordering output wires yields the exact FIPS-197 AES S-box at $D = 3$ with zero extra gates.

6.2 Gate-Exact NanGate 45 nm Silicon Synthesis

Table 3 reports the post-synthesis standard-cell area and timing of all 30 forward circuits mapped gate for gate onto the NanGate 45 nm Open Cell Library [Nan11] in Yosys [WG13] and timed in Berkeley ABC [BM10], both unmasked and as D-stage pipelined first-order ($t = 1, d = 2$) `DOM-indep` modules (Section 5.2); Table 9 reports the 30 inverse circuits. Unmasked combinational delay tracks LTP, ranging from 141.2 ps (LTP = 4, χ_5 class) to 2,043.5 ps (LTP = 33, 7-bit inversion). Under pipelined `DOM-indep`, registering every AND level cuts the critical path to a single stage (139.0–802.7 ps across all 30 forward circuits, e.g., 1,516.4 → 574.7 ps on $D = 3$ AES and 2,043.5 → 718.5 ps on 7-bit inversion). At the same time, total flip-flops ($4N_{\text{AND}} + 2N_{\text{sync}}$) scale with both N_{AND} and D: moving from $D = 4$ FIPS-197 AES (36 AND, 138 sync flip-flops, 1,976.38 μm^2) to $D = 3$ AES (42 AND, 48 sync flip-flops, 1,748.15 μm^2) cuts sync registers by nearly two thirds and masked area by 11.5%.

6.3 Inverse S-Box Circuit Synthesis (S^{-1})

Direct ANF synthesis of S^{-1} blows up in both gate count and depth. Instead, we compile compact, low-depth S^{-1} circuits for all 30 S-boxes (SM C, Table 9) via a four-tier algebraic and structural pipeline followed by the Stage-4 pass (full derivations in SM B.2).

1. **Tier 1: involutions and affine-folded core reuse (9 circuits).** When S^{-1} is affine equivalent to S or to another archived circuit, the identity $S^{-1}(y) =$

$B \cdot S_{\text{ref}}(Ay \oplus a) \oplus b$ reuses the non-linear core of S_{ref} with zero extra AND gates and depth. For the 6-bit cubic involution, $S^{-1} = S$ (9/15/9, $D = 3$). For the five inversion-class circuits, Dillon's APN permutation, the inverse-Gold S-box (whose inverse $\sim_{\text{aff}} x^5$ reuses our 7-AND Gold core), and the SAC-ideal Gold-class circuit (whose inverse reuses the x^{25} core of the Gold x^5 row), we split S_{ref} into its input linear cone, non-linear core, and output linear cone, fold (A, a) and (B, b) into those cones over $\mathbb{F}_2$, and re-synthesize both linear layers via Paar's CSE [Paa97]. This yields 21/50/0 ($D = 3$, 13 fewer XORs than the published 21/63) for 6-bit inversion, 42/106/4 ($D = 3$) for $D = 3$ AES, 36/95/4 ($D = 4$) for FIPS-197 AES, and 7/26/5 ($D = 1$) for inverse-Gold (halving both AND count and depth).

2. **Tier 2: structural reversal, shear unwinding, directional-derivative and transposition inversion, and backward unrolling (13 circuits).** Every Category B circuit (plus the swapped Gold circuits of SM C.1, C.21) is a generalized Feistel network or a shear/transposition extension of a lower-degree core Q, so we invert its structure directly. Reversing the rounds of the 3-round Feistel networks $\mathsf{YSH}_{6,1}$ and $\mathsf{YSH}_{6,3}$ and the six-round single-bit-branch network $\mathsf{YSH}_{8,2}$ gives 9/27/1, 9/17/8 ($D = 3$), and 28/84/2 ($D = 6$). For the output-sheared cores of $\mathsf{YSH}_{6,2}$, $\mathsf{YSH}_{7,1}$, and $\mathsf{YSH}_{8,1}$ ($Q \sim_{\text{aff}} x^{38}$, x^3, x^{254}), unwinding the involution shear takes one AND gate before Q^{-1}, giving 9/37/0 ($D = 2$, a 3× AND reduction over the 27-AND forward circuit), 30/158/12 ($D = 3$), and 43/109/10 ($D = 4$). For the shear extensions of $Q \sim_{\text{aff}} x^5$ ($\mathsf{YSH}_{6,6}$, $\mathsf{YSH}_{6,4}$, $\mathsf{YSH}_{6,5}$), all three inverses share an 18-AND, $D = 2$ tower-field norm core $Q^{-1}(y)$: $\mathsf{YSH}_{6,6}^{-1}$ evaluates the 9-AND vector directional derivative $s(y) \wedge \Delta_c Q^{-1}(y)$ in parallel at minimum depth $D = 2$ (27/50/3), while for $\mathsf{YSH}_{6,4}^{-1}$ and $\mathsf{YSH}_{6,5}^{-1}$ the forward shears collapse under Q^{-1} to 4-AND and 3-AND scalar rank-1 shift corrections $Q^{-1}(y) \oplus \sum_j s_j(y) c_{\text{in},j}$ at $D = 3$ (22/49/1 and 21/57/0, one level shallower than $\mathsf{YSH}_{6,5}$). Similarly, for $x^3_{\text{swap}(1,3)}$ and $x^5_{\text{swap}(1,25)}$, a parallel two-point transposition indicator with Q^{-1} gives 13/38/6 ($D = 2$) and 34/164/9 ($D = 3$, 17 fewer ANDs than forward). Finally, for $\mathsf{YSH}_{5,2}$ and $\mathsf{YSH}_{8,3}$, backward unrolling with linear-span closure reuses each forward AND gate once, giving 5/11/2 ($D = 2$, matching S) and 18/77/8 ($D = 6$).
3. **Tier 3: composite-field norm, Gold composition, and Karatsuba power-map factorizations (4 circuits).** For Category A power maps whose inverses have higher degree, we factor the inverse exponent into shallow algebraic stages. The 6-bit Q_{2258} (x^5) inverse $x^{38} = y^2 \cdot N^4$ uses the relative norm $N = y^9 \in \mathbb{F}_{2^3}$ (18/37/0 at $D = 2$). At $n = 7$, the Gold (x^3) inverse x^{85} factors via $85 \equiv 5 \times 9 \times 16 \pmod{127}$ using our $D = 1$ circuits for y^5 and z^9 (29/149/15 at $D = 2$, the minimum depth for degree 4), while a 23-AND, $D = 1$ Karatsuba multiplier over $\mathbb{F}_2[z]/(z^7 + z + 1)$ gives the Kasami (x^{13}) inverse $y^{88} = (y^3 \cdot y^8)^8$ (37/139/0 at $D = 2$) and the x^7 inverse $y^{109} = ((y^5)^5)^{64} \cdot (y^5)^{32}$ (51/220/15 at $D = 3$, five fewer AND gates than forward).
4. **Tier 4: 5-bit Hodge-dual coset-span synthesis (4 circuits).** The remaining 5-bit S-boxes (the Q_{71}-class circuit, $\mathsf{YSH}_{5,1}$, Gold x^5, and the χ_5-class circuit) all have cubic inverses with uniform spectrum $\mathcal{D}(S^{-1}) = [3^{31}]$. Decomposing each cubic component via Hodge duality on $\mathbb{F}_2^5$ as $L_c(y) \cdot Q_c(y) \oplus R_c(y)$ with linear L_c and $\deg Q_c, \deg R_c \leq 2$, and sharing $r \in \{4, 5\}$ level-1 AND gates across five independent components, yields exact $D = 2$ circuits: 9/32/2, 9/22/2, 9/21/3 for the χ_5-class inverse (6 fewer XORs than 9/27/2 in [BDM26b]), and 10/37/4 for the Gold inverse x^{25} (matching 10/37 in [BDM26b] and saving 4 ANDs, at +6 XORs, over our 14/31/2 circuit in SM C.8).

Table 4: Comparison with published circuits (AND/XOR/D, NOT in parentheses; Δ = published − ours). Baselines are from [BDM26b] unless noted ([JBKK25, KBJ+26] count 2-fanin #NL ∈ {AND, OR, NAND, NOR} and #L ∈ {XOR, XNOR}): [a]depth-1 minimizer [BDM26a] (x^3 on $\mathbb{F}_{2^5}$ is 7/19/1; on 7-bit Gold x^3, SLICE [KBJ+26] reaches 11/41/1 at LTP = 6 and 18/40/1, 20/37/1 at LTP = 5); [b]Tier-4 inverse of Gold x^5 (C.5), archive circuit (C.8) is 14/31/2 (2) with S^{-1} at 7/26/1 (5); [c]exact x^{38} inverse (C.16) is 18/37/2; [d]smallest random $(\delta, L) = (4, 16)$ quadratic 5-bit S-box in [BDM26a] (5/20, 5/24, 5/24); [e]Tier-4 inverse of χ_5-class (C.7); [f]depth-3 AES of [HB25] (42/97/3, LTP = 14), refined by SLICE [KBJ+26] to 42/94/3 (LTP = 13, $\Delta = 0/-7$) and 42/107/3 (LTP = 12, $\Delta = 0/+6$); [g]SLICE [KBJ+26] also reaches 20/49/2 (LTP = 7) and 12/39/3 (LTP = 10); [h]at $D = 4$, [JBKK25] reaches 33/85/4 (LTP = 23) and 34/81/4–34/100/4 (LTP ∈ [15, 28]); SLICE [KBJ+26] reaches 33/95/4 (LTP = 14); at $D \in \{5, 6\}$, [JBKK25] reaches 32/81; [i](AND, XOR) circuit of [BDM26b] at $D = 3$ (native SPEEDY [LMMR21] uses 12 two-level NAND/NOR gates at $D = 3$).

n	S-box or profile	Cat.	(δ, L, d)	Published	**Ours (S)**	**Ours (S^{-1})**	Δ (S)	**Class (SM)**
5	Keccak χ_5 [BDMD+20]	A	(8, 16, 2)	5/5/1 (5)	5/8/1 (1)	9/21/2 (3)	0/−3	$\sim_{\text{aff}}$ χ_5 (C.7)
5	Gold x^5	A	(2, 8, 2)	7/21/1[a]	7/22/1	10/37/2 (4)	0/−1	$= x^5$ (C.5)
5	Gold class, SAC-ideal	A	(2, 8, 2)	7/21/1[a]	7/21/1 (4)	10/41/2 (10)	0/0	$\sim_{\text{aff}}$ x^5 (C.6)
5	Q_{71} class [BBŞ17]	A	(4, 16, 2)	5/20/1[d]	5/12/1 (3)	9/32/2 (2)	0/+8	$\sim_{\text{aff}}$ Q_{71} (C.2)
5	χ_5^{-1} class	A	(8, 16, 3)	9/27/2	9/21/2 (3)[e]	5/8/1 (1)	0/+6	$\sim_{\text{aff}}$ χ_5^{-1} (C.7 inv.)
5	$\mathsf{YSH}_{5,1}$ (cubic, δ=8)	B	(8, 16, 3)	9/27/2	6/11/2	9/22/2 (2)	+3/+16	$\mathsf{YSH}_{5,1}$ (C.3)
5	$\mathsf{YSH}_{5,2}$ (quartic, δ=8)	B	(8, 16, 4)	–	5/12/2 (1)	5/11/2 (2)	–	$\mathsf{YSH}_{5,2}$ (C.4)
5	inverse Gold x^{25}	A	(2, 8, 3)	10/37/2	10/37/2 (4)	7/22/1	0/0	$= x^{25}$ (C.5 inv.)[b]
5	inversion x^{30}	A	(2, 12, 4)	12/51/2	14/45/2	14/57/2	−2/+6	$= x^{15}$ (C.9)
5	$x^3_{\text{swap}(1,3)}$ vs. DB$_1$ [AL26]	A	(4, 12, 4)	9/36/2	11/20/2 (3)	13/38/2 (6)	−2/+16	$\sim_{\text{aff}}$ $x^3_{\text{swap}(1,3)}$ (C.1)
6	Q_{2258} (x^5 class)	A	(4, 16, 2)	8/25/1[a]	8/34/1	18/37/2	0/−9	$\sim_{\text{aff}}$ x^5 (C.16)
6	Feistel$_{3R}(\chi_3)$ vs. Q_{2258}^{-1}	A	(4, 16, 3)	11/42/2	9/15/3 (9)	9/15/3 (9)	+2/+27	$\sim_{\text{aff}}$ Feistel$_{3R}(\chi_3)$ (C.17)[c]
6	$\mathsf{YSH}_{6,1}$ (quartic, δ=4)	B	(4, 16, 4)	–	9/27/3 (1)	9/27/3 (1)	–	$\mathsf{YSH}_{6,1}$ (C.10)
6	Dillon APN	A	(2, 16, 4)	17/75/2[g]	28/68/2	28/78/2 (7)	−11/+7	= Dillon (C.19)
6	inversion x^{62} (div.)	A	(4, 16, 5)	29/109/3	21/48/3	21/50/3	+8/+61	$\sim_{\text{aff}}$ x^{62} (C.18)
6	inversion x^{62} (tower)	A	(4, 16, 5)	21/63/3	21/48/3	21/50/3	0/+15	$\sim_{\text{aff}}$ x^{62} (C.18)
6	SPEEDY [LMMR21]	B	(8, 24, 5)	20/51/3[i]	12/31/4	21/57/3	+8/+20	$\mathsf{YSH}_{6,5}$, δ=6 (C.14)
7	Gold x^3	A	(2, 16, 2)	11/45/1[a]	14/57/1	29/149/2 (15)	−3/−12	$= x^3$ (C.22)
7	Kasami x^{13}	A	(2, 16, 3)	27/115/2	30/114/2	37/139/2	−3/+1	$= x^{13}$ (C.23)
7	power map x^7	A	(6, 40, 3)	30/140/2	56/98/2	51/220/3 (15)	−26/+42	$= x^7$ (C.25)
7	inversion x^{126}	A	(2, 20, 6)	–	42/182/3 (6)	42/229/3 (6)	–	$\sim_{\text{aff}}$ x^{126} (C.24)
7	$\mathsf{YSH}_{7,1}$ (quartic, δ=4)	B	(4, 32, 4)	–	15/70/2 (10)	30/158/3 (12)	–	$\mathsf{YSH}_{7,1}$ (C.20)
7	$x^5_{\text{swap}(1,25)}$ class	A	(4, 20, 6)	–	51/76/3 (1)	34/164/3 (9)	–	$\sim_{\text{aff}}$ $x^5_{\text{swap}(1,25)}$ (C.21)
8	AES (FIPS-197)	A	(4, 32, 7)	33/84/4[h]	36/99/4 (4)	36/95/4 (4)	−3/−15	= AES (C.30)
8	AES class (D=3)	A	(4, 32, 7)	42/97/3[f]	42/101/3 (4)	42/106/3 (4)	0/−4	$\sim_{\text{aff}}$ AES (C.29)
8	$\mathsf{YSH}_{8,2}$ (deg. 7, δ=10)	B	(10, 72, 7)	–	28/61/6 (1)	28/84/6 (2)	–	$\mathsf{YSH}_{8,2}$ (C.27)
8	$\mathsf{YSH}_{8,3}$ (deg. 7, δ=14)	B	(14, 72, 7)	–	18/71/5 (2)	18/77/6 (8)	–	$\mathsf{YSH}_{8,3}$ (C.28)

6.4 Comparison with Published Circuits

Table 4 compares our circuits with the best published implementations, including top-down polynomial division [BDM26a, BDM26b], the depth-3 AES inverter of Hadžić and Bloem [HB25], whole-S-box Boyar–Peralta optimization (eRNBP/eBPD) [JBKK25], and segmented low-depth synthesis (SLICE) [KBJ+26]. Positive entries in Δ = published−ours indicate gate savings by Yoshan. Category A rows test gate and depth minimization within a fixed function or proven affine class (or profile baseline where noted), whereas Category B rows measure the hardware savings unlocked when the lookup table can vary across an EA-separated permutation with the same or better $(\delta, L, \deg)$ profile.

6.5 Provenance and Stage Ablation of the Benchmark Circuits

Table 10 (SM B) traces each of the 30 benchmark circuits across the pipeline, from its Stage-1 seed and gate count through the certified netlist archived after Stages 2–3 to the final netlist printed in SM C after the Stage-4 pass (sub-DAG shaver and stage-preserving Paar CSE, Section 4.3). Three clear lessons emerge from this trace.

Where the circuits come from (Stage 1). Except for the Category A rows that reproduce a catalogue or library function verbatim (the seven '=' rows of Table 2 and the Q_{2258} circuit, which computes the Q_{2258} table itself), no printed lookup table equals

any entry of the circuit library, 5-bit database, or catalogue; in particular, none of the 12 Category B permutations does. Of the 30 circuits, 15 start from a library circuit (source (2), Section 4.1), 11 from a structural seed (source (3)), and 4 from an archive elite (source (1)). Within Category B, eight of the 12 classes descend from a library seed that Stage 2 moved into an EA-separated class: $\mathsf{YSH}_{6,2}$ from the cubic Q_{2258}^{-1} (x^{38}) seed (legacy key `q2256_inv_6b_11and_d2`), $\mathsf{YSH}_{7,1}$ from the fractional subfield tower, $\mathsf{YSH}_{8,2}$ from the Canright AES tower [Can05], $\mathsf{YSH}_{6,1}$ from a 9-AND 3-round Feistel library circuit (`golden_6b_row26`), $\mathsf{YSH}_{5,1}$ from the KECCAK χ_5 circuit, and $\mathsf{YSH}_{6,4}$–$\mathsf{YSH}_{6,6}$ from the 9-AND Q_{2258} seed. Two others start from structural templates—the 3-round Feistel template ($\mathsf{YSH}_{6,3}$) and CCZ-lifted inversion ($\mathsf{YSH}_{8,1}$)—and two start from archive elites: $\mathsf{YSH}_{5,2}$ from a 16-AND, $D = 2$, $\delta = 6$ elite (of which Stage 2 kept five AND gates) and $\mathsf{YSH}_{8,3}$ from $\mathsf{YSH}_{8,2}$ via Stage-2 AND erasure (-10 AND, -1 level). In Category A, three start from structural seeds that landed in other catalogue classes: a power-map LUT node for $x^3_{\mathrm{swap}(1,3)}$ (SM C.1), the depth-1 quadratic template for Q_{71} (SM C.2), and an unbalanced Feistel template for $x^5_{\mathrm{swap}(1,25)}$ (SM C.21). The remaining Category A circuits start from the library circuit, Misty template ($\mathrm{Feistel}_{3\mathrm{R}}(\chi_3)$), or power-map node of their own class, except for the Kasami and χ_5-class circuits, which descend from archive elites. Notably, for $\mathsf{YSH}_{5,1}$ and $\mathsf{YSH}_{6,4}$–$\mathsf{YSH}_{6,6}$ the seed core survives intact inside the archived netlist (the depth-1 χ_5 core in $\mathsf{YSH}_{5,1}$ and all nine depth-1 products of Q_{2258} in $\mathsf{YSH}_{6,4}$–$\mathsf{YSH}_{6,6}$), while the Q_{71}-class circuit (SM C.2) is a depth-1 quadratic template instance whose $(\delta, L) = (4, 16)$ profile passed the grade gate directly.

What Stage 2 changes. Stage 2 modifies permutations as well as circuits, reshaping the cryptographic profile first. Its degree-raising shear mutation produces four Category B classes: adding one shear gate to the χ_5 circuit (5/10, $D = 1$) yields the cubic $\mathsf{YSH}_{5,1}$ (6/11/0, $D = 2$, keeping δ, L while shifting $\mathcal{D}(S)$ from $[2^{31}]$ to $[2^{15}, 3^{16}]$), and chaining one, two, and three shear gates onto the Q_{2258} circuit (9/30, $D = 1$, $(\delta, L, \deg) = (4, 16, 2)$) yields $\mathsf{YSH}_{6,6}$ ($(8, 24, 4)$, $D = 2$), $\mathsf{YSH}_{6,4}$ ($(8, 28, 5)$, $D = 3$), and $\mathsf{YSH}_{6,5}$ ($(6, 24, 5)$, $D = 4$). Each shear buys one algebraic degree for one AND gate and one level, trading off δ. Other function-changing moves swap two LUT entries of a 5-bit x^3 power-map node and lower it to the quartic $x^3_{\mathrm{swap}(1,3)}$ (SM C.1, 11/20/3, $D = 2$), rewire `golden_6b_row26` into the quartic $\mathsf{YSH}_{6,1}$ (9/27, $D = 3 \to$ 9/29/1, $D = 3$, $\mathcal{D}(S) = [3^7, 4^{56}]$), turn the cubic Q_{2258}^{-1} seed into the quartic $\mathsf{YSH}_{6,2}$ (26/65, $D = 2 \to$ 27/64/0, $D = 3$, whose inverse unwinds one output shear onto the 8-AND quadratic Q_{2258} core), and lower a mutated unbalanced Feistel template into the 7-bit swapped Gold circuit $x^5_{\mathrm{swap}(1,25)}$ (SM C.21, 51/77/1, $D = 3$). Likewise, AND erasure shrinks a 16-AND, $D = 2$ elite to the 5-AND quartic $\mathsf{YSH}_{5,2}$ (16/39 $\to$ 5/15/1, $D = 3$) and transforms the Canright AES tower into $\mathsf{YSH}_{8,2}$ (36/136, $D = 4 \to$ 28/61/1, $D = 6$: -8 AND, -75 XOR at $+2$ levels and $(\delta, L) = (10, 72)$). Applied repeatedly to $\mathsf{YSH}_{8,2}$, AND erasure alone traces the certified degree-7 family of Table 8 (SM B) from 28 AND ($\delta = 10$) down to 18 AND ($\delta = 14$, $\mathsf{YSH}_{8,3}$). When the seed already computes the target class, Stage 2 and in-loop CSE trim linear gates (67 $\to$ 58 XOR for 6-bit inversion, 116 $\to$ 103 for $D = 3$ AES, 120 $\to$ 107 for $\mathsf{YSH}_{8,1}$) or one AND gate (9 $\to$ 8 for Q_{2258}, 43 $\to$ 42 at -1 level for 7-bit inversion), and also reach the SAC-ideal x^5 representative of SM C.6 (7/21/4, $D = 1$). By contrast, the Piling-Up/LSAP module (Section 4.2) contributed zero benchmark circuits: across the nine telemetry-enabled runs (4,061 netlists), `diff_piling_up_lsap` appears on 111 netlists (98 LUTs) and 26 direct-graduate netlists (23 at $n = 5$, 18 in $(8, 16, 4)$; 1 each at $n = 6, 7, 8$), plus 96,344 LSAP crossover calls at 0.18–0.75 s/seed ($< 1\%$ wall time), none beating Table 2.

What Stage 4 changes. Stage 4 is function-preserving and never raises depth (checked on all 2^n inputs). On the 30 forward circuits, it cut 182 XORs and one AND ($\mathsf{YSH}_{6,3}$: 10 $\to$ 9), left 10 unchanged, and lowered $\mathsf{YSH}_{5,2}$ from 5/15/1, $D = 3$ to 5/12/1, $D = 2$ (its level-3 AND input was ready at level 1). Top forward XOR cuts are 177 $\to$ 98 (x^7), 119 $\to$ 99 (FIPS-197 AES), 50 $\to$ 34 (Q_{2258}), and 43 $\to$ 31 ($\mathsf{YSH}_{6,5}$). On the 30 inverse circuits (after

each tier's CSE), it cut 2 ANDs ($\mathsf{YSH}_{6,3}^{-1}: 10 \to 9$; $\mathsf{YSH}_{6,5}^{-1}: 22 \to 21$) and 189 XORs across 18 circuits (up to −87 on $\mathsf{YSH}_{8,2}^{-1}$, −67 on $\mathsf{YSH}_{8,3}^{-1}$).

6.6 Discussion

Co-synthesis vs. fixed-target minimization. When a cipher designer specifies a target profile $(\delta, L, \deg)$ rather than a fixed lookup table, co-synthesis cuts non-linear cost substantially. On SPEEDY's 6-bit profile, $\mathsf{YSH}_{6,5}$ $(12/31/0, D = 4)$ improves (δ, β) from $(8, 36)$ to $(6, 14)$ at $L = 24$, matching native SPEEDY's 12 NAND/NOR gates $(D = 3)$ [LMMR21] in (AND, XOR) and cutting 20/51 [BDM26b] to 12/31 at +1 level. Likewise, on χ_5^{-1}'s profile, $\mathsf{YSH}_{5,1}$ saves 3 AND and 16 XOR gates; the self-inverse 3-round χ_3-Feistel saves 2 AND and 27 XOR gates against Q_{2258}^{-1} while matching x^{13}'s forward invariants; and $\mathsf{YSH}_{6,1}$ reaches $(4, 16, 4)$ with $\mathcal{D}(S) = \mathcal{D}(S^{-1}) = [3^7, 4^{56}]$ at 9/27/1 in both directions. In Category A, co-synthesis also pays off whenever pipeline depth drives masked cost. Compared with $D = 4$ AES (33/84/4 [BDM26b], 33/85/4 at LTP = 23 or 34/81/4 at LTP = 28 [JBKK25], 33/95/4 at LTP = 14 [KBJ+26]) and $D \in \{5, 6\}$ (32/81 [JBKK25]), our $D = 3$ AES-class circuit $(42/101/4, \text{LTP} = 24)$ trades 9–10 AND gates for 1–3 fewer pipeline stages. Its non-linear core is the $D = 3$ inverter of Hadžić and Bloem [HB25] $(42/97/3, \text{LTP} = 14)$, refined by SLICE [KBJ+26] via whole-circuit eBPD to 42/94/3 (LTP = 13) and 42/107/3 (record LTP = 12); we contribute the gate-exact netlist pair with its $D = 3$ inverse (42/106/4) and full masked-silicon analysis. Under $d = 2$ `DOM-indep` (Tables 3 and 9), moving from our $D = 4$ FIPS-197 circuit (36/99/4) to $D = 3$ saves 90 forward sync flip-flops against 24 extra gadget flip-flops, dropping total flip-flops from 282 to 216 (276 to 216 for S^{-1}) and masked area by 11.5%/8.9% (33/84/4 with 4 NOT [BDM26b] already needs 132 gadget flip-flops and 1,218.28 μm^2 before sync registers by Eq. (8); composable `HPC2` and `HPC3`/`HPC3.1` save 516 vs. 636 and 216 vs. 282 flip-flops, SM A.1). At stage delay 574.7 vs. 371.0 ps, three cycles win for any clock period ≥ 575 ps. We also give the first gate-level and provably AND-optimal 5-AND, $D = 1$ circuit (5/12/3, or 5/11/1 pre-Stage-4) for the 5-bit Q_{71} class [BBŞ17] (SM C.2), trim 6-bit tower inversion from 21/63 to 21/48 (S^{-1}: 21/50, $D = 3$), and reach SAC deviation 0 in the x^5 class $(7/21/4, D = 1$, SM C.6). On fixed classical targets, however, dedicated minimizers—quadratic synthesis [BDM26a], A* MC search [JBK24], polynomial division [BDM26b], and SLICE [KBJ+26]—still lead on Q_{2258} (8/25 vs. 8/34), 5-bit x^5 (7/21 vs. 7/22), 7-bit x^3 (11/45 [BDM26a], 11/41 at LTP = 6, or 18/40, 20/37 at LTP = 5 [KBJ+26] vs. 14/57), Dillon's APN (17/75/2 [BDM26b], 20/49/2 at LTP = 7, or 12/39/3 at LTP = 10 [KBJ+26] vs. 28/68/2), 7-bit x^7 (30/140 vs. 56/98), 5-bit inversion (12/51 vs. 14/45), Kasami (27/115 vs. 30/114), and $D = 4$ AES (33–34 vs. 36 AND). The two paradigms complement one another: Yoshan discovers low-AND Category B S-boxes where fixed classes are costly, and eRNBP/eBPD [JBKK25] and SLICE [KBJ+26] can polish its DAGs at fixed (AND, D).

Algebraic structure of the Category B and swapped-power circuits. The 12 Category B classes $\mathsf{YSH}_{n,i}$ (along with the swapped Gold classes $x^3_{\text{swap}(1,3)}$ and $x^5_{\text{swap}(1,25)}$) share the subspace-partition spectrum $\mathcal{D}(S) = [d_1^{2^k-1}, d_2^{2^n-2^k}]$ with $k = \dim V_{\leq d_1}$ (uniform $[3^{63}]$ for $\mathsf{YSH}_{6,3}$; nested flags for $\mathsf{YSH}_{5,2}, \mathsf{YSH}_{8,3}$) and split into two structural families. The first family, *non-Feistel subspace-partition S-boxes*, pairs a wide quadratic first level with a short chain of degree-raising shears or transpositions. Its hyperplane-split circuits $(k = n - 1)$ reach high degree at very low AND cost: 10+1 gates at $D = 2$ for $x^3_{\text{swap}(1,3)}$, 14+1 for $\mathsf{YSH}_{7,1}$, 9+1 for $\mathsf{YSH}_{6,6}$, and 9+2 or 9+3 for the degree-5 $\mathsf{YSH}_{6,4}$ and $\mathsf{YSH}_{6,5}$. Similarly, $\mathsf{YSH}_{6,2}$ $(k = 5)$ and $\mathsf{YSH}_{8,1}$ $(k = 1)$ apply one output shear to a Q_{2258}^{-1} or 8-bit inversion core to eliminate quadratic components without Feistel structure, with $\mathsf{YSH}_{6,2}^{-1}$ unwinding to 9/37/0 at $D = 2$. At $n = 8$, the six-round generalized Feistel $\mathsf{YSH}_{8,2}$ (evolved from the Canright AES tower, Section 6.5) trades $(\delta, L) = (10, 72)$ and two levels for

15 fewer AND gates than $\mathsf{YSH}_{8,1}$ at degree 7, and its AND-erasure descendant $\mathsf{YSH}_{8,3}$ saves ten more AND gates and one level at $(\delta, L, \beta) = (14, 72, 38)$—making it the smallest pipelined `DOM-indep` 8-bit circuit in our study (1,142.74 μm^2, 72 gadget flip-flops). The second family, *Feistel-type S-boxes*, offers symmetric forward and inverse cost. Here $\mathsf{YSH}_{5,2}$ reaches degree 4 at χ_5's 5-AND count and $D = 2$ in both directions (5/12/1 and 5/11/2), while the balanced 3-round networks $\mathsf{YSH}_{6,1}$ and $\mathsf{YSH}_{6,3}$ (3/3/3 AND gates per level) eliminate all quadratic components at 9 AND gates and $D = 3$ with $\delta = 4$, $L = 16$; $\mathsf{YSH}_{6,3}$ (9/17/9, $\mathcal{D}(S) = [3^{63}]$) also reaches $N_{\text{reconv}} = 0$ and a 55% shorter stage delay than $\mathsf{YSH}_{6,1}$.

Boomerang uniformity and the price of 3-round Feistel seeds. Boomerang uniformity β cleanly separates the two families in Table 2. All ten non-Feistel circuits achieve $\beta \in [6, 20]$ (close to AES at 6 and χ_5 at 16, and well below SPEEDY at 36), and the six-round $\mathsf{YSH}_{8,2}, \mathsf{YSH}_{8,3}$ achieve $\beta = 22, 38$. By contrast, $\mathsf{YSH}_{6,1}$, $\mathsf{YSH}_{6,3}$, and $\text{Feistel}_{3\text{R}}(\chi_3)$ hit $\beta = 2^n = 64$, and $\mathsf{YSH}_{5,2}$ hits $\beta = 32$. For 3-round Feistel maps ($L_{i+1} = R_i$, $R_{i+1} = L_i \oplus F_{i+1}(R_i)$ on m-bit branches), this probability-one boomerang switch is classical [LW14, CDL16, BPU16, BC18]. An input difference $(\alpha, 0)$ fixes R_0 and shifts $R_1 = L_0 \oplus F_1(R_0)$ by α, while an output difference $(0, \alpha)$ swaps R_1 and $R_1 \oplus \alpha$ on decryption, leaving right half $R_0 \oplus F_2(R_1) \oplus F_2(R_1 \oplus \alpha)$ and left difference α. Consequently, $\text{BCT}((\alpha, 0), (0, \alpha)) = 2^{2m}$ for all $2^m - 1$ non-zero α (7 maximal entries at $n = 6$). Because Eq. (6) does not include $\beta(S)$, these circuits stayed in the archive for their strong $(\delta, L, \deg)$ profile and inverse symmetry; any cipher using them must get boomerang resistance from its linear layer and round count, whereas the non-Feistel $\mathsf{YSH}_{n,i}$ resist boomerang switches directly (Section 8).

7 Verification

Because the search transforms circuits through stochastic mutations, SMT sub-DAG shaving, and CSE, we enforce soundness with an independent post-search verification pipeline. Exported SystemVerilog netlists are certified against their lookup tables by exhaustive simulation and an SMT miter (Section 7.1), every metric and post-synthesis standard-cell netlist is independently recomputed and formally equivalence-checked (Section 7.2), the equivalence classification of Section 6.1 is solved directly on the lookup tables, and all 60 circuits are audited for symbolic mask independence and intra-stage combinational support disjointness (Section 7.3).

7.1 Certification of the Exported Netlists

Before archiving a candidate, the engine emits its structural SystemVerilog netlist (Section 5.1) and checks the emitted text rather than the in-memory DAG. Behavioral constructs (e.g., `case`) are rejected. The continuous AND/XOR/NOT assignments are evaluated bit-parallel on all 2^n input vectors, and a candidate is archived only if every output bit resolves, forms a permutation, and matches the DAG truth table (a complete proof for $n \leq 8$). Independently of the simulator, a Z3 miter [dMB08] solves $\exists\, x \in \mathbb{F}_2^n : G(x) \oplus \text{LUT}(x) \neq 0$ and requires `unsat`; all 31,337 archived netlists of Section 6 carry both certificates.

7.2 Independent Recomputation and Verification Guarantees

For all 60 listings in SM C, an independent script parses the printed netlist, simulates all 2^n inputs, verifies $S(\text{LUT}_{S^{-1}}[y]) = y$ and $\text{LUT}_{S^{-1}}[S(x)] = x$, counts total and per-level AND gates, and recomputes δ, L, deg, $\deg^{-1}$, $\mathcal{D}(S)$, $\mathcal{D}(S^{-1})$, β, FP, and SAC (producing Table 2 and "Ours" in Table 4). For Tables 3 and 9, every Yosys NanGate 45 nm netlist (60 unmasked and 60 `DOM-indep`) is checked for exact 1-to-1 cell counts and proven

equivalent to its pre-synthesis SystemVerilog module via `equiv_status -assert` (combinational) and `miter -equiv` with `sat -verify -tempinduct` (sequential `DOM-indep`). Every Category A permutation is checked for exact affine equivalence against Table 6 with its (A, a, B, b) witness verified on all 2^n inputs, and all representatives in Table 11 are re-verified from their netlists. Every reported circuit is thus a gate-exact netlist with certified gate counts, depth D, 45 nm silicon equivalence, cryptographic spectra, and (for '=' or '$\sim_{\text{aff}}$') certified catalogue equivalence, while evolutionary gate counts remain upper bounds rather than lower bounds.

7.3 Masking-Related Design Rules and Glitch-Reconvergence Audit

During search, YOSHAN rejects gate-level candidates whose fresh-randomness demand (counted conservatively at two bits per AND gate) exceeds 128 bits per pipeline stage or 1,280 bits per evaluation, as well as any linear gate with fan-out above 16, to bound capacitive coupling [DCBG+17, LBS19]. This cap applies to the whole gate-level circuit (no mutated gate-level candidate with > 64 explicit AND gates is accepted; seeds and LUT nodes lowered at export are exempt), and all 60 benchmark netlists use at most 35 AND gates per stage. Across all 60 circuits, we also audited two structural conditions governing first-order ($t = 1, d = 2$) glitch-extended probing security [GMK16, FGMDP+18]. First, under *linear mask independence*, tracking each wire's symbolic mask vector $\mu(w) \in \mathbb{F}_2^{n+N_{\text{AND}}}$ ($\mu(x_i) = e_i$, $\mu(u \oplus v) = \mu(u) \oplus \mu(v)$, $\mu(\neg u) = \mu(u)$, $\mu(c_k) = e_{n+k}$ for the output of gadget k) confirms across all 593 forward and 649 inverse AND gates (1,242 total) that no AND gate receives an unmasked operand ($\mu(a) = 0$) or two operands with identical steady-state masks ($\mu(a) = \mu(b)$). Second, under *intra-stage support disjointness*, tracing each AND operand's combinational fan-in cone $\text{Supp}_{\text{st}\ell}(w)$ back to the stage-ℓ register boundary shows that 10 forward circuits ($x^3_{\text{swap}(1,3)}$, $\mathsf{YSH}_{5,1}$, SAC-ideal Gold, inverse-Gold, $\mathsf{YSH}_{6,3}$, $\mathsf{YSH}_{6,4}$, 6-bit cubic involution, Dillon APN, $x^5_{\text{swap}(1,25)}$, 7-bit x^7) and 3 inverse circuits ($\mathsf{YSH}^{-1}_{5,1}$, χ_5-class inverse, cubic involution) satisfy $\text{Supp}_{\text{st}\ell}(a) \cap \text{Supp}_{\text{st}\ell}(b) = \emptyset$ at every AND gate. Thus no single glitch-extended probe at $a_0 \wedge b_1$ observes both shares of any stage-ℓ variable or both domains of any preceding gadget, whereas the remaining circuits have $N_{\text{reconv}} > 0$ reconvergent gates (Tables 3 and 9; e.g., 3/9 in $\mathsf{YSH}_{6,1}$, $N_{\text{reconv}} \in \{9, 11, 12\}$ in AES). Both checks are necessary single-stage conditions rather than a full multi-stage proof: stage-$(\ell - 1)$ inner-domain products are registered unblinded, so two gadgets sharing a stage-$(\ell - 1)$ input can still be observed jointly by one glitch-extended probe in stage ℓ. Removing reconvergence uses the mitigations of SM A.1; full SILVER [KSM20] or PROLEAD [MM22] verification is future work.

Artifact. SM C prints the lookup tables and netlists of all 60 circuits, and Table 10 records the Stage-1 seed and stage-by-stage gate counts so every metric, invariant, and provenance claim can be re-derived independently.

8 Design Considerations and Future Directions

Equivalence classification and cipher-level cryptanalysis. While all 2,681 Category A archive permutations are certified identical or affine equivalent to the 712-entry reference catalogue (Table 6) with zero unresolved invariant collisions, two equivalence questions remain open: testing whether the 6-bit χ_3-Feistel cubic involution $\text{Feistel}_{3\text{R}}(\chi_3)$ (proven EA-inequivalent to x^{13} by its third-derivative rank spectrum in Section 6.1) is CCZ-equivalent to x^{13}, and testing the 28,305 Category B permutations against broader multi-point modifications or general CCZ invariants [Kal21]. At the cipher level, hyperplane-split S-boxes ($\mathcal{D}(S) = [2^{2^{n-1}-1}, d_2^{2^{n-1}}]$ for $x^3_{\text{swap}(1,3)}$, $\mathsf{YSH}_{5,1}$, $\mathsf{YSH}_{6,4}$–$\mathsf{YSH}_{6,6}$, $\mathsf{YSH}_{7,1}$, $x^5_{\text{swap}(1,25)}$, and nested $[2^7, 3^8, 4^{16}]$ for $\mathsf{YSH}_{5,2}$) place their $2^{n-1} - 1$ lower-degree com-

ponents inside a hyperplane $H = \langle v^* \rangle^\perp$. An SPN linear layer M using them must therefore: (i) keep $M^\top$ from mapping active next-round masks into H; (ii) diffuse the single high-degree coordinate modulo H across multiple next-round S-boxes so division-property and integral trails cannot bypass degree growth; and (iii) block backward trails through $V_{\leq 2}(S^{-1})$ in decryption and boomerang rounds, since forward degree uniformity can still hide a large quadratic inverse subspace (e.g., $\mathsf{YSH}_{6,2}$ has $\mathcal{D}(S) = [3^{31}, 4^{32}]$ vs. $\mathcal{D}(\mathsf{YSH}_{6,2}^{-1}) = [2^{15}, 3^{48}]$ with $\dim V_{\leq 2}(S^{-1}) = 4$). By contrast, the 3-round Feistel $\mathsf{YSH}_{6,1}$ and $\mathsf{YSH}_{6,3}$ have minimum component degree 3 in all directions, but like any 3-round Feistel permutation [CDL16, BPU16, BC18] they carry $\beta = 2^n$ (Section 6.6), as does $\mathsf{YSH}_{5,2}$ ($\beta = 32$). Ciphers using them must draw boomerang resistance from the linear layer and round count; adding $\beta(S)$ to Eq. (6) alongside 4-round Feistel/Misty seeds is a direct extension (non-Feistel $\mathsf{YSH}_{n,i}$ already satisfy $\beta \leq 20$).

Linear-layer polishing, physical place-and-route, and co-evolution. Coupling Yoshan's archive with linear-network minimizers [BP10, ME19, Sto16], whole-S-box ERNBP/EBPD [JBKK25], and segmented SLICE [KBJ+26] passes can further compress cross-stage XOR cones and LTP at invariant (AND, D). Likewise, SAT technology mapping [JCL+25] can exploit multi-input cells in unmasked flows, while place-and-route parasitics, SILVER/PROLEAD [KSM20, MM22], and joint (S, S^{-1}) fitness are natural next steps.

Continuous-to-discrete relaxation. In the Piling-Up module (Eq. (5)), interior wires with $p_m \approx 0.5$ drive $\prod_m (1 - 2p_m) \to 0$ exponentially with depth, shrinking the relaxed Walsh spectrum before binarization while Hungarian projection crosses sharp algebraic boundaries. Straight-through evaluation, interleaved Hungarian/Sinkhorn steps, and a joint differential-linear objective via the Chabaud–Vaudenay identity [CV95, CCZ98, Car21] $\sum_{\alpha,v} W_S(\alpha, v)^4 = 2^{2n} \sum_{a,b} \mathrm{DDT}(a, b)^2$ offer direct remedies.

9 Conclusion

Under Boolean masking, S-box area and randomness grow with AND count while latency grows with AND depth, confining prior automated searches to depth-1 quadratics or fixed-table minimization. Yoshan overcomes this trade-off by co-synthesizing the permutation and its circuit in one quality-diversity loop that unites structural seeding, cryptographically guided DAG evolution, auxiliary differentiable Piling-Up seeding and crossover with Hungarian LSAP projection, and multi-level optimization with four-tier inverse compilation. Across $n \in \{5, 6, 7, 8\}$, the engine produced 31,337 certified netlists in 56 profiles, with all 60 forward and inverse benchmark circuits verified, synthesized in 45 nm CMOS (unmasked and $d = 2$ `DOM-indep`), and traced in Table 10. In Category A, co-synthesis yields a gate-exact depth-3 AES-class pair (42/101/4 forward, 42/106/4 inverse at $D = 3$ [HB25]) that saves 66/60 flip-flops and 11.5%/8.9% `DOM-indep` area at 25% lower latency than our $D = 4$ FIPS-197 circuit, alongside a provably AND-optimal 5/12/3 ($D = 1$) circuit for Q_{71} [BBŞ17], a 21/48/0 6-bit inversion-class circuit, and 11/20/3 ($D = 2$) and 51/76/1 ($D = 3$, 34-AND inverse) circuits for $x^3_{\mathrm{swap}(1,3)}$ and $x^5_{\mathrm{swap}(1,25)}$ [BKRS22]. Meanwhile, polynomial division [BDM26a, BDM26b], EBPD [JBKK25], and SLICE [KBJ+26] remain best in linear gates and combinational depth on fixed targets. In Category B, freeing the lookup table yields 12 EA-separated benchmark classes across two structural families: non-Feistel subspace-partition S-boxes ($\beta \in [12, 20]$)—including the 12-AND degree-5 $\mathsf{YSH}_{6,5}$ ($\delta = 6, L = 24, \beta = 14$), the 15-AND $D = 2$ 7-bit $\mathsf{YSH}_{7,1}$, the 6-bit $\mathsf{YSH}_{6,2}$ (9-AND $D = 2$ inverse), and the degree-7 8-bit $\mathsf{YSH}_{8,2}$ and $\mathsf{YSH}_{8,3}$ (28 and 18 AND)—and symmetric-cost Feistel-type S-boxes ($\beta = 2^n$), led by the 9-AND $D = 3$ $\mathsf{YSH}_{6,1}$ and $\mathsf{YSH}_{6,3}$ and the 5-AND $D = 2$ $\mathsf{YSH}_{5,2}$.

Acknowledgments

The authors thank Google for providing the computational infrastructure used to conduct this research. R. H., B. A., and I. L. were partly supported by the Israel Science Foundation (ISF), grants 2569/21 and 3276/25.

References

[AL26] Binyamin Alony and Itamar Levi. Optimal 5-bit S-boxes dataset. SelecSys Labs, Faculty of Engineering, Bar-Ilan University, 2026. https://github.com/Selecsys-Lab/optimal-5bit-sboxes.

[BBK+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In *Cryptographic Hardware and Embedded Systems — CHES 2013*, volume 8086 of *LNCS*, pages 142–158. Springer, 2013. doi:10.1007/978-3-642-40349-1_9.

[BBŞ17] Dušan Božilov, Begül Bilgin, and Hacı Ali Şahin. A note on 5-bit quadratic permutations' classification. *IACR Transactions on Symmetric Cryptology*, 2017(1):398–404, 2017. doi:10.46586/tosc.v2017.i1.398-404.

[BC18] Christina Boura and Anne Canteaut. On the boomerang uniformity of cryptographic Sboxes. *IACR Transactions on Symmetric Cryptology*, 2018(3):290–310, 2018. doi:10.46586/tosc.v2018.i3.290-310.

[BCC10] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential properties of power functions. *International Journal of Information and Coding Theory*, 1(2):149–170, 2010. doi:10.1504/IJICOT.2010.032132.

[BCDC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and Luffa. In *Fast Software Encryption — FSE 2011*, volume 6733 of *LNCS*, pages 252–269. Springer, 2011. doi:10.1007/978-3-642-21702-9_15.

[BDCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 33–50. Springer, 2003. doi:10.1007/3-540-39200-9_3.

[BDD+23] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. BipBip: A low-latency tweakable block cipher with small dimensions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):326–368, 2023. doi:10.46586/tches.v2023.i1.326-368.

[BDM26a] Marie Bolzer, Sébastien Duval, and Marine Minier. A new tool to find lightweight (AND, XOR) implementations of quadratic vectorial Boolean functions up to dimension 9. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 73(1):478–491, 2026. doi:10.1109/TCSI.2025.3602151.

[BDM26b] Marie Bolzer, Sébastien Duval, and Marine Minier. A tool for lightweight (AND, XOR) implementations of large-degree S-boxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):605–629, 2026. doi:10.46586/tches.v2026.i2.605-629.

[BDMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low AND depth and efficient inverses: A guide on S-boxes for low-latency masking. *IACR Transactions on Symmetric Cryptology*, 2020(1):144–184, 2020. doi:10.46586/tosc.v2020.i1.144-184.

[BDMW10] K. A. Browning, J. F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN permutation in dimension six. In *Finite Fields: Theory and Applications (FQ9)*, volume 518 of *Contemporary Mathematics*, pages 33–42. American Mathematical Society, 2010. doi:10.1090/conm/518/10194.

[BDPVA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0. Keccak Team, 2011. https://keccak.team/files/Keccak-reference-3.0.pdf.

[BFP16] Joan Boyar, Magnus Gausdal Find, and René Peralta. On various nonlinearity measures for Boolean functions. *Cryptography and Communications*, 8(3):313–330, 2016. doi:10.1007/s12095-015-0150-9.

[BKRS22] Lilya Budaghyan, Nikolay S. Kaleyski, Constanza Riera, and Pantelimon Stănică. On the behavior of some APN permutations under swapping points. *Cryptography and Communications*, 14(2):319–345, 2022. doi:10.1007/s12095-021-00520-z.

[BM10] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification, 22nd International Conference (CAV 2010)*, volume 6174 of *LNCS*, pages 24–40. Springer, 2010. doi:10.1007/978-3-642-14295-6_5.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *Experimental Algorithms, 9th International Symposium (SEA 2010)*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010. doi:10.1007/978-3-642-13193-6_16.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES S-box. In *Information Security and Privacy Research (SEC 2012)*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012. doi:10.1007/978-3-642-30436-1_24.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-engineering the S-box of Streebog, Kuznyechik and STRIBOBr1. In *Advances in Cryptology — EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 372–402. Springer, 2016. doi:10.1007/978-3-662-49890-3_15.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991. doi:10.1007/BF00630563.

[Can05] David Canright. A very compact S-box for AES. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005. doi:10.1007/11545262_32.

[Car21] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021. doi:10.1017/9781108606806.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998. doi:10.1023/A:1008344232130.

[CDL16] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of lightweight S-boxes using Feistel and MISTY structures. In *Selected Areas in Cryptography — SAC 2015*, volume 9566 of *LNCS*, pages 373–393. Springer, 2016. doi:10.1007/978-3-319-31301-6_22.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021. doi:10.1109/TC.2020.3022979.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In *Advances in Cryptology — EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, 2018. doi:10.1007/978-3-319-78375-8_22.

[CV95] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *LNCS*, pages 356–365. Springer, 1995. doi:10.1007/BFb0053450.

[DCBG+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In *Constructive Side-Channel Analysis and Secure Design — COSADE 2017*, volume 10348 of *LNCS*, pages 1–18. Springer, 2017. doi:10.1007/978-3-319-64647-3_1.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9.

[dMB08] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

[DMB19] Lauren De Meyer and Begül Bilgin. Classification of balanced quadratic functions. *IACR Transactions on Symmetric Cryptology*, 2019(2):169–192, 2019. doi:10.46586/tosc.v2019.i2.169-192.

[DR01] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In *Cryptography and Coding*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001. doi:10.1007/3-540-45325-3_20.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009. doi:10.1007/978-3-642-01001-9_16.

[FGMDP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, 2018. doi:10.46586/tches.v2018.i3.89-120.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security (TIS '16)*, page 3. ACM, 2016. doi:10.1145/2996366.2996426.

[GR16] Dahmun Goudarzi and Matthieu Rivain. On the multiplicative complexity of Boolean functions and bitsliced higher-order masking. In *Cryptographic Hardware and Embedded Systems — CHES 2016*, volume 9813 of *LNCS*, pages 457–478. Springer, 2016. doi:10.1007/978-3-662-53140-2_22.

[HB25] Vedad Hadžić and Roderick Bloem. Efficient and composable masked AES S-box designs using optimized inverters. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):656–683, 2025. doi:10.46586/tches.v2025.i1.656-683.

[JBK24] Yongjin Jeon, Seungjun Baek, and Jongsung Kim. Toward finding S-box circuits with optimal multiplicative complexity. *IEEE Transactions on Computers*, 73(8):2036–2050, 2024. doi:10.1109/TC.2024.3398507.

[JBKK25] Yongjin Jeon, Seungjun Baek, Giyoon Kim, and Jongsung Kim. A framework for generating S-box circuits with Boyar–Peralta algorithm-based heuristics, and its applications to AES, SNOW3G, and Saturnin. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):586–631, 2025. doi:10.46586/tches.v2025.i1.586-631.

[JCL+25] Chenhao Jia, Tingting Cui, Qing Ling, Yan He, Kai Hu, Yu Sun, and Meiqin Wang. How small can S-boxes be? *IACR Transactions on Symmetric Cryptology*, 2025(1):592–622, 2025. Code: https://github.com/ChenhaoJia/Area-Optimized-Implementation-for-S-box. doi:10.46586/tosc.v2025.i1.592-622.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In *Fast Software Encryption — FSE 1997*, volume 1267 of *LNCS*, pages 28–40. Springer, 1997. doi:10.1007/BFb0052332.

[Kal21] Nikolay S. Kaleyski. Invariants for EA- and CCZ-equivalence of APN and AB functions. *Cryptography and Communications*, 13(6):995–1023, 2021. doi:10.1007/s12095-021-00541-8.

[KBJ+26] Giyoon Kim, Seungjun Baek, Yongjin Jeon, Vedad Hadžić, and Jongsung Kim. Bitsliced segment-based search technique for low-depth and hardware-efficient S-box circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):1152–1176, 2026. Code: https://github.com/sboxcircuit/slice. doi:10.46586/tches.v2026.i3.1152-1176.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS 2022)*, pages 1799–1812. ACM, 2022. doi:10.1145/3548606.3559362.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In *Fast Software Encryption — FSE 1994*, volume 1008 of *LNCS*, pages 196–211. Springer, 1995. doi:10.1007/3-540-60590-8_16.

[KPF+24] Alexandr Kuznetsov, Nikolay Poluyanenko, Emanuele Frontoni, Sergey Kandiy, and Olha Pieshkova. Optimized simulated annealing for efficient generation of highly nonlinear S-boxes. *Soft Computing*, 28(5):3905–3920, 2024. doi:10.1007/s00500-023-09334-y.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER — statistical independence and leakage verification. In *Advances in Cryptology — ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 787–816. Springer, 2020. doi:10.1007/978-3-030-64837-4_26.

[Kuh55] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi:10.1002/nav.3800020109.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. In *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233. Springer, 1994. doi:10.1007/978-1-4615-2694-0_23.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations: And an explanation based on externally-amplified couplings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):293–317, 2019. doi:10.46586/tches.v2019.i2.293-317.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers: Engineering an ultra-low-latency cipher from gate level for secure processor architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):510–545, 2021. doi:10.46586/tches.v2021.i4.510-545.

[LP07] Gregor Leander and Axel Poschmann. On the classification of 4-bit S-boxes. In *Arithmetic of Finite Fields (WAIFI 2007)*, volume 4547 of *LNCS*, pages 159–176. Springer, 2007. doi:10.1007/978-3-540-73074-3_13.

[LW14] Yongqiang Li and Mingsheng Wang. Constructing S-boxes for lightweight cryptography with Feistel structure. In *Cryptographic Hardware and Embedded Systems — CHES 2014*, volume 8731 of *LNCS*, pages 127–146. Springer, 2014. doi:10.1007/978-3-662-44709-3_8.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1994. doi:10.1007/3-540-48285-7_33.

[MC15] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015. doi:10.48550/arXiv.1504.04909.

[ME19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES SBoxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):91–125, 2019. doi:10.46586/tches.v2019.i4.91-125.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD — a probing-based hardware leakage detection tool. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):311–348, 2022. doi:10.46586/tches.v2022.i4.311-348.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited: or why proofs in the robust probing model are needed. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):256–292, 2019. doi:10.46586/tches.v2019.i2.256-292.

[Nan11] Nangate Inc. NanGate FreePDK45 open cell library. OpenROAD-flow-scripts distribution of NangateOpenCellLibrary_typical.lib, 2011. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts.

[Nat23] National Institute of Standards and Technology. Advanced encryption standard (AES). Technical Report FIPS 197, NIST, 2023. doi:10.6028/NIST.FIPS.197-upd1.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security (ICICS 2006)*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006. doi:10.1007/11935308_38.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *LNCS*, pages 55–64. Springer, 1994. doi:10.1007/3-540-48285-7_6.

[Paa97] Christof Paar. Optimized arithmetic for Reed-Solomon encoders. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT 1997)*, page 250. IEEE, 1997. doi:10.1109/ISIT.1997.613165.

[PČR16] Stjepan Picek, Marko Čupić, and Leon Rotim. A new cost function for evolution of S-boxes. *Evolutionary Computation*, 24(4):695–718, 2016. doi:10.1162/EVCO_a_00191.

[Ras22] Shahram Rasoolzadeh. Low-latency Boolean functions and bijective S-boxes. *IACR Transactions on Symmetric Cryptology*, 2022(3):403–447, 2022. doi:10.46586/tosc.v2022.i3.403-447.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949. doi:10.1002/j.1538-7305.1949.tb00928.x.

[SMTM01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In *Advances in Cryptology — ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 239–254. Springer, 2001. doi:10.1007/3-540-45682-1_15.

[Sto16] Ko Stoffelen. Optimizing S-box implementations for several criteria using SAT solvers. In *Fast Software Encryption — FSE 2016*, volume 9783 of *LNCS*, pages 140–160. Springer, 2016. doi:10.1007/978-3-662-52993-5_8.

[SYY+02] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. The block cipher SC2000. In *Fast Software Encryption — FSE 2001*, volume 2355 of *LNCS*, pages 312–327. Springer, 2002. doi:10.1007/3-540-45473-X_26.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *Advances in Cryptology — EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015. doi:10.1007/978-3-662-46800-5_12.

[TSADM19] Eleonora Testa, Mathias Soeken, Luca Gaetano Amarù, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19)*, pages 74:1–74:6. ACM, 2019. doi:10.1145/3316781.3317893.

[WG13] Claire Wolf and Johann Glaser. Yosys — a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip 2013)*, 2013. https://yosyshq.net/yosys/files/yosys-austrochip2013.pdf.

Supplementary Material (SM)

Yoshan: Algebraic-Circuit Co-Synthesis and Search Methodology of Hardware-Efficient S-Boxes

This Supplementary Material (SM) is organized into three sections: **SM A** details the masked AND gadgets, PINI composability, and differentiable Piling-Up/LSAP hyper-parameters; **SM B** provides the campaign configurations, reference catalogue, archive breakdown across all 56 profiles, S^{-1} silicon synthesis, provenance table, AND-erasure family, and algebraic inverse derivations; and **SM C** prints the complete hexadecimal lookup tables and verified gate-level netlists for all 30 forward (S) and 30 inverse (S^{-1}) benchmark S-boxes.

A Methodology Details

A.1 Masked AND Gadget, Composability, and Mitigation Costs

Figure 3 shows the first-order ($t = 1, d = 2$) domain-oriented masking (`DOM-indep`) AND gadget [GMK16] synthesized in Section 5.2. The inner-domain products $a_0 \wedge b_0$ and $a_1 \wedge b_1$ need no randomness, while the cross-domain products $a_0 \wedge b_1$ and $a_1 \wedge b_0$ are blinded with one fresh random bit r and registered in four flip-flops (`DFF_X1`) before each domain compresses its two terms into an output share, preventing glitches from combining shares of different domains. When the two inputs (a, b) of an AND gate share a common variable in their intra-stage combinational fan-in cone ($\mathrm{Supp}_{\mathrm{st}\ell}(a) \cap \mathrm{Supp}_{\mathrm{st}\ell}(b) \neq \emptyset$), a glitch propagating through both linear cones into $a_0 \wedge b_1$ can observe both shares of that common variable. Three mitigations exist; we state their $d = 2$ per-gadget costs on the NanGate cells of Section 5.2 and their analytical effect on our AES circuits: (i) Resharing one operand (`DOM-dep` [GMK16]) costs +1 random bit, +2 `XOR2_X1`, and +2 `DFF_X1` per reconvergent gate and inserts one register cut on that path (a two-cycle multiplication [GMK16, §3.3]), which in a synchronous pipeline also requires 2 `DFF_X1` per parallel stage wire to align the stage; moreover, DOM gadgets carry no general multi-stage PINI composability guarantee [CGLS21, MMSS19]. (ii) The glitch-robust PINI `HPC2` gadget [CGLS21, Alg. 5] keeps one random bit per gate (N_{AND} in total) at a two-cycle latency ($2D$ pipeline stages, doubling the synchronization registers of Eq. (8) to $4N_{\mathrm{sync}}$): at $d = 2$ it uses $2d^2 - d = 6$ `AND2_X1`, $d = 2$ `INV_X1`, $2d(d-1) = 4$ `XOR2_X1` (or 6 `XOR2_X1` when counting explicit fresh-mask blinding on both cross-domain paths), and 8 `DFF_X1` per gate (or 10 `DFF_X1` when both operands a and b are independently latched in the first cycle rather than sharing the stage-input synchronization registers). Under composable $d = 2$ `HPC2`, moving from our $D = 4$ FIPS-197 AES circuit (36/99/4, $N_{\mathrm{sync}} = 69$) to our $D = 3$ AES-class circuit (42/101/4, $N_{\mathrm{sync}} = 24$) reduces pipeline latency from 8 to 6 cycles, synchronization flip-flops from $4 \times 69 = 276$ to $4 \times 24 = 96$, and total flip-flops from $360 + 276 = 636$ to $420 + 96 = 516$ under the 10-FF, 6-XOR convention (3,802.24 vs. 3,341.00 μm^2, saving 120 `DFF_X1` and 12.1% area) or from $288 + 276 = 564$ to $336 + 96 = 432$ under the 8-FF, 6-XOR convention (3,476.66 vs. 2,961.15 μm^2, deducting $2N_{\mathrm{AND}}$ flip-flops and saving 132 `DFF_X1` and 14.8% area; under the 4-XOR, 8-FF convention the totals are 3,361.74 vs. 2,827.09 μm^2, saving 15.9% area). (iii) Single-cycle PINI gadgets—`HPC3` [KM22] (4 `AND2_X1`, 6 `XOR2_X1`, 4 `DFF_X1`) and Hadžić and Bloem's `HPC3.1` [HB25] (4 `AND2_X1`, 4 `XOR2_X1`, 4 `DFF_X1`)—restore latency D at 2 random bits per gate ($2N_{\mathrm{AND}}$ in total) with the exact register count $N_{\mathrm{DFF}} = 4N_{\mathrm{AND}} + 2N_{\mathrm{sync}}$ of Eq. (8): our $D = 3$ AES circuit then uses 216 `DFF_X1` at 3 cycles (1,748.15 μm^2 with `HPC3.1`, 1,882.21 μm^2 with `HPC3`) versus

282 `DFF_X1` at 4 cycles for $D = 4$ FIPS-197 AES (1,976.38 and 2,091.29 μm^2), saving 66 `DFF_X1` and 11.5%/10.0% area. Thus the register and latency advantage of reducing D from 4 to 3 holds unchanged under composable PINI masking.

A.2 Differentiable Template Module: Terms and Hyper-Parameters

This subsection lists the exact definitions behind Eq. (5) (Section 4.2). *Parameterization.* Round wire permutations are parameterized by $P^{(r)} = \mathrm{softmax}(W^{(r)}_{\mathrm{perm}}/\tau)$ and branch polynomials by temperature-sharpened sigmoids of weights $W^{(r)}_{\mathrm{poly}}$; the relaxed Walsh spectrum $\widetilde{W}_S(\alpha, v) = H_{2^n}(1 - 2\widetilde{S}_v(x))$ is evaluated for every relaxed component $\widetilde{S}_v$ ($v \neq 0$) on all 2^n inputs. After gradient descent, the weights are projected to a discrete circuit: one LSAP per round permutation turns the row-stochastic softmax matrix into a permutation matrix (a vertex of the Birkhoff polytope), and K sequential masked rectangular LSAP passes select the monomials of each branch polynomial. *Templates.* The standalone command-line interface (CLI) supports three 12-round (or 3–4-round) templates at $n = 7$, while the distributed hunter uses the width-generic 3-round Misty template for all $n \in \{5, 6, 7, 8\}$. At $n = 7$, the Keccak-like SPN template uses the fixed invertible 7×7 circulant diffusion matrix $M_{\mathrm{diff}} = \mathrm{circ}(1, 1, 1, 1, 1, 0, 0) \in \mathrm{GL}(7, \mathbb{F}_2)$ (no M_{diff} is used for $n \in \{5, 6, 8\}$, which run the Misty template). At $n = 7$, the unbalanced Feistel template uses a $(4, 3)$ branch split ($L \in \mathbb{F}_2^4, R \in \mathbb{F}_2^3$): in each round, $R = (R_0, R_1, R_2)$ is expanded cyclically to $R_{\mathrm{exp}} = (R_0, R_1, R_2, R_0)$, passed through a 4-bit quadratic Keccak-χ layer $F_i(R_{\mathrm{exp}}) = R_{\mathrm{exp},i} \oplus (\overline{R_{\mathrm{exp},i+1 \bmod 4}} \wedge R_{\mathrm{exp},i+2 \bmod 4})$ to update $L \leftarrow L \oplus F(R_{\mathrm{exp}})$, and followed by the learned 7×7 wire permutation $P^{(r)}$. The Misty template uses $(n - 3, 3)$ (standalone $n = 7$) or balanced $(\lceil n/2 \rceil, \lfloor n/2 \rfloor)$ (in-hunter $n \in \{5, 6, 7, 8\}$) branch splits. Branch polynomials are parameterized by temperature-sharpened sigmoids $\sigma\big(\kappa(\tau)(\sigma(W^{(r)}_{\mathrm{poly}}) - \frac{1}{2})\big)$ with $\kappa(\tau) = \kappa_0 + \kappa_1 \max(0, 1 - 2\tau)$, $(\kappa_0, \kappa_1) = (5, 25)$ for the SPN and $(4, 16)$ for the Misty template. The relaxed component for an output mask $v \neq 0$ is $\widetilde{S}_v(x) = \widetilde{\mathrm{XOR}}\big(\{\widetilde{S}_j(x) : v_j = 1\}\big)$. *Loss terms.* $\mathcal{L}_{\mathrm{SAC}} = \sum_{i,k} \big(2^{-n} \sum_x \widetilde{\mathrm{XOR}}(\widetilde{S}_k(x), \widetilde{S}_k(x \oplus e_i)) - \frac{1}{2}\big)^2$; $\mathcal{L}_{\mathrm{sym}} = \sum_r \sum_i (P^{(r)}_{ii})^2$ (SPN template only; it penalizes fixed points of the round wire permutation); $\mathcal{L}_{\mathrm{ent}} = c_{\mathrm{ent}} \cdot \mathrm{mean}_{x,k}\, \widetilde{S}_k(x)(1 - \widetilde{S}_k(x))$ with $c_{\mathrm{ent}} = 300$ for the SPN and 25 for the Misty template; $\mathcal{L}_{\ell_1} = c \sum \sigma(W_{\mathrm{poly}})$ with $c = 3 \cdot 10^{-3}$ for the SPN and $3 \cdot 10^{-4}$ for the Misty template; and $\lambda(\tau) = \max(0.01, 1 - 2\tau)$. *Schedule and runtime.* $\tau_0 = 0.5$ and $\tau_{t+1} = \max(0.01, \rho\, \tau_t)$ with $\rho = 0.02^{1/(0.9E)}$ over E epochs; Adam with a cosine-decayed learning rate of 0.12 for the SPN/Feistel and $2 \cdot 10^{-3}$ for the Misty template, gradient norm clipped at 5. The standalone CLI runs $E = 1{,}200$ epochs (and 100 epochs for standalone homotopy crossover $W_{\mathrm{child}} = \alpha W_A + (1 - \alpha) W_B$). The in-hunter Misty seeder runs $E = 20$ ($n \in \{5, 6\}$) or $E = 16$ ($n \in \{7, 8\}$) Simultaneous Perturbation Stochastic Approximation (SPSA) epochs on 3 rounds, evaluating the spectral term over coordinate and adjacent-pair components with exponent 8 instead of 32; in-hunter homotopy crossover blends logits with $\alpha \sim \mathrm{Uniform}(0.3, 0.7)$ and runs $E_{\mathrm{cross}} = 4$ ($n \leq 6$) or 2 ($n \geq 7$) SPSA epochs. On one CPU thread, one in-hunter seed takes 0.18 s ($n = 5$), 0.21 s ($n = 6$), 0.48 s ($n = 7$), and 0.75 s ($n = 8$), and one crossover call takes 0.02–0.22 s. *Projection.* Each round permutation is obtained by one LSAP on the row-stochastic softmax matrix; each branch polynomial by K sequential rectangular LSAP passes on its sigmoid weight matrix (candidate monomials $\times$ branch outputs), where the entries selected in a pass are masked with -10^9 before the next pass so that every pass adds one disjoint assignment ($K = 5$; $K = 2$ for Misty branches; bijectivity is structural for additive $L \leftarrow L \oplus F(R)$ branch updates and checked at runtime by Stage 3 for SPN projections).

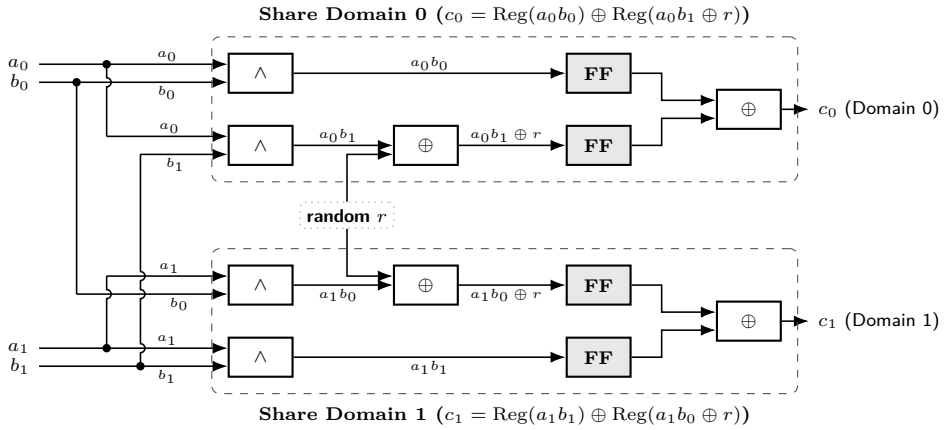


Figure 3: First-order ($t = 1$, $d = 2$) DOM-indep AND gadget [GMK16]. The cross-domain products $a_0 \wedge b_1$ and $a_1 \wedge b_0$ are blinded with a fresh random bit r, and all four products are stored in registers (FF) before each domain compresses its two terms into an output share.

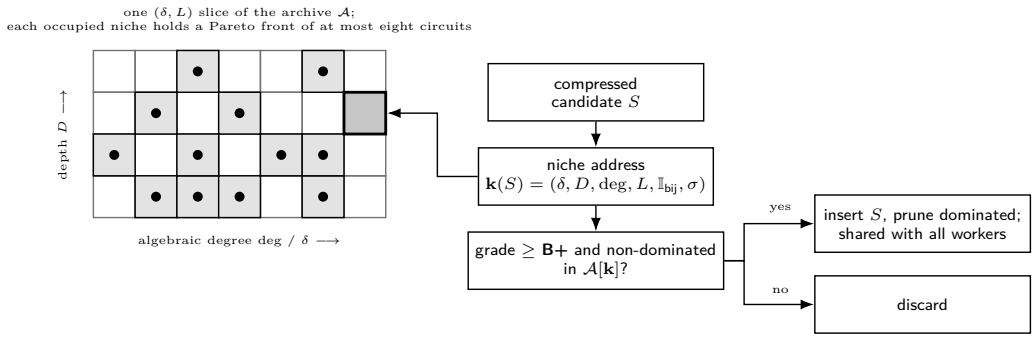


Figure 4: The quality-diversity archive (Section 4.4). A candidate is addressed to its niche by Eq. (6) and kept if it has grade **B+** or better and is not Pareto-dominated there.

B Campaign Suite, Archive Breakdown, and Reference Tables

This appendix lists the campaign configurations and targets (Table 5), the reference catalogue of Section 6.1 (Table 6), the archive summary (Table 7) and its profile-by-profile breakdown (Table 11), the synthesis of the 30 inverse circuits (Table 9), the provenance of the 30 benchmark circuits (Table 10), the AND-erasure family of $\mathsf{YSH}_{8,2}$ (Table 8), and the derivations behind the inverse circuits of Section 6.3.

B.1 Multi-Orbit Invariant Collisions and Catalogue Breakdown

Several profiles of Table 11 contain multiple reference classes sharing the coarse profile $(n, \delta, L, \deg)$ or even identical forward invariants; the exact affine-equivalence solver separates all of them with zero unresolved entries (Category U = 0). For $n = 5$, the quadratic APN cosets x^3 and x^5 in $(2, 8, 2)$ split the 85 archive permutations into 72 in the x^3 class ($1 = x^3$, $71 \sim_{\mathrm{aff}} x^3$) and 13 in the x^5 class (the printed Gold circuit of SM C.5 is the table of x^5 itself and the SAC-ideal circuit of SM C.6 is one of the 12 affine-equivalent tables), and the inverse-Gold cosets x^7 ($x^{25} = (x^5)^{-1}$) and x^{11} ($x^{21} = (x^3)^{-1}$) in $(2, 8, 3)$ split the

Table 5: The eight campaigns of the suite. Each width n has one campaign aiming at near-optimal profiles and one aiming at relaxed or lightweight profiles. All campaigns use a budget of 30,000 steps and a base patience $P = 3{,}500$; ρ_g is the probability of choosing a library seed when the episode does not start from an archive elite (Section 4.1). The target (δ, L) steers Stage 2 only: it decides when an episode switches from cryptographic to circuit-shrinking mutations and when the lexicographic order flips to the cost score (in `optimal` mode the annealing band is additionally relaxed to $(\delta + 2, 2L)$); admission to the archive is independent of the campaign and always governed by the grade gate of Table 1, so a candidate that only reaches the steering target but misses grade **B+** is never archived.

Campaign	n	Mode	Target (δ, L)	ρ_g
`c01_5b_optimal`	5	optimal	$(2, 12)$	0.7
`c02_5b_relaxed`	5	relaxed	$(16, 24)$	0.6
`c03_6b_optimal`	6	optimal	$(4, 32)$	0.7
`c04_6b_relaxed`	6	relaxed	$(16, 48)$	0.6
`c05_7b_apn_optimal`	7	optimal	$(6, 40)$	0.7
`c06_7b_lightweight`	7	lightweight	$(8, 36)$	0.6
`c07_8b_aes_optimal`	8	optimal	$(4, 32)$	0.7
`c08_8b_lightweight`	8	lightweight	$(16, 80)$	0.6

160 permutations into 133 in the x^7 class (including one $= x^7$ and our 14-AND circuit of SM C.8) and 27 $\sim_{\text{aff}} x^{11}$. In the 5-bit $(4, 12, 4)$ profile (828 distinct permutations), 690 permutations are certified Category A across the canonical two-point-swapped power-map orbits (317 in the four orbits of x^3_{swap}, including 145 $\sim_{\text{aff}} x^3_{\text{swap}(1,3)}$ containing SM C.1; 348 in the four forward and four inverse orbits of x^5_{swap}; and 25 in $x^{15}_{\text{swap}(0,1)}$, including one exact match), while 138 are certified Category B. In $(4, 16, 2)$, all 7 archive permutations are Category A (5 $\sim_{\text{aff}} Q_{71}$ including SM C.2 and 2 $\sim_{\text{aff}} Q_{72}$ [BBŞ17]); in $(4, 16, 4)$ (597 permutations), 199 belong to the three non-zero swap orbits of $x^{15}_{\text{swap}(1,w)}$ (108 for $w = 2$, 43 for $w = 3$ including one exact match, 48 for $w = 6$) and 398 are Category B; and in $(8, 16, 2)$ (8 permutations), 6 $\sim_{\text{aff}} \chi_5$, 1 $\sim_{\text{aff}} Q_{53}$, and 1 $\sim_{\text{aff}} Q_{59}$. For $n = 6$, the 4 Category A permutations in $(4, 16, 3)$ split into 2 $\sim_{\text{aff}} x^{13}$ (Q^{-1}_{2258}) and 2 $\sim_{\text{aff}} \text{Feistel}_{3\text{R}}(\chi_3)$ (including our 9-AND cubic involution of SM C.17). For $n = 7$, the three Gold cosets x^3, x^5, x^9 in $(2, 16, 2)$ split the 15 permutations into 6, 6, and 3; the Kasami and Welch cosets x^{13} and x^{11} in $(2, 16, 3)$ split the 151 permutations into 76 in the x^{13} class and 75 in the x^{11} class; the five quartic AB cosets $x^{15}, x^{23}, x^{27}, x^{29}, x^{43}$ in $(2, 16, 4)$ split the 581 permutations into 436 $\sim_{\text{aff}} x^{29}$, 144 $\sim_{\text{aff}} x^{23}$, and one $\sim_{\text{aff}} x^{43}$ (the x^{85} inverse of the Gold x^3 circuit); in $(4, 20, 6)$, all 3 archive permutations are Category A (1 $\sim_{\text{aff}} x^5_{\text{swap}(1,25)}$ of SM C.21, 1 $\sim_{\text{aff}} (x^5_{\text{swap}(1,25)})^{-1}$, and 1 $\sim_{\text{aff}} x^{13}_{\text{swap}(1,2)}$); and in $(4, 24, 6)$, both archive permutations are Category A (1 $\sim_{\text{aff}} x^{63}_{\text{swap}(1,2)}$ and 1 $\sim_{\text{aff}} x^{63}_{\text{swap}(1,3)}$). Over the whole archive, all 2,681 Category A permutations are certified: 15 equal a catalogue lookup table ('=') and 2,666 carry an exhaustively verified affine-equivalence witness ('$\sim_{\text{aff}}$').

B.2 Inverse Derivations

This subsection collects the algebraic identities behind the Tier-2, Tier-3, and Tier-4 inverse circuits of Section 6.3.

Tier 2: shear and transposition extensions of a lower-degree core. For $\mathsf{YSH}_{6,6}$, $\mathsf{YSH}_{6,4}$, and $\mathsf{YSH}_{6,5}$, $S(x) = T(Q(x))$ where $Q \sim_{\text{aff}} x^5$ (Q_{2258}) and T applies one, two, or three chained involution shears along constant directions in $\mathbb{F}_2^6$. All three inverses reuse the 18-AND, $D = 2$ tower-field norm circuit for $Q^{-1}(y)$ below and add the shear correction in

Table 6: The reference catalogue of Section 6.1 (712 unique verified lookup tables across $n \in \{5,6,7,8\}$; machine-readable tables, SHA-256 digests, and provenance in `reference_catalogue/reference_catalogue.json`). Power maps x^d ($x \mapsto x^d$ over $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p_n(x))$ with $0 \mapsto 0$) are listed one representative per cyclotomic coset $\{d \cdot 2^k \bmod (2^n-1)\}$ with $\gcd(d, 2^n-1) = 1$; two-point-swapped power permutations $x^d_{\mathrm{swap}(u,v)}$ cover all canonical Frobenius and inversion linear-equivalence orbits [BCC10, BKRS22]. Every non-involution entry is accompanied by its compositional inverse S^{-1}.

n	$p_n(x)$	LUTs	Catalogue families and representatives
5	x^5+x^2+1 (`0x25`)	247	Power cosets x^1, x^3, x^5 (Gold APN), x^7, x^{11} (inv. Gold), x^{15} (inversion); all 75 quadratic affine classes $Q_1, \ldots, Q_{75}$ and inverses $Q_1^{-1}, \ldots, Q_{75}^{-1}$ [BBŞ17]; all 17 strong quartic classes of [DMB19, Table 7] and inverses; all 4 canonical linear swap orbits $\{(0,1),(1,2),(1,3),(1,6)\}$ of $x^3, x^5, x^7, x^{11}, x^{15}$ and inverses [BKRS22]; Keccak $\chi_5^{\pm 1}$ [BDPVA11], Ascon$^{\pm 1}$ [DEMS21], Fides-5$^{\pm 1}$ [BBK+13], SC2000-5$^{\pm 1}$ [SYY+02], DryGASCON128$^{\pm 1}$, Shamash$^{\pm 1}$
6	x^6+x+1 (`0x43`)	79	Power cosets x^1, x^5 (Q_{2258}), x^{11}, x^{13} (Q_{2258}^{-1}), x^{23}, x^{31} (inversion); all 8 optimal $(\delta, L) = (4, 16)$ quadratic classes $Q_{2256}, \ldots, Q_{2263}$ and cubic inverses [DMB19, Table 15] (plus degree-separation certificate covering the remaining 2,255 quadratic classes of [DMB19]); 3-round Feistel/MISTY structures Feistel$_{3R}(\chi_3)$, Feistel$_{3R}(x^3)$, MISTY$_{3R}(\chi_3^{\pm 1})$, MISTY$_{3R}(x^3, x^5)$, Open Butterfly H_α [CDL16, BPU16, BDMD+20]; all 8 canonical linear swap orbits of x^5, x^{31} and inverses; SPEEDY$^{\pm 1}$ [LMMR21], Dillon APN$^{\pm 1}$ [BDMW10], SC2000-6$^{\pm 1}$ [SYY+02], Fides-6$^{\pm 1}$ [BBK+13], BipBip-6$^{\pm 1}$ [BDD+23]
7	x^7+x+1 (`0x83`)	178	All 18 bijective power cosets x^1, x^3, x^5 (Gold), x^7, x^9 (Gold), x^{11} (Welch), x^{13} (Kasami), $x^{15}, x^{19}, x^{21}, x^{23}, x^{27}, x^{29}, x^{31}, x^{43}, x^{47}, x^{55}, x^{63}$ (inversion); all 10 canonical linear swap orbits $\{(0,1),(1,2),(1,3),(1,8),(1,12),(1,13),(1,14),(1,24),(1,25),(1,28)\}$ of $x^3, x^5, x^9, x^{13}, x^7, x^{11}, x^{63}$ and inverses [BCC10, BKRS22]
8	$x^8+x^4+x^3+x+1$ (`0x11B`)	208	All 16 bijective power cosets $x^1, x^7, x^{11}, x^{13}, x^{19}, x^{23}, x^{29}, x^{31}, x^{37}, x^{43}, x^{47}, x^{53}, x^{59}, x^{61}, x^{91}, x^{127}$ (inversion); all 20 canonical linear swap orbits of x^{127}, x^{254} and two-point swaps of $x^7, x^{11}, x^{13}, x^{19}, x^{23}, x^{29}, x^{31}, x^{37}, x^{43}, x^{47}, x^{53}, x^{59}, x^{61}, x^{91}$ [BCC10, BKRS22]; 8-bit 3-round Feistel$_{4,4,4}$, MISTY$_{4,4,4}$, Bridge$_{4,4,4}$, and unbalanced MISTY$_{5,3,5}$ [CDL16, BDMD+20]; AES$^{\pm 1}$ [Nat23], Skinny-8$^{\pm 1}$

Table 7: Archive summary across $n \in \{5,6,7,8\}$. Netlists: distinct certified netlists (every entry passed exhaustive simulation and the Z3 miter of Section 7); Distinct: distinct permutations; Profiles: distinct $(n, \delta, L, \deg)$; Cat. A/B: distinct permutations per category against the 712-entry catalogue (Section 6.1); **A+**/**A**/**B+**: netlists per grade of Table 1; Pareto: distinct (AND, XOR, D) triples non-dominated (componentwise) among the netlists of the same profile, summed over the profiles of Table 11 (which lists only min-AND and min-D representatives).

n	Netlists	Distinct	Profiles	Cat. A	Cat. B	**A+**	**A**	**B+**	Pareto
5	2,390	2,209	12	1,300	909	295	1,006	1,089	18
6	23,956	23,891	12	474	23,417	466	28	23,462	19
7	976	895	14	870	25	775	153	48	17
8	4,015	3,991	18	37	3,954	50	9	3,956	50
All	31,337	30,986	56	2,681	28,305	1,586	1,196	28,555	104

Table 8: Stage-2 AND erasure applied to $\mathsf{YSH}_{8,2}$ (Section 6.5): the certified family of 8-bit degree-7 permutations it produces, after the Stage-4 pass, ordered by AND count; its 18-AND member is printed in full as $\mathsf{YSH}_{8,3}$ (SM C.28). All members passed exhaustive simulation and the Z3 miter; β, FP, and SAC were recomputed from the lookup tables.

A/X/N	D	δ	L	$\mathcal{D}(S)$	β	FP	SAC
28/61/1	6	10	72	$[6^3, 7^{252}]$	22	0	.1094
27/62/1	6	12	72	$[6^3, 7^{252}]$	22	0	.1094
26/63/1	6	12	72	$[6^7, 7^{248}]$	22	0	.1094
25/63/1	6	16	72	$[6^7, 7^{248}]$	24	1	.1875
24/64/1	6	12	72	$[6^3, 7^{252}]$	32	2	.125
23/65/1	6	14	72	$[6^{15}, 7^{240}]$	36	1	.125
22/66/2	6	12	76	$[6^{15}, 7^{240}]$	36	3	.125
21/67/2	6	14	76	$[6^{31}, 7^{224}]$	36	2	.125
20/68/2	6	14	76	$[6^{31}, 7^{224}]$	36	3	.125
19/68/2	6	14	72	$[4^1, 6^{62}, 7^{192}]$	32	2	.125
18/71/2	5	14	72	$[4^1, 5^2, 6^{60}, 7^{192}]$	38	0	.1094

Table 9: Gate-exact Yosys synthesis and ABC NLDM timing of the 30 inverse (S^{-1}) circuits of Section 6.3 on NanGate 45 nm (1 GE = 0.798 μm^2, GE in SM C). $\mathcal{D}(S^{-1})$: degree spectrum; n/T: width/Tier (Section 6.3); FF/S: total/sync `DFF_X1` ($G = 4R$ gadget flip-flops; cf. Table 3); N_{reconv} as in Table 3.

			Unmasked (S^{-1})				**Masked ($d = 2$ DOM, S^{-1})**					
S-box (SM)	n/T	$\mathcal{D}(S^{-1})$	A/X/N	LTP	μm^2	Delay (ps)	D/R	A/X/N	FF/S	μm^2	Stage (ps)	N_{reconv}
$(x^3_{\text{swap}(1,3)})^{-1}$ class (C.1)	5/T2	$[3^{15}, 4^{16}]$	13/38/6	12	77.67	623.2	2/13	52/128/6	84/32	642.66	323.9	5
Q_{71}^{-1} class (C.2)	5/T4	$[3^{31}]$	9/32/2	10	61.71	530.1	2/9	36/100/2	68/32	506.46	302.7	2
$\mathsf{YSH}_{5,1}^{-1}$: cubic δ=8 (C.3)	5/T4	$[3^{31}]$	9/22/2	8	45.75	366.8	2/9	36/80/2	62/26	447.41	329.7	0
$\mathsf{YSH}_{5,2}^{-1}$: quartic δ=8 (C.4)	5/T2	$[2^7, 3^8, 4^{16}]$	5/11/2	7	23.94	286.9	2/5	20/42/2	40/20	270.26	306.3	1
Gold x^5 (C.5)	5/T4	$[3^{31}]$	10/37/4	11	71.82	595.9	2/10	40/114/4	70/30	543.17	353.4	2
Gold class, SAC-ideal (C.6)	5/T1	$[3^{31}]$	10/41/10	13	81.40	652.6	2/10	40/122/10	86/46	631.48	346.9	5
χ_5 class (C.7)	5/T4	$[3^{31}]$	9/21/3	8	44.69	329.7	2/9	36/78/3	62/26	444.75	258.1	0
inverse-Gold (C.8)	5/T1	$[2^{31}]$	7/26/5	10	51.60	485.6	1/7	28/80/5	38/10	331.97	358.2	5
x^{15} (C.9)	5/T1	$[4^{31}]$	14/57/0	14	105.87	794.1	2/14	56/170/0	88/32	728.84	517.0	7
$\mathsf{YSH}_{6,1}^{-1}$: quartic δ=4 (C.10)	6/T2	$[3^7, 4^{56}]$	9/27/1	13	53.20	764.6	3/9	36/90/1	78/42	535.19	407.4	3
$\mathsf{YSH}_{6,2}^{-1}$: quartic k=5 (C.11)	6/T2	$[2^{15}, 3^{48}]$	9/37/0	8	68.63	468.6	2/9	36/110/0	76/40	557.54	387.3	4
$\mathsf{YSH}_{6,3}^{-1}$: cubic δ=4 (C.12)	6/T2	$[3^{31}, 4^{32}]$	9/17/8	10	40.96	399.2	3/9	36/70/8	72/36	479.86	317.0	1
$\mathsf{YSH}_{6,4}^{-1}$: deg. 5 δ=8 (C.13)	6/T2	$[3^{15}, 4^{16}, 5^{32}]$	22/49/1	12	102.14	632.8	3/22	88/186/1	132/44	987.92	480.4	2
$\mathsf{YSH}_{6,5}^{-1}$: deg. 5 δ=6 (C.14)	6/T2	$[3^{15}, 4^{16}, 5^{32}]$	21/57/0	15	113.32	869.9	3/21	84/198/0	134/50	1011.33	381.2	7
$\mathsf{YSH}_{6,6}^{-1}$: quartic δ=8 (C.15)	6/T2	$[3^7, 4^{56}]$	27/50/3	10	110.12	561.3	2/27	108/208/3	140/32	1081.56	479.7	2
Q_{2258} class (C.16)	6/T3	$[3^{63}]$	18/37/0	9	78.20	473.2	2/18	72/146/0	96/24	743.74	480.1	2
χ_3-Feistel invol. (C.17)	6/T1	$[3^{63}]$	9/15/9	10	38.30	336.8	3/9	36/66/9	72/36	474.01	254.4	0
inversion (C.18)	6/T1	$[5^{63}]$	21/50/0	14	102.14	771.4	3/21	84/184/0	152/68	1070.38	317.0	3
Dillon APN (C.19)	6/T1	$[3^7, 4^{56}]$	28/78/7	15	158.00	819.8	2/28	112/268/7	168/56	1310.32	423.5	13
$\mathsf{YSH}_{7,1}^{-1}$: quartic δ=4 (C.20)	7/T2	$[4^7, 5^{120}]$	30/158/12	24	290.47	1511.7	3/30	120/436/12	226/106	1851.89	759.6	23
$(x^5_{\text{swap}(1,25)})^{-1}$ class (C.21)	7/T2	$[4^{63}, 6^{64}]$	34/164/9	21	302.71	1263.9	3/34	136/464/9	216/80	1866.79	698.8	28
Gold x^3 (C.22)	7/T3	$[4^{127}]$	29/149/15	22	276.64	1191.7	2/29	116/414/15	166/50	1542.80	772.0	28
Kasami x^{13} (C.23)	7/T3	$[3^{127}]$	37/139/0	17	261.21	1056.7	2/37	148/426/0	204/56	1759.86	735.9	6
inversion (C.24)	7/T1	$[6^{127}]$	42/229/6	33	413.36	2118.5	3/42	168/626/6	306/138	2564.77	716.5	39
power x^7 (C.25)	7/T3	$[5^{127}]$	51/220/15	33	413.36	1904.0	3/51	204/644/15	284/80	2537.11	976.9	27
$\mathsf{YSH}_{8,1}^{-1}$: deg. 7 δ=6 (C.26)	8/T2	$[6^1, 7^{254}]$	43/109/10	24	225.04	1380.5	4/43	172/390/10	264/92	2004.58	578.1	9
$\mathsf{YSH}_{8,2}^{-1}$: deg. 7 δ=10 (C.27)	8/T2	$[6^3, 7^{252}]$	28/84/2	28	164.92	1524.5	6/28	112/280/2	296/184	1905.62	431.9	7
$\mathsf{YSH}_{8,3}^{-1}$: deg. 7 δ=14 (C.28)	8/T2	$[5^1, 6^{62}, 7^{192}]$	18/77/8	28	146.30	1389.2	6/18	72/226/8	274/202	1680.59	438.1	5
AES (D=3) (C.29)	8/T1	$[7^{255}]$	42/106/4	22	215.99	1337.5	3/42	168/380/4	216/48	1764.11	576.4	11
AES (FIPS) (C.30)	8/T1	$[7^{255}]$	36/95/4	23	192.05	1281.9	4/36	144/334/4	276/132	1936.48	462.5	9

parallel via the directional-derivative identity

$$S^{-1}(y) = Q^{-1}(y \oplus s(y)c) = Q^{-1}(y) \oplus \big(s(y) \wedge \Delta_c Q^{-1}(y)\big), \tag{9}$$

where $\Delta_c Q^{-1}(y) = Q^{-1}(y) \oplus Q^{-1}(y \oplus c)$ has degree $\leq \deg(Q^{-1}) - 1 = 2$. For $\mathsf{YSH}_{6,6}^{-1}$ (27/50/3), $s(y)$ is quadratic (depth 1) and $s(y) \wedge \Delta_c Q^{-1}(y)$ is evaluated directly with 9 additional AND gates in parallel with $Q^{-1}(y)$ at the theoretical minimum depth $D = 2$. For $\mathsf{YSH}_{6,4}^{-1}$ (22/49/1) and $\mathsf{YSH}_{6,5}^{-1}$ (21/57/0), the chained forward shears correspond under Q^{-1} to constant input-shift vectors $c_{\text{in},j}$ gated by cubic scalar predicates $s_j(y)$

$(Q^{-1}(y) \oplus \sum_j s_j(y) c_{\mathrm{in},j})$, so the correction needs only 4 and 3 additional AND gates at depth $D = 3$ (one level shallower than $\mathsf{YSH}_{6,5}$). For the output-sheared circuits $\mathsf{YSH}_{6,2}$, $\mathsf{YSH}_{7,1}$, and $\mathsf{YSH}_{8,1}$ the shear $T(y) = y \oplus ((y_i \wedge y_j) \cdot c)$ with $c_i = c_j = 0$ is unwound with one AND gate; for the two-point-swapped Gold circuits $x^3_{\mathrm{swap}(1,3)}$ (SM C.1) and $x^5_{\mathrm{swap}(1,25)}$ (SM C.21), $S^{-1}(y) = Q^{-1}(y) \oplus \mathbb{I}_{\{y \in \{y_\alpha, y_\beta\}\}} \cdot (\alpha \oplus \beta)$ with the two-point indicator of the swapped values evaluated in parallel.

Tier 2: backward unrolling. For $\mathsf{YSH}_{5,2}$, whose two quartic coordinates are $y_2 = x_0 \oplus y_4 \oplus y_0 y_1$ and $y_3 = (x_0 \oplus x_4) \oplus y_0 \oplus y_0 y_4$, three coordinates of S^{-1} are quadratic in y: $x_0 = y_2 \oplus y_4 \oplus y_0 y_1$, $x_3 = \overline{y_1 \oplus y_2 \oplus y_0 y_1 \oplus y_3 y_4}$, and $x_4 = x_0 \oplus y_3 \oplus y_0 \oplus y_0 y_4$ (one level, three AND gates); the remaining two, $x_1 = y_4 \oplus u \oplus x_0 u$ with $u = x_0 \oplus x_4$ and $x_2 = y_0 \oplus x_3 \oplus x_3 x_4$, re-use them at level 2, so $\mathsf{YSH}^{-1}_{5,2}$ costs 5/11/2 at $D = 2$. The same propagation recovers the eight inputs of $\mathsf{YSH}_{8,3}$ with every forward AND gate re-used once (18/77/8 at $D = 6$).

Tier 3: power-map factorizations. The four Tier-3 inverse circuits factor the inverse exponent of a Category A power map into low-depth algebraic stages:

- *6-bit Q_{2258} (x^5) inverse (x^{38}):* Over $\mathbb{F}_{2^3} \cong \mathbb{F}_2[z]/(z^3 + z + 1)$ and the quadratic extension $\mathbb{F}_{2^3}[\gamma]/(\gamma^2 + \gamma + \nu)$ with $\nu = 1$ ($\mathrm{Tr}_{\mathbb{F}_{2^3}/\mathbb{F}_2}(1) = 1$, since 3 is odd), writing $y = A + B\gamma$ gives the relative norm $N = y^9 = A^2 + AB + B^2 \in \mathbb{F}_{2^3}$ using one 6-AND $\mathbb{F}_{2^3}$-product AB at level 1; $y^{38} = y^2 \cdot N^4$ then costs two 6-AND $\mathbb{F}_{2^3}$-products $(A^2 + B^2)N^4$ and $B^2 N^4$ at level 2, giving an 18/37/0, $D = 2$ circuit (SM C.16).
- *7-bit Gold (x^3) inverse (x^{85}):* Factoring $85 \equiv 5 \times 9 \times 16 \pmod{127}$ composes our 14-AND, $D = 1$ circuit for y^5 with the 15-AND, $D = 1$ circuit for z^9 and folds the Frobenius w^{16} by Paar CSE, giving a 29/149/15, $D = 2$ circuit at the theoretical minimum depth for degree 4 (SM C.22).
- *7-bit Kasami (x^{13}) inverse (x^{88}, $88 \equiv 11 \times 8 \pmod{127}$) and x^7 inverse (x^{109}, $109 \equiv 25 \times 64 + 5 \times 32 \pmod{127}$):* Over $\mathbb{F}_2[z]/(z^7 + z + 1)$, a $(4, 3)$ Karatsuba split $A(z) = A_L(z) + z^4 A_H(z)$ ($\deg A_L \le 3$, $\deg A_H \le 2$) multiplies $A_L B_L$ by recursive $(2, 2)$ Karatsuba ($3 \times 3 = 9$ ANDs), $A_H B_H$ by 3-term Karatsuba (6 ANDs), and $(A_L + A_H)(B_L + B_H)$ by recursive $(2, 2)$ Karatsuba (8 new ANDs, since $\deg A_H \le 2$ makes the $(3, 3)$ product equal $a_3 b_3$ from $A_L B_L$), giving $9 + 6 + 8 = 23$ ANDs at $D = 1$ before XOR reduction modulo $z^7 + z + 1$. This yields $y^{88} = (y^3 \cdot y^8)^8$ ($14 + 23 = 37/139/0$ at $D = 2$, SM C.23) and $y^{109} = ((y^5)^5)^{64} \cdot (y^5)^{32}$ via two 14-AND y^5 stages and one 23-AND Karatsuba product (51/220/15 at $D = 3$, five fewer ANDs than forward, SM C.25).

Tier 4: Hodge-dual coset-span synthesis. Under the exterior-algebra Hodge isomorphism $\star$: $\bigwedge^3 \mathbb{F}_2^5 \xrightarrow{\sim} \bigwedge^2 (\mathbb{F}_2^5)^*$, the homogeneous cubic part f_3 of a component $f_c(y) = \langle c, S^{-1}(y) \rangle$ ($c \in \mathbb{F}_2^5 \setminus \{0\}$) maps to an alternating 2-form $\omega_c = \star f_3$ on $\mathbb{F}_2^5$. Because 5 is odd, ω_c has symplectic rank in $\{0, 2, 4\}$ ($\dim \ker \omega_c \ge 1$) and splits into at most two rank-2 terms $\alpha_1 \wedge \beta_1 + \alpha_2 \wedge \beta_2$. Choosing a non-zero linear form $L_c(y)$ in the annihilator of one rank-2 summand ensures $\deg(f_c|_{L_c(y)=0}) \le 2$, so every cubic component on $\mathbb{F}_2^5$ decomposes as $f_c(y) = L_c(y) \cdot Q_c(y) \oplus R_c(y)$ with $\deg Q_c \le 2$ and $\deg R_c \le 2$ (and $\deg R_c \le 1$ when $\mathrm{rank}(\omega_c) \le 2$). For each Tier-4 target, `solve_5b_cubic_inverses.py` enumerates all valid divisors L_c per non-zero $c \in \mathbb{F}_2^5$ and searches for a full-rank basis $(c_1, \ldots, c_5) \in \mathrm{GL}(5, \mathbb{F}_2)$ whose ten quadratic parts $(Q_{c_i}, R_{c_i})_{i=1}^5$ lie in the $\mathbb{F}_2$-span of $r \in \{4, 5\}$ shared level-1 products $p_j = (u_j \cdot y) \wedge (v_j \cdot y)$; five level-2 AND gates then form $L_{c_i}(y) \wedge Q_{c_i}(y)$, and a 5×5 output XOR layer recovers $S^{-1}(y)$ in $r + 5$ ANDs at $D = 2$. Writing each 5-bit mask $m = \sum_{b=0}^4 m_b 2^b$ for $\bigoplus_{b=0}^4 m_b y_b$, the exact tuples $(c_i)_{i=1}^5$, $(L_{c_i})_{i=1}^5$, and $(u_j, v_j)_{j=1}^r$ are: Q_{71}^{-1} (SM C.2, $r = 4$, 9/32/2): $(20, 21, 31, 12, 3)$, $(2, 2, 5, 8, 1)$, and $(12, 21), (6, 16), (8, 16), (15, 18)$; $\mathsf{YSH}^{-1}_{5,1}$ (SM C.3, $r = 4$, 9/22/2): $(1, 2, 4, 8, 18)$, $(8, 4, 11, 9, 16)$, and $(1, 2), (1, 4), (9, 16), (10, 20)$; χ_5^{-1} (SM C.7, $r = 4$,

9/21/3): $(4, 8, 10, 16, 17)$, $(2, 1, 8, 16, 20)$, and $(1, 8), (3, 24), (2, 12), (4, 16)$; and Gold x^{25} (SM C.5, $r = 5$, 10/37/4): $(18, 19, 3, 29, 5)$, $(1, 2, 4, 8, 16)$, and $(9, 22), (13, 17), (3, 5), (8, 18), (14, 16)$.

Table 10: Provenance of the 30 benchmark circuits (Section 6.5). **Seed**: the Stage-1 seed of the episode that produced the archived circuit—a published or algebraic library circuit (L), a structural template or power-map lookup-table node (T), or an archive elite (A); generator strings in parentheses are internal library/template keys whose nominal suffixes reflect pre-expansion or base templates (e.g., legacy key `q2256_inv_6b_11and_d2` expands the cubic Q_{2258}^{-1} (x^{38}) LUT to 26/65, D=2; `dillon_apn_6b_12and_d3` expands to 28/75, D=2; `gold_x3_7b_11and_d1` expands to 14/59, D=1 and stays at 14 ANDs because the per-candidate D=1 SAT budget at n=7 timed out below 14 ANDs; `ccz_lifted_8b_inversion_aes` appends a 1-AND shear to the 42-AND, D=3 inverter), whereas **Seed A/X, D** reports the actual AND/XOR count and AND depth of the expanded logic-gate DAG at step 0 (– where the seed is a template or lookup-table node lowered during Stage 2; unlowered step-0 nodes log nominal depth 1 before lowering to 11/20, 15/32, 14/45 at D=2 for the three non-quadratic 5-bit power-map nodes and 51/7 at D=3 for the 7-bit unbalanced Feistel template); **Archived**: the certified netlist admitted to the archive after Stages 2–3; **Printed**: the netlist of SM C after the Stage-4 pass (Section 6.5).

S-box (SM)		Stage-1 seed (generator)	Seed A/X, D	Archived A/X/N, D	Printed A/X/N, D
$x^3_{\mathrm{swap}(1,3)}$ class (C.1)	T	power-map LUT node (`novel_power_map_5b`)	–	11/20/3, 2	11/20/3, 2
Q_{71} class (C.2)	T	depth-1 quadratic template, SAT-sampled (`mc5_md1`)	5/–, 1	5/14/3, 1	5/12/3, 1
$\mathsf{YSH}_{5,1}$ (C.3)	L	Keccak χ_5 circuit (`keccak_chi5`), +1 shear gate	5/10, 1	6/11/0, 2	6/11/0, 2
$\mathsf{YSH}_{5,2}$ (C.4)	A	archive elite, δ=6 niche (`swarm_transfection`)	16/39, 2	5/15/1, 3	5/12/1, 2
Gold x^5 (C.5)	T	power-map LUT node (`novel_power_map_5b`)	7/22, 1	7/22/0, 1	7/22/0, 1
x^5 class, SAC = 0 (C.6)	T	power-map LUT node (`novel_power_map_5b`)	7/21, 1	7/23/4, 1	7/21/4, 1
χ_5 class (C.7)	A	archive elite (`swarm_transfection`)	5/11, 1	5/8/1, 1	5/8/1, 1
inverse-Gold (C.8)	T	power-map LUT node (`novel_power_map_5b`)	–	14/32/2, 2	14/31/2, 2
x^{15} (C.9)	T	power-map LUT node (`novel_power_map_5b`)	–	14/45/0, 2	14/45/0, 2
$\mathsf{YSH}_{6,1}$ (C.10)	L	algebraic library circuit (`golden_6b_row26`)	9/27, 3	9/29/1, 3	9/27/1, 3
$\mathsf{YSH}_{6,2}$ (C.11)	L	cubic Q_{2258}^{-1} (x^{38}) circuit (legacy key `q2256_inv_6b_11and_d2`)	26/65, 2	27/64/0, 3	27/64/0, 3
$\mathsf{YSH}_{6,3}$ (C.12)	T	3-round Feistel template (`structural_feistel_6b`)	9/18, 3	10/19/9, 3	9/17/9, 3
$\mathsf{YSH}_{6,4}$ (C.13)	L	Q_{2258} circuit (`q2258_6b_8and_d1`), +2 shear gates	9/30, 1	11/28/1, 3	11/22/1, 3
$\mathsf{YSH}_{6,5}$ (C.14)	L	Q_{2258} circuit (`q2258_6b_8and_d1`), +3 shear gates	9/30, 1	12/43/0, 4	12/31/0, 4
$\mathsf{YSH}_{6,6}$ (C.15)	L	Q_{2258} circuit (`q2258_6b_8and_d1`), +1 shear gate	9/30, 1	10/32/0, 2	10/22/0, 2
Q_{2258} class (C.16)	L	Q_{2258} circuit (`q2258_6b_8and_d1`)	9/30, 1	8/50/0, 1	8/34/0, 1
χ_3-Feistel invol. (C.17)	T	Misty template (`structural_misty_6b`)	–	9/18/9, 3	9/15/9, 3
inversion (C.18)	L	$\mathbb{F}_{(2^3)^2}$ inversion (`algebraic_6b_sota_inversion_x31_d3`)	21/67, 3	21/58/0, 3	21/48/0, 3
Dillon APN (C.19)	L	Dillon APN circuit (`dillon_apn_6b_12and_d3`)	28/75, 2	28/74/0, 2	28/68/0, 2
$\mathsf{YSH}_{7,1}$ (C.20)	L	fractional subfield tower (`fractional_tower_7b_9and_d2`)	15/71, 2	15/71/10, 2	15/70/10, 2
$x^5_{\mathrm{swap}(1,25)}$ class (C.21)	T	unbalanced Feistel template (`unbalanced_feistel_7b`)	–	51/77/1, 3	51/76/1, 3
Gold x^3 (C.22)	L	Gold x^3 circuit (`gold_x3_7b_11and_d1`)	14/59, 1	14/59/0, 1	14/57/0, 1
Kasami x^{13} (C.23)	A	archive elite from a Kasami power-map node (`apn_kasami_k2_x13`)	30/115, 2	30/114/0, 2	30/114/0, 2
inversion (C.24)	L	sheared 7-bit inversion (`sheared_7b_inversion_standard`)	43/182, 4	42/182/6, 3	42/182/6, 3
power map x^7 (C.25)	T	power-map LUT node (`power_map_x7_7b_d2`)	–	56/177/0, 2	56/98/0, 2
$\mathsf{YSH}_{8,1}$ (C.26)	T	CCZ-lifted 8-bit inversion (`ccz_lifted_8b_inversion_aes`)	43/120, 4	43/107/7, 4	43/107/7, 4
$\mathsf{YSH}_{8,2}$ (C.27)	L	Canright tower-field AES [Can05] (`canright_aes_8b_32and_d4`)	36/136, 4	28/61/1, 6	28/61/1, 6
$\mathsf{YSH}_{8,3}$ (C.28)	A	archive elite $\mathsf{YSH}_{8,2}$ (`swarm_transfection`), + AND erasure	28/62, 6	18/73/2, 5	18/71/2, 5
AES, D=3 (C.29)	L	depth-3 AES inverter [HB25] (`aes_8b_42and_d3`)	42/116, 3	42/103/4, 3	42/101/4, 3
AES FIPS-197 (C.30)	L	Camellia circuit, same inversion class (`camellia_8b_32and_d4`)	–	36/119/4, 4	36/99/4, 4

Table 11: The 56 $(n, \delta, L, \deg)$ profiles present in the archive of Table 7. Gr.: threshold tier of Table 1; Netlists: distinct certified netlists; Dist.: distinct permutations; Cat. A/B: distinct permutations in each category (all 2,681 Category A permutations are proven equal or affine equivalent to their catalogue reference in Table 6, with zero unresolved collisions). Min-AND rep.: the netlist of the profile with the fewest AND gates (ties broken by smaller depth, then in favour of a benchmark circuit of Section 6, then by fewer XOR gates); its counts A/X/N and depth D are those after the function-preserving Stage-4 pass of Section 4.3, re-verified by exhaustive simulation (for the ANF-synthesized representatives with more than 100 AND gates, after merging duplicate gates and pruning dead wires only). Class: as in Table 2 (=: exact catalogue table; $\sim_{\text{aff}}$: certified affine equivalent; $\mathsf{YSH}_{n,i}$: Category B benchmark class); the inverses of benchmark circuits are archive members in their own right and are named as such. Min-D rep.: the lowest-depth netlist of the profile, when its depth is smaller than that of the min-AND representative. The 56 profiles contain 104 per-profile non-dominated $(\text{AND}, \text{XOR}, D)$ points in total (Table 7).

n	$(\delta, L, \deg)$	Gr.	Netlists	Dist.	Cat. A/B	Min-AND (A/X/N)	D	Class	Min-D (if shallower)
5	$(2, 8, 2)$	**A+**	130	85	85/0	7/21/4	1	A ($\sim_{\text{aff}} x^5$)	–
5	$(2, 8, 3)$	**A+**	165	160	160/0	14/31/2	2	A ($\sim_{\text{aff}} x^7$)	–
5	$(2, 12, 4)$	**A**	163	149	149/0	14/45/0	2	A ($= x^{15}$)	–
5	$(4, 12, 4)$	**A**	843	828	690/138	11/20/3	2	A ($\sim_{\text{aff}} x^3_{\text{swap}(1,3)}$)	–
5	$(4, 16, 2)$	**B+**	17	7	7/0	5/12/3	1	A ($\sim_{\text{aff}} Q_{71}$)	–
5	$(4, 16, 3)$	**B+**	1	1	1/0	9/32/2	2	A ($\sim_{\text{aff}} Q_{71}^{-1}$; inverse of Q_{71} class)	–
5	$(4, 16, 4)$	**B+**	618	597	199/398	12/26/2	2	B	–
5	$(6, 12, 4)$	**B+**	6	3	0/3	16/36/2	2	B	–
5	$(6, 16, 4)$	**B+**	25	16	0/16	7/16/1	4	B	15/33/3, $D = 2$, B
5	$(8, 16, 2)$	**B+**	12	8	8/0	5/8/1	1	A ($\sim_{\text{aff}} \chi_5$)	–
5	$(8, 16, 3)$	**B+**	20	14	1/13	5/12/1	3	B	6/11/0, $D = 2$, B ($\mathsf{YSH}_{5,1}$)
5	$(8, 16, 4)$	**B+**	390	341	0/341	5/12/1	2	B ($\mathsf{YSH}_{5,2}$)	–
6	$(2, 16, 4)$	**A+**	466	462	462/0	28/68/0	2	A (= Dillon)	–
6	$(4, 16, 2)$	**A**	6	4	4/0	8/34/0	1	A ($\sim_{\text{aff}} x^5$)	–
6	$(4, 16, 3)$	**A**	12	6	4/2	9/37/0	2	B ($\mathsf{YSH}_{6,2}^{-1}$)	–
6	$(4, 16, 4)$	**A**	3	3	0/3	9/17/8	3	B ($\mathsf{YSH}_{6,3}^{-1}$)	–
6	$(4, 16, 5)$	**A**	6	4	4/0	21/48/0	3	A ($\sim_{\text{aff}} x^{31}$)	–
6	$(4, 20, 5)$	**A**	1	1	0/1	29/71/0	3	B	–
6	$(6, 20, 5)$	**B+**	2	1	0/1	55/90/2	3	B	–
6	$(6, 24, 5)$	**B+**	19,380	19,369	0/19,369	12/31/0	4	B ($\mathsf{YSH}_{6,5}$)	21/57/0, $D = 3$, B ($\mathsf{YSH}_{6,5}^{-1}$)
6	$(6, 28, 5)$	**B+**	738	723	0/723	52/84/4	3	B	–
6	$(8, 24, 4)$	**B+**	2	2	0/2	10/22/0	2	B ($\mathsf{YSH}_{6,6}$)	–
6	$(8, 24, 5)$	**B+**	1,729	1,717	0/1,717	12/24/2	4	B	49/80/2, $D = 3$, B
6	$(8, 28, 5)$	**B+**	1,611	1,599	0/1,599	11/22/1	3	B ($\mathsf{YSH}_{6,4}$)	–
7	$(2, 16, 2)$	**A+**	19	15	15/0	14/57/0	1	A ($= x^3$)	–
7	$(2, 16, 3)$	**A+**	160	151	151/0	30/114/0	2	A ($= x^{13}$)	–
7	$(2, 16, 4)$	**A+**	596	581	581/0	29/149/15	2	A ($\sim_{\text{aff}} x^{43}$; inverse of Gold x^3)	–
7	$(2, 20, 6)$	**A**	124	116	116/0	42/182/6	3	A ($\sim_{\text{aff}} x^{63}$)	–
7	$(4, 20, 6)$	**A**	8	3	3/0	34/164/9	3	A ($\sim_{\text{aff}} (x^5_{\text{swap}(1,25)})^{-1}$; inverse of $x^5_{\text{swap}(1,25)}$ class)	–
7	$(4, 24, 6)$	**A**	3	2	2/0	114/199/0	3	A ($\sim_{\text{aff}} x^{63}_{\text{swap}(1,2)}$)	–
7	$(4, 32, 4)$	**A**	9	4	0/4	15/70/10	2	B ($\mathsf{YSH}_{7,1}$)	–
7	$(4, 32, 5)$	**A**	9	5	0/5	30/158/12	3	B ($\mathsf{YSH}_{7,1}^{-1}$)	–
7	$(4, 36, 6)$	**B+**	10	5	0/5	43/179/11	4	B	–
7	$(4, 40, 6)$	**B+**	4	2	0/2	43/179/5	4	B	–
7	$(6, 20, 6)$	**B+**	10	3	0/3	55/76/5	3	B	–
7	$(6, 24, 6)$	**B+**	21	6	0/6	56/79/3	3	B	–
7	$(6, 40, 3)$	**B+**	2	1	1/0	56/98/0	2	A ($= x^7$)	–
7	$(6, 40, 5)$	**B+**	1	1	1/0	51/220/15	3	A ($\sim_{\text{aff}} x^{55}$; inverse of power map x^7)	–
8	$(4, 32, 7)$	**A+**	50	37	37/0	36/95/4	4	A ($= \text{AES}^{-1}$; inverse of AES FIPS-197)	42/101/4, $D = 3$, A ($\sim_{\text{aff}}$ AES)
8	$(6, 36, 7)$	**A**	5	3	0/3	245/411/2	3	B	–
8	$(6, 40, 7)$	**A**	4	2	0/2	246/418/4	3	B	–
8	$(6, 56, 7)$	**B+**	10	10	0/10	37/106/13	5	B	43/107/7, $D = 4$, B ($\mathsf{YSH}_{8,1}$)
8	$(8, 56, 7)$	**B+**	2	1	0/1	246/417/5	3	B	–
8	$(8, 60, 7)$	**B+**	2	1	0/1	246/415/3	3	B	–
8	$(10, 68, 7)$	**B+**	1,167	1,167	0/1,167	239/851/3	3	B	–
8	$(10, 72, 7)$	**B+**	600	596	0/596	28/61/1	6	B ($\mathsf{YSH}_{8,2}$)	239/876/4, $D = 3$, B
8	$(12, 64, 7)$	**B+**	519	518	0/518	240/845/2	3	B	–
8	$(12, 68, 7)$	**B+**	1,113	1,113	0/1,113	240/847/0	3	B	–
8	$(12, 72, 7)$	**B+**	526	526	0/526	24/64/1	6	B	240/848/2, $D = 3$, B
8	$(12, 76, 7)$	**B+**	1	1	0/1	22/66/2	6	B	–
8	$(14, 68, 7)$	**B+**	1	1	0/1	26/61/1	6	B	–
8	$(14, 72, 7)$	**B+**	4	4	0/4	18/71/2	5	B ($\mathsf{YSH}_{8,3}$)	–
8	$(14, 76, 7)$	**B+**	3	3	0/3	20/68/2	6	B	–
8	$(16, 72, 7)$	**B+**	3	3	0/3	25/63/1	6	B	–
8	$(16, 76, 7)$	**B+**	1	1	0/1	27/60/1	6	B	–
8	$(16, 80, 7)$	**B+**	4	4	0/4	27/59/1	6	B	–

C Lookup Tables and Gate Netlists

This section of the SM lists every S-box that the body names together with its inverse S^{-1}: for both directions we give the lookup table in hexadecimal and the complete gate netlist, in the order in which the gates are emitted (`in_val` is the input word, bit 0 first; `out_val` is the output word, written most significant bit (MSB) first). For every listing we recomputed δ, L, NL, $\deg(S)$, $\mathcal{D}(S)$, the number of fixed points (FP) and the SAC deviation from the printed lookup table, and we simulated the printed netlist on all 2^n inputs: each netlist computes its lookup table exactly. The gate counts and AND depths below are those of the printed netlists, which contain no gate that feeds no output. Unmasked and first-order pipelined DOM-indep ($d = 2$) silicon areas (μm^2 and Gate Equivalents, 1 GE = 0.798 μm^2) and NLDM critical-path delays (ps) are obtained by gate-exact Yosys and ABC synthesis on the NanGate 45 nm Open Cell Library (Section 5.2). Category A entries are either equal to an entry of the 712-LUT reference catalogue (=) or certified affine equivalent to it ($\sim_{\text{aff}}$, with the explicit witness $S(x) = B \cdot S_{\text{ref}}(Ax \oplus a) \oplus b$ given by the column vectors $[L(1 \ll 0), \ldots, L(1 \ll (n-1))]$ of $A, B \in \mathrm{GL}(n, \mathbb{F}_2)$ and constants $a, b \in \mathbb{F}_2^n$); Category B entries ($\mathsf{YSH}_{n,i}$, with inverses $\mathsf{YSH}_{n,i}^{-1}$) are certified EA-separated from every entry of the reference catalogue (Section 6.1) and lie in 12 pairwise disjoint EA classes. Two equivalent wrapper conventions appear in the listings: either the n-bit input word `in_val` is split into `a_in[i:i]` bits ($i = 0$ least significant bit (LSB) to $n-1$ MSB) and the output word `out_val` is assembled by a Verilog concatenation `{y_n-1, ..., y_0}` (MSB first, rightmost wire LSB), or the bits are addressed directly by index as `x[i]` and `y[i]` ($i = 0$ LSB to $n-1$ MSB); `sync_buf_*` assignments are zero-cost wire aliases emitted by the stage scheduler and are not counted as gates.

C.1 5-bit swapped Gold $x^3_{\mathbf{swap(1,3)}}$ class, $\delta = 4$, $L = 12$ (5-bit, Category A)

- **Profile**: $\delta = 4$, $L = 12$ (NL = 10), $\deg(S) = 4$, $\mathcal{D}(S) = [2^{15}, 4^{16}]$, FP = 0, SAC deviation 0.125.
- **Cost**: 11 AND, 20 XOR, 3 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 45.22 μm^2 / 56.7 GE, delay 334.4 ps, logic depth 7; pipelined $d = 2$ DOM-indep: 66 `DFF_X1` [44 gadget + 22 sync], 480.93 μm^2 / 602.7 GE, stage delay 310.0 ps).
- **Remark**: Certified affine equivalent to the two-point-swapped Gold permutation $x^3_{\text{swap}(1,3)}$ over $\mathbb{F}_2[x]/(x^5 + x^2 + 1)$ [BKRS22] ($\sim_{\text{aff}}$ `GF32_Gold_x3_swap(1,3)`, with $A =$ [`0x17`, `0x14`, `0x06`, `0x02`, `0x0a`], $a =$ `0x0e`, $B =$ [`0x19`, `0x1b`, `0x09`, `0x1e`, `0x1d`], $b =$ `0x05`). Same $(\delta, L, \deg)$ as DB_1 [AL26] (9/36/2 in [BDM26b]): two more AND gates and 16 fewer XOR gates at the same depth. Its quadratic components form a hyperplane, and its AND gates are 10 at level 1 and one at level 2 (Section 6.6).
- **Inverse (S^{-1}, $\sim_{\mathbf{aff}}$ $(x^3_{\mathbf{swap}(1,3)})^{-1}$)**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [3^{15}, 4^{16}]$; cost: 13 AND, 38 XOR, 6 NOT, AND depth $D = 2$ (Tier 2: Parallel Transposition Inversion on 10-AND Gold inverse core; unmasked NanGate 45 nm: 77.67 μm^2 / 97.3 GE, delay 623.2 ps, logic depth 12; pipelined $d = 2$ DOM-indep: 84 `DFF_X1` [52 gadget + 32 sync], 642.66 μm^2 / 805.3 GE, stage delay 323.9 ps).

Lookup table (LUT[0 . . . 31]):

```
0x15, 0x0f, 0x19, 0x14, 0x1d, 0x16, 0x0b, 0x17, 0x02, 0x1a, 0x1e, 0x11, 0x07, 0x0e, 0x01, 0x1f
0x00, 0x12, 0x09, 0x0c, 0x1b, 0x18, 0x08, 0x10, 0x13, 0x03, 0x0a, 0x0d, 0x05, 0x04, 0x06, 0x1c
```

Netlist (11 AND, 20 XOR, 3 NOT, $D = 2$):

```
a_in = in_val[4:0]; split_2 = a_in[0:0]; split_3 = a_in[1:1];
split_4 = a_in[2:2]; split_5 = a_in[3:3]; split_6 = a_in[4:4];
and_7 = split_2 & split_3; and_8 = split_2 & split_4; and_9 = split_3 & split_4;
and_10 = split_2 & split_5; and_11 = split_3 & split_5;
and_12 = split_4 & split_5; and_13 = split_2 & split_6;
and_14 = split_3 & split_6; and_15 = split_4 & split_6;
and_16 = split_5 & split_6; xor_18 = split_5 ^ and_7; xor_19 = split_2 ^ and_9;
xor_21 = split_3 ^ and_12; xor_34 = split_4 ^ and_13; and_17 = and_7 & and_15;
xor_22 = and_8 ^ and_15; xor_28 = and_10 ^ and_15; xor_20 = split_6 ^ xor_18;
xor_29 = xor_18 ^ xor_19; xor_37 = and_11 ^ xor_19; xor_35 = xor_19 ^ xor_34;
xor_24 = and_17 ^ xor_21; xor_25 = and_12 ^ xor_22; xor_23 = and_14 ^ xor_20;
xor_38 = xor_22 ^ xor_20; xor_30 = xor_28 ^ xor_29; xor_36 = xor_35 ^ xor_24;
xor_26 = xor_25 ^ xor_23; xor_31 = and_16 ^ xor_23; xor_39 = xor_37 ^ xor_38;
not_27 = ~xor_26; xor_32 = xor_31 ^ xor_24; not_40 = ~xor_39; not_33 = ~xor_32;
concat_41 = { not_40, xor_36, not_33, xor_30, not_27 }; out_val = concat_41;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 31]$):

```
0x10, 0x0e, 0x08, 0x19, 0x1d, 0x1c, 0x1e, 0x0c, 0x16, 0x12, 0x1a, 0x06, 0x13, 0x1b, 0x0d, 0x01
0x17, 0x0b, 0x11, 0x18, 0x03, 0x00, 0x05, 0x07, 0x15, 0x02, 0x09, 0x14, 0x1f, 0x04, 0x0a, 0x0f
```

Inverse netlist (13 AND, 38 XOR, 6 NOT, $D = 2$):

```
xor_0 = x[1] ^ x[4]; xor_1 = x[2] ^ x[3]; xor_2 = x[0] ^ x[2];
xor_3 = x[0] ^ xor_1; xor_4 = x[4] ^ xor_3; xor_5 = xor_0 ^ xor_3;
xor_6 = x[4] ^ xor_1; xor_7 = x[2] ^ xor_0; xor_8 = x[4] ^ xor_2;
xor_9 = x[1] ^ xor_1; xor_10 = x[1] ^ xor_2; xor_11 = x[1] ^ x[2];
xor_12 = x[2] ^ x[4]; not_0 = ~x[0]; not_1 = ~x[1]; not_2 = ~xor_1;
and_0 = xor_2 & xor_4; and_1 = x[3] & xor_5; and_2 = xor_2 & xor_6;
and_3 = xor_1 & xor_7; and_4 = xor_1 & xor_8; and_5 = not_0 & not_1;
and_6 = not_2 & x[4]; and_7 = and_5 & and_6; xor_13 = and_2 ^ xor_0;
xor_14 = and_1 ^ and_4; xor_15 = and_3 ^ xor_13; xor_16 = and_0 ^ and_1;
xor_17 = xor_13 ^ xor_16; xor_18 = x[2] ^ and_1; xor_19 = xor_15 ^ xor_18;
xor_20 = and_1 ^ and_3; xor_21 = x[4] ^ and_0; xor_22 = and_4 ^ xor_21;
xor_23 = and_0 ^ xor_15; xor_24 = xor_15 ^ and_7; xor_25 = x[3] ^ and_0;
xor_26 = and_2 ^ xor_25; not_3 = ~xor_19; not_4 = ~xor_20;
and_8 = xor_17 & xor_9; and_9 = not_3 & xor_3; and_10 = not_4 & xor_10;
and_11 = xor_22 & xor_11; and_12 = xor_12 & xor_14; xor_27 = and_4 ^ and_9;
xor_28 = and_10 ^ and_11; xor_29 = xor_14 ^ xor_28; xor_30 = xor_0 ^ and_10;
xor_31 = xor_27 ^ xor_30; xor_32 = xor_13 ^ xor_29; xor_33 = xor_23 ^ and_8;
xor_34 = xor_27 ^ xor_24; xor_35 = xor_26 ^ and_9; xor_36 = and_12 ^ xor_29;
xor_37 = xor_35 ^ xor_36; not_5 = ~xor_37; y[0] = xor_31; y[1] = xor_32;
y[2] = xor_33; y[3] = xor_34; y[4] = not_5;
```

C.2 5-bit quadratic Q_{71} class, $\delta = 4$, $L = 16$, 5 AND (5-bit, Category A)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 8), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{31}]$, FP = 0, SAC deviation 0.5.
- **Cost**: 5 AND, 12 XOR, 3 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 26.07 μm^2 / 32.7 GE, delay 198.3 ps, logic depth 5; pipelined $d = 2$ DOM-indep: 30 `DFF_X1` [20 gadget + 10 sync], 228.76 μm^2 / 286.7 GE, stage delay 202.1 ps).
- **Remark**: Certified affine equivalent to quadratic class Q_{71} of Božilov, Bilgin, and Şahin [BBŞ17] ($\sim_{\text{aff}}$ `BBS17_Q71`, with A = [`0x0b`, `0x02`, `0x0a`, `0x1d`, `0x0f`], a = `0x03`, B = [`0x0d`, `0x0c`, `0x0a`, `0x03`, `0x1a`], b = `0x03`). Because every non-zero component of Q_{71} has algebraic degree 2 ($\mathcal{D}(S) = [2^{31}]$), its five output coordinates are linearly independent modulo RM(1, 5), so any Boolean circuit requires at least 5 AND gates; this circuit is therefore provably AND-optimal and depth-optimal (5 AND, $D = 1$), achieving $\delta = 4$ instead of 8 at the AND count and depth of χ_5. Structure and origin: an instance of the depth-1 quadratic template $y_k = M_k x \oplus (\alpha_k \cdot x \oplus a_k)(\beta_k \cdot x \oplus b_k)$ (one product per output), drawn by the template's SAT sampler with this profile directly (Section 6.5); the linear-layer passes reduced the sampled netlist from 5/20/3 to 5/12/3 (5/11/1 pre-Stage-4).
- **Inverse (S^{-1}, $\sim_{\text{aff}}$ Q_{71}^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{31}]$; cost: 9 AND, 32 XOR, 2 NOT, AND depth $D = 2$ (Tier 4: Hodge-dual linear-divisor coset-span $D = 2$

synthesis; unmasked NanGate 45 nm: 61.71 μm^2 / 77.3 GE, delay 530.1 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 68 `DFF_X1` [36 gadget + 32 sync], 506.46 μm^2 / 634.7 GE, stage delay 302.7 ps).

Lookup table (LUT$[0 \dots 31]$):

```
0x02, 0x04, 0x0e, 0x19, 0x0a, 0x0f, 0x17, 0x03, 0x11, 0x06, 0x1c, 0x1a, 0x01, 0x15, 0x1d, 0x18
0x08, 0x0c, 0x10, 0x05, 0x00, 0x07, 0x09, 0x1f, 0x0b, 0x1e, 0x12, 0x16, 0x1b, 0x0d, 0x13, 0x14
```

Netlist (5 AND, 12 XOR, 3 NOT, $D = 1$):

```
xor_0 = x[2] ^ x[4]; xor_1 = x[2] ^ x[3]; xor_2 = x[0] ^ xor_0;
xor_3 = x[1] ^ x[3]; xor_4 = x[0] ^ x[1]; xor_5 = x[1] ^ xor_1;
xor_6 = x[1] ^ x[4]; not_0 = ~xor_6; not_1 = ~xor_0; not_2 = ~xor_2;
and_0 = x[2] & xor_1; and_1 = xor_3 & xor_2; and_2 = xor_4 & xor_5;
and_3 = not_0 & x[4]; and_4 = not_1 & not_2; xor_7 = and_2 ^ xor_3;
xor_8 = and_4 ^ xor_1; xor_9 = and_3 ^ xor_4; xor_10 = and_0 ^ xor_6;
xor_11 = x[3] ^ and_1; y[0] = xor_7; y[1] = xor_8; y[2] = xor_9; y[3] = xor_10;
y[4] = xor_11;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 31]$):

```
0x14, 0x0c, 0x00, 0x07, 0x01, 0x13, 0x09, 0x15, 0x10, 0x16, 0x04, 0x18, 0x11, 0x1d, 0x02, 0x05
0x12, 0x08, 0x1a, 0x1e, 0x1f, 0x0d, 0x1b, 0x06, 0x0f, 0x03, 0x0b, 0x1c, 0x0a, 0x0e, 0x19, 0x17
```

Inverse netlist (9 AND, 32 XOR, 2 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[4]; xor_1 = x[1] ^ x[3]; xor_2 = x[0] ^ x[2];
xor_3 = x[2] ^ x[3]; xor_4 = x[2] ^ xor_0; xor_5 = x[1] ^ x[2];
xor_6 = xor_1 ^ xor_2; xor_7 = x[1] ^ x[4]; xor_8 = x[1] ^ xor_0;
not_0 = ~x[2]; and_0 = xor_3 & xor_4; and_1 = xor_5 & x[4];
and_2 = x[3] & x[4]; and_3 = xor_6 & xor_7; xor_9 = x[2] ^ and_0;
xor_10 = and_1 ^ xor_1; xor_11 = x[2] ^ and_2; xor_12 = x[3] ^ and_3;
xor_13 = and_2 ^ xor_10; xor_14 = x[4] ^ xor_9; xor_15 = not_0 ^ xor_12;
xor_16 = xor_0 ^ xor_13; xor_17 = xor_0 ^ and_1; xor_18 = and_3 ^ xor_17;
xor_19 = xor_11 ^ xor_12; not_1 = ~xor_18; and_4 = xor_14 & x[1];
and_5 = xor_15 & x[1]; and_6 = xor_2 & xor_16; and_7 = not_1 & x[3];
and_8 = xor_19 & x[0]; xor_20 = and_6 ^ and_8; xor_21 = and_4 ^ and_5;
xor_22 = and_4 ^ xor_20; xor_23 = and_7 ^ not_0; xor_24 = xor_11 ^ xor_21;
xor_25 = x[4] ^ and_8; xor_26 = xor_21 ^ xor_25; xor_27 = xor_8 ^ xor_22;
xor_28 = xor_23 ^ xor_27; xor_29 = xor_9 ^ xor_22; xor_30 = xor_13 ^ xor_20;
xor_31 = xor_23 ^ xor_30; y[0] = xor_24; y[1] = xor_26; y[2] = xor_28;
y[3] = xor_29; y[4] = xor_31;
```

C.3 $\mathsf{YSH}_{5,1}$: 5-bit cubic, $\delta = 8$, $L = 16$ (5-bit, Category B)

- **Profile**: $\delta = 8$, $L = 16$ (NL = 8), $\deg(S) = 3$, $\mathcal{D}(S) = [2^{15}, 3^{16}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 6 AND, 11 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 23.94 μm^2 / 30.0 GE, delay 231.6 ps, logic depth 5; pipelined $d = 2$ DOM-indep: 48 `DFF_X1` [24 gadget + 24 sync], 316.01 μm^2 / 396.0 GE, stage delay 187.6 ps).
- **Remark**: Same $(\delta, L, \deg)$ as χ_5^{-1} (9/27/2 in [BDM26b]): three fewer AND gates and 16 fewer XOR gates at the same depth. Structure and origin: the five level-1 products $x_1x_2, x_2x_3, x_3x_4, x_4x_0, x_0x_1$ are those of the Keccak χ_5 seed circuit (without the level-2 gate the netlist computes that seed's lookup table exactly), and the single level-2 AND gate is a Stage-2 output shear that raises the degree to 3 (Section 6.5).
- **Inverse ($\mathsf{YSH}_{5,1}^{-1}$)**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{31}]$; cost: 9 AND, 22 XOR, 2 NOT, AND depth $D = 2$ (Tier 4: Hodge-dual linear-divisor coset-span $D = 2$ synthesis; unmasked NanGate 45 nm: 45.75 μm^2 / 57.3 GE, delay 366.8 ps, logic depth 8; pipelined $d = 2$ DOM-indep: 62 `DFF_X1` [36 gadget + 26 sync], 447.41 μm^2 / 560.7 GE, stage delay 329.7 ps).

Lookup table (LUT$[0 \dots 31]$):

```
0x00, 0x12, 0x09, 0x18, 0x14, 0x06, 0x0d, 0x1c, 0x0a, 0x1a, 0x03, 0x10, 0x16, 0x04, 0x0f, 0x1e
0x05, 0x15, 0x0c, 0x1f, 0x11, 0x01, 0x08, 0x1b, 0x0b, 0x19, 0x02, 0x13, 0x17, 0x07, 0x0e, 0x1d
```

Netlist (6 AND, 11 XOR, 0 NOT, $D = 2$):

```
and_6 = x[1] & x[2]; xor_11 = x[0] ^ x[2]; and_7 = x[2] & x[3];
xor_13 = x[1] ^ x[3]; and_8 = x[3] & x[4]; xor_15 = x[2] ^ x[4];
xor_19 = x[4] ^ x[1]; xor_12 = xor_11 ^ and_6; xor_14 = xor_13 ^ and_7;
xor_16 = xor_15 ^ and_8; and_9 = x[4] & x[0]; and_29 = x[0] & xor_14;
xor_17 = x[3] ^ x[0]; xor_26 = and_29 ^ xor_17; xor_18 = xor_26 ^ and_9;
and_10 = x[0] & x[1]; xor_20 = xor_19 ^ and_10; y[0] = xor_20; y[1] = xor_18;
y[2] = xor_16; y[3] = xor_14; y[4] = xor_12;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 31]$):

```
0x00, 0x15, 0x1a, 0x0a, 0x0d, 0x10, 0x05, 0x1d, 0x16, 0x02, 0x08, 0x18, 0x12, 0x06, 0x1e, 0x0e
0x0b, 0x14, 0x01, 0x1b, 0x04, 0x11, 0x0c, 0x1c, 0x03, 0x19, 0x09, 0x17, 0x07, 0x1f, 0x0f, 0x13
```

Inverse netlist (9 AND, 22 XOR, 2 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[3]; xor_1 = x[1] ^ x[4]; xor_2 = x[0] ^ x[2];
xor_3 = x[1] ^ x[3]; xor_4 = x[2] ^ x[4]; xor_5 = x[1] ^ xor_0;
and_0 = x[0] & x[1]; and_1 = x[0] & x[2]; and_2 = xor_0 & x[4];
and_3 = xor_3 & xor_4; xor_6 = x[2] ^ and_1; xor_7 = and_2 ^ xor_1;
xor_8 = and_0 ^ xor_2; xor_9 = and_3 ^ xor_6; xor_10 = xor_7 ^ xor_9;
xor_11 = x[4] ^ xor_6; xor_12 = xor_0 ^ xor_9; xor_13 = x[4] ^ xor_8;
xor_14 = xor_1 ^ xor_6; not_0 = ~xor_10; not_1 = ~xor_12; and_4 = xor_8 & x[3];
and_5 = xor_7 & x[2]; and_6 = xor_5 & not_0; and_7 = xor_0 & xor_11;
and_8 = not_1 & x[4]; xor_15 = and_5 ^ xor_7; xor_16 = xor_13 ^ and_4;
xor_17 = x[3] ^ xor_15; xor_18 = x[2] ^ and_6; xor_19 = xor_14 ^ and_7;
xor_20 = xor_0 ^ and_8; xor_21 = xor_15 ^ xor_20; y[0] = xor_16; y[1] = xor_17;
y[2] = xor_18; y[3] = xor_19; y[4] = xor_21;
```

C.4 $\mathsf{YSH}_{5,2}$: 5-bit quartic, $\delta = 8$, $L = 16$, $D = 2$ (5-bit, Category B)

- **Profile**: $\delta = 8$, $L = 16$ (NL = 8), $\deg(S) = 4$, $\mathcal{D}(S) = [2^7, 3^8, 4^{16}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 5 AND, 12 XOR, 1 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 25.00 μm^2 / 31.3 GE, delay 295.4 ps, logic depth 7; pipelined $d = 2$ DOM-indep: 40 `DFF_X1` [20 gadget + 20 sync], 272.92 μm^2 / 342.0 GE, stage delay 246.2 ps).
- **Remark**: Algebraic degree 4 at the 5-AND count of χ_5 and at depth $D = 2$, the minimum for degree 4: the three quadratic coordinates $y_0 = x_2 \oplus x_3 \oplus x_3x_4$, $y_4 = x_1 \oplus u \oplus x_0u$ ($u = x_0 \oplus x_4$), and $y_1 = \overline{x_3 \oplus x_1 \oplus ux_1} \oplus x_0$ are computed at level 1, and the two quartic coordinates reuse them in the two level-2 products $y_2 = x_0 \oplus y_4 \oplus y_0y_1$ and $y_3 = u \oplus y_0 \oplus y_0y_4$, so S and S^{-1} both cost 5 AND at $D = 2$. The 16 quartic components are the complement of the hyperplane $V_{\leq 3}$, inside which the 7 quadratic components form a 3-dimensional subspace. As a two-round unbalanced Feistel-type network, it has maximal boomerang uniformity $\beta = 32$ (e.g., $\mathrm{BCT}(2, 2) = 32$), like the 3-round Feistel circuits of Section 6.6. Stage-1 seed: an archive elite with 16 AND gates at $D = 2$ and $\delta = 6$, of which Stage 2 kept five AND gates (Section 6.5); Stage-4 post-processing reduced the archived 5/15/1, $D = 3$ netlist to 5/12/1, $D = 2$.
- **Inverse ($\mathsf{YSH}_{5,2}^{-1}$)**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [2^7, 3^8, 4^{16}]$; cost: 5 AND, 11 XOR, 2 NOT, AND depth $D = 2$ (Tier 2: Structural backward unrolling (three quadratic coordinates at depth 1, two quartic coordinates at depth 2); unmasked NanGate 45 nm: 23.94 μm^2 / 30.0 GE, delay 286.9 ps, logic depth 7; pipelined $d = 2$ DOM-indep: 40 `DFF_X1` [20 gadget + 20 sync], 270.26 μm^2 / 338.7 GE, stage delay 306.3 ps).

Lookup table (LUT[0 . . . 31]):

```
0x02, 0x0c, 0x14, 0x18, 0x0f, 0x05, 0x15, 0x19, 0x09, 0x03, 0x13, 0x1f, 0x00, 0x0e, 0x16, 0x1a
0x1e, 0x04, 0x0a, 0x12, 0x1b, 0x0d, 0x07, 0x17, 0x1c, 0x06, 0x08, 0x10, 0x1d, 0x0b, 0x01, 0x11
```

Netlist (5 AND, 12 XOR, 1 NOT, $D = 2$):

```
xor_0 = x[2] ^ x[3]; xor_1 = x[3] ^ x[1]; and_0 = x[3] & x[4];
xor_2 = x[0] ^ x[4]; xor_3 = xor_0 ^ and_0; and_1 = xor_2 & x[1];
xor_4 = x[1] ^ xor_2; xor_5 = xor_2 ^ xor_3; xor_6 = xor_1 ^ and_1;
not_0 = ~xor_6; and_2 = x[0] & xor_2; xor_7 = not_0 ^ x[0];
xor_8 = xor_4 ^ and_2; and_3 = xor_3 & xor_7; xor_9 = x[0] ^ xor_8;
and_4 = xor_8 & xor_3; xor_10 = xor_9 ^ and_3; xor_11 = xor_5 ^ and_4;
y[0] = xor_3; y[1] = xor_7; y[2] = xor_10; y[3] = xor_11; y[4] = xor_8;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 31]$):

```
0x0c, 0x1e, 0x00, 0x09, 0x11, 0x05, 0x19, 0x16, 0x1a, 0x08, 0x12, 0x1d, 0x01, 0x15, 0x0d, 0x04
0x1b, 0x1f, 0x13, 0x0a, 0x02, 0x06, 0x0e, 0x17, 0x03, 0x07, 0x0f, 0x14, 0x18, 0x1c, 0x10, 0x0b
```

Inverse netlist (5 AND, 11 XOR, 2 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[3]; and_0 = x[0] & x[1]; and_1 = x[3] & x[4];
and_2 = x[0] & x[4]; xor_1 = x[2] ^ and_0; xor_2 = x[4] ^ xor_1;
xor_3 = and_2 ^ xor_0; xor_4 = x[1] ^ and_1; xor_5 = xor_1 ^ xor_4;
xor_6 = xor_2 ^ xor_3; xor_7 = x[4] ^ xor_3; xor_8 = x[0] ^ xor_5;
not_0 = ~xor_5; and_3 = xor_2 & xor_3; and_4 = not_0 & xor_6;
xor_9 = and_3 ^ xor_7; xor_10 = and_4 ^ xor_8; not_1 = ~xor_10; y[0] = xor_2;
y[1] = xor_9; y[2] = not_1; y[3] = not_0; y[4] = xor_6;
```

C.5 5-bit Gold APN x^5 (5-bit, Category A)

- **Profile**: $\delta = 2$, $L = 8$ (NL = 12), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{31}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 7 AND, 22 XOR, 0 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 42.56 μm^2 / 53.3 GE, delay 311.2 ps, logic depth 6; pipelined $d = 2$ DOM-indep: 38 `DFF_X1` [28 gadget + 10 sync], 316.54 μm^2 / 396.7 GE, stage delay 288.5 ps).
- **Remark**: The exact lookup table of x^5 over $\mathbb{F}_2[x]/(x^5 + x^2 + 1)$ (the invariants do not separate x^3 and x^5); 7/24/1 in [BDM26b] and 7/19/1 in [BDM26a].
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{31}]$; cost: 10 AND, 37 XOR, 4 NOT, AND depth $D = 2$ (Tier 4: Hodge-dual linear-divisor coset-span $D = 2$ synthesis; unmasked NanGate 45 nm: 71.82 μm^2 / 90.0 GE, delay 595.9 ps, logic depth 11; pipelined $d = 2$ DOM-indep: 70 `DFF_X1` [40 gadget + 30 sync], 543.17 μm^2 / 680.7 GE, stage delay 353.4 ps).

Lookup table ($\mathrm{LUT}[0 \dots 31]$):

```
0x00, 0x01, 0x05, 0x16, 0x11, 0x19, 0x04, 0x1e, 0x1f, 0x18, 0x12, 0x07, 0x14, 0x1a, 0x09, 0x15
0x0c, 0x06, 0x17, 0x0f, 0x10, 0x13, 0x1b, 0x0a, 0x0e, 0x02, 0x1d, 0x03, 0x08, 0x0d, 0x0b, 0x1c
```

Netlist (7 AND, 22 XOR, 0 NOT, $D = 1$):

```
a_in = in_val[4:0]; split_2 = a_in[0:0]; split_3 = a_in[1:1];
split_4 = a_in[2:2]; split_5 = a_in[3:3]; split_6 = a_in[4:4];
xor_7 = split_2 ^ split_4; xor_9 = split_3 ^ split_4; and_14 = split_3 & split_5;
and_16 = split_4 & split_5; xor_26 = split_2 ^ split_5;
xor_8 = split_3 ^ split_6; xor_13 = split_4 ^ split_6;
and_15 = split_3 & split_6; xor_12 = split_5 ^ xor_7; and_17 = split_2 & xor_9;
xor_24 = split_3 ^ and_16; xor_10 = split_5 ^ xor_8; xor_11 = split_2 ^ xor_8;
xor_21 = split_5 ^ and_15; and_20 = xor_12 & xor_13; and_18 = split_2 & xor_10;
and_19 = xor_7 & xor_11; xor_22 = and_20 ^ xor_21; xor_27 = xor_26 ^ and_20;
xor_29 = and_16 ^ and_18; xor_31 = and_17 ^ and_18; xor_35 = split_2 ^ and_19;
xor_25 = xor_22 ^ xor_13; xor_28 = xor_24 ^ xor_27; xor_30 = xor_21 ^ xor_29;
xor_32 = xor_24 ^ xor_31; xor_36 = xor_35 ^ xor_22; xor_34 = and_14 ^ xor_25;
xor_33 = xor_32 ^ xor_25; concat_37 = { xor_36, xor_34, xor_33, xor_30, xor_28 };
out_val = concat_37;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 31]$):

```
0x00, 0x01, 0x19, 0x1b, 0x06, 0x02, 0x11, 0x0b, 0x1c, 0x0e, 0x17, 0x1e, 0x10, 0x1d, 0x18, 0x13
0x14, 0x04, 0x0a, 0x15, 0x0c, 0x0f, 0x03, 0x12, 0x09, 0x05, 0x0d, 0x16, 0x1f, 0x1a, 0x07, 0x08
```

Inverse netlist (10 AND, 37 XOR, 4 NOT, $D = 2$):

```
xor_0 = x[1] ^ x[4]; xor_1 = x[2] ^ xor_0; xor_2 = x[0] ^ x[3];
xor_3 = x[0] ^ x[1]; xor_4 = x[2] ^ xor_2; xor_5 = x[0] ^ x[4];
xor_6 = x[0] ^ x[2]; xor_7 = xor_3 ^ xor_4; and_0 = xor_2 & xor_1;
and_1 = xor_4 & xor_5; and_2 = xor_3 & xor_6; and_3 = xor_0 & x[3];
and_4 = xor_7 & x[4]; xor_8 = and_2 ^ and_3; xor_9 = and_4 ^ xor_1;
xor_10 = and_0 ^ and_2; xor_11 = x[0] ^ and_0; xor_12 = x[3] ^ xor_8;
xor_13 = x[4] ^ xor_12; xor_14 = x[4] ^ and_0; xor_15 = and_1 ^ xor_14;
xor_16 = xor_3 ^ and_1; xor_17 = xor_8 ^ xor_16; xor_18 = xor_9 ^ xor_10;
xor_19 = and_3 ^ xor_11; xor_20 = xor_3 ^ and_3; xor_21 = xor_8 ^ xor_11;
xor_22 = x[2] ^ xor_12; not_0 = ~xor_13; not_1 = ~xor_17; not_2 = ~xor_18;
not_3 = ~xor_19; and_5 = not_0 & x[0]; and_6 = xor_15 & x[1];
and_7 = not_1 & x[2]; and_8 = not_2 & x[3]; and_9 = not_3 & x[4];
xor_23 = and_1 ^ and_6; xor_24 = and_7 ^ xor_23; xor_25 = xor_9 ^ xor_24;
xor_26 = and_5 ^ and_6; xor_27 = xor_20 ^ xor_26; xor_28 = xor_21 ^ and_7;
xor_29 = xor_26 ^ xor_28; xor_30 = xor_22 ^ and_5; xor_31 = and_9 ^ xor_23;
xor_32 = xor_30 ^ xor_31; xor_33 = xor_10 ^ and_8; xor_34 = and_9 ^ xor_25;
xor_35 = xor_33 ^ xor_34; xor_36 = xor_2 ^ xor_25; y[0] = xor_27;
y[1] = xor_29; y[2] = xor_32; y[3] = xor_35; y[4] = xor_36;
```

C.6 5-bit Gold class ($\sim_{\text{aff}}$ x^5) with ideal avalanche, 7 AND (5-bit, Category A)

- **Profile**: $\delta = 2$, $L = 8$ (NL = 12), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{31}]$, FP = 1, SAC deviation 0.
- **Cost**: 7 AND, 21 XOR, 4 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 43.09 μm^2 / 54.0 GE, delay 282.0 ps, logic depth 6; pipelined $d = 2$ DOM-indep: 38 `DFF_X1` [28 gadget + 10 sync], 315.48 μm^2 / 395.3 GE, stage delay 243.3 ps).
- **Remark**: Affine-equivalent to x^5 (certified by the solver of Section 6.1: $S(x) = B\,S_{\text{ref}}(Ax \oplus a) \oplus b$ with $S_{\text{ref}}(u) = u^5 \bmod (x^5 + x^2 + 1)$, $A = [$`0x03`, `0x08`, `0x05`, `0x0a`, `0x12`$]$, $a =$ `0x01`, $B = [$`0x08`, `0x1b`, `0x18`, `0x1c`, `0x1e`$]$, $b =$ `0x16`), hence Category A; it is not affine-equivalent to x^3. Unlike the power map itself (SAC deviation 0.5), this affine representative satisfies the strict avalanche criterion exactly: each of the 25 single-bit input changes flips each output bit for exactly 16 of the 32 inputs, with one fixed point, at the same 7 AND gates and depth 1 as x^5 (7/21/4 against 7/22/0). Stage-1 seed: a power-map lookup-table node (`novel_power_map_5b`) that Stage 2 lowered and re-wired (Section 6.5); Stage-4 post-processing reduced the archived 7/23/4 netlist to 7/21/4. The inverse re-uses the x^{25} core of the Gold x^5 row inside the two affine layers (Tier 1).
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{31}]$; cost: 10 AND, 41 XOR, 10 NOT, AND depth $D = 2$ (Tier 1: Affine-folded reuse of the x^{25} inverse core of the Gold x^5 circuit; unmasked NanGate 45 nm: 81.40 μm^2 / 102.0 GE, delay 652.6 ps, logic depth 13; pipelined $d = 2$ DOM-indep: 86 `DFF_X1` [40 gadget + 46 sync], 631.48 μm^2 / 791.3 GE, stage delay 346.9 ps).

Lookup table (LUT$[0 \ldots 31]$):

```
0x1e, 0x06, 0x14, 0x13, 0x00, 0x17, 0x10, 0x18, 0x1d, 0x1f, 0x0b, 0x16, 0x02, 0x0f, 0x0e, 0x1c
0x01, 0x12, 0x05, 0x09, 0x07, 0x1b, 0x19, 0x1a, 0x0d, 0x04, 0x15, 0x03, 0x0a, 0x0c, 0x08, 0x11
```

Netlist (7 AND, 21 XOR, 4 NOT, $D = 1$):

```
and_0 = x[0] & x[1]; xor_0 = x[0] ^ x[2]; xor_1 = x[1] ^ x[2];
xor_2 = x[1] ^ x[3]; xor_3 = x[0] ^ x[3]; and_1 = x[2] & x[3];
xor_4 = x[3] ^ x[4]; xor_5 = x[2] ^ x[4]; and_2 = x[1] & x[4];
and_3 = xor_0 & x[4]; and_4 = xor_0 & xor_2; and_5 = xor_1 & xor_4;
and_6 = xor_3 & xor_5; xor_6 = xor_5 ^ and_3; xor_7 = and_4 ^ xor_5;
xor_8 = x[0] ^ and_5; xor_9 = x[1] ^ and_6; xor_10 = and_0 ^ and_6;
xor_11 = and_2 ^ xor_7; xor_12 = and_1 ^ xor_8; xor_13 = x[3] ^ xor_9;
xor_14 = xor_9 ^ xor_7; xor_15 = xor_4 ^ xor_10; xor_16 = and_5 ^ xor_10;
xor_17 = xor_8 ^ xor_11; xor_18 = xor_13 ^ xor_11; xor_19 = xor_12 ^ xor_14;
xor_20 = xor_6 ^ xor_16; not_0 = ~xor_17; not_1 = ~xor_18; not_2 = ~xor_19;
not_3 = ~xor_20; y[0] = xor_15; y[1] = not_1; y[2] = not_3; y[3] = not_2;
y[4] = not_0;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0\ldots31]$):

```
0x04, 0x10, 0x0c, 0x1b, 0x19, 0x12, 0x01, 0x14, 0x1e, 0x13, 0x1c, 0x0a, 0x1d, 0x18, 0x0e, 0x0d
0x06, 0x1f, 0x11, 0x03, 0x02, 0x1a, 0x0b, 0x05, 0x07, 0x16, 0x17, 0x15, 0x0f, 0x08, 0x00, 0x09
```

Inverse netlist (10 AND, 41 XOR, 10 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[1]; xor_1 = x[3] ^ x[4]; xor_2 = x[2] ^ xor_0;
xor_3 = xor_0 ^ xor_1; xor_4 = x[0] ^ xor_1; xor_5 = x[4] ^ xor_2;
xor_6 = x[0] ^ x[2]; xor_7 = xor_1 ^ xor_2; xor_8 = x[1] ^ x[3];
xor_9 = x[3] ^ xor_6; xor_10 = x[4] ^ xor_0; xor_11 = x[4] ^ xor_6;
xor_12 = x[1] ^ x[4]; not_0 = ~xor_7; not_1 = ~xor_5; not_2 = ~xor_8;
not_3 = ~xor_4; not_4 = ~xor_9; not_5 = ~x[1]; not_6 = ~xor_0; not_7 = ~xor_1;
and_0 = not_0 & not_1; and_1 = not_2 & xor_3; and_2 = not_3 & not_4;
and_3 = not_5 & xor_2; and_4 = xor_10 & not_6; xor_13 = and_2 ^ and_3;
xor_14 = and_1 ^ xor_0; xor_15 = x[2] ^ xor_13; xor_16 = and_0 ^ and_3;
xor_17 = and_0 ^ xor_14; xor_18 = xor_4 ^ and_1; xor_19 = xor_13 ^ xor_18;
xor_20 = xor_5 ^ and_0; xor_21 = and_2 ^ and_4; xor_22 = xor_20 ^ xor_21;
xor_23 = xor_1 ^ xor_16; xor_24 = x[3] ^ and_2; xor_25 = xor_3 ^ and_4;
not_8 = ~xor_17; and_5 = xor_15 & not_7; and_6 = not_8 & x[0];
and_7 = xor_19 & xor_11; and_8 = xor_22 & xor_2; and_9 = xor_23 & not_6;
xor_26 = and_1 ^ and_6; xor_27 = and_7 ^ xor_26; xor_28 = x[3] ^ xor_27;
xor_29 = and_5 ^ and_9; xor_30 = xor_14 ^ and_9; xor_31 = xor_24 ^ xor_30;
xor_32 = xor_15 ^ and_5; xor_33 = and_8 ^ xor_28; xor_34 = xor_32 ^ xor_33;
xor_35 = xor_13 ^ xor_26; xor_36 = xor_29 ^ xor_12; xor_37 = xor_35 ^ xor_36;
xor_38 = xor_16 ^ xor_29; xor_39 = xor_25 ^ xor_38; xor_40 = and_4 ^ xor_28;
not_9 = ~xor_39; y[0] = xor_31; y[1] = xor_34; y[2] = xor_37; y[3] = not_9;
y[4] = xor_40;
```

C.7 5-bit χ_5 class (5-bit, Category A)

- **Profile**: $\delta = 8$, $L = 16$ (NL = 8), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{31}]$, FP = 5, SAC deviation 0.5.
- **Cost**: 5 AND, 8 XOR, 1 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 18.62 μm^2 / 23.3 GE, delay 141.2 ps, logic depth 4; pipelined $d = 2$ DOM-indep: 28 `DFF_X1` [20 gadget + 8 sync], 205.88 μm^2 / 258.0 GE, stage delay 139.0 ps).
- **Remark**: Certified affine equivalent to Keccak χ_5 ($\sim_{\mathrm{aff}} \chi_5$, with $A = [$`0x01`, `0x04`, `0x02`, `0x0c`, `0x11`$]$, $a =$ `0x1f`, $B = [$`0x14`, `0x04`, `0x02`, `0x08`, `0x01`$]$, $b =$ `0x1b`), whose usual circuit has 5 AND, 5 XOR and 5 NOT gates [BDMD+20].
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{31}]$; cost: 9 AND, 21 XOR, 3 NOT, AND depth $D = 2$ (Tier 4: Hodge-dual linear-divisor coset-span $D = 2$ synthesis; unmasked NanGate 45 nm: 44.69 μm^2 / 56.0 GE, delay 329.7 ps, logic depth 8; pipelined $d = 2$ DOM-indep: 62 `DFF_X1` [36 gadget + 26 sync], 444.75 μm^2 / 557.3 GE, stage delay 258.1 ps).

Lookup table ($\mathrm{LUT}[0\ldots31]$):

```
0x00, 0x15, 0x06, 0x13, 0x10, 0x04, 0x02, 0x16, 0x08, 0x1d, 0x0a, 0x1f, 0x0c, 0x18, 0x1a, 0x0e
0x14, 0x09, 0x12, 0x0f, 0x05, 0x19, 0x17, 0x0b, 0x1e, 0x03, 0x1c, 0x01, 0x1b, 0x07, 0x0d, 0x11
```

Netlist (5 AND, 8 XOR, 1 NOT, $D = 1$):

```
xor_11 = x[1] ^ x[2]; xor_7 = x[1] ^ x[3]; and_10 = x[0] & x[4];
and_14 = x[3] & x[4]; xor_15 = x[0] ^ x[4]; not_0 = ~xor_7;
and_13 = xor_11 & xor_7; xor_19 = x[3] ^ and_10; xor_22 = x[1] ^ and_14;
and_16 = xor_15 & x[2]; and_9 = not_0 & x[2]; xor_23 = x[0] ^ and_16;
xor_21 = xor_15 ^ and_13; xor_18 = xor_15 ^ and_9; y[0] = xor_23; y[1] = xor_22;
y[2] = xor_21; y[3] = xor_19; y[4] = xor_18;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0\ldots31]$):

```
0x00, 0x1b, 0x06, 0x19, 0x05, 0x14, 0x02, 0x1d, 0x08, 0x11, 0x0a, 0x17, 0x0c, 0x1e, 0x0f, 0x13
0x04, 0x1f, 0x12, 0x03, 0x10, 0x01, 0x07, 0x16, 0x0d, 0x15, 0x0e, 0x1c, 0x1a, 0x09, 0x18, 0x0b
```

Inverse netlist (9 AND, 21 XOR, 3 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[1]; xor_1 = x[3] ^ x[4]; xor_2 = x[2] ^ x[3];
xor_3 = x[2] ^ x[4]; and_0 = x[0] & x[3]; and_1 = xor_0 & xor_1;
and_2 = xor_2 & x[1]; and_3 = x[2] & x[4]; xor_4 = and_1 ^ x[4];
xor_5 = and_0 ^ x[0]; xor_6 = xor_4 ^ xor_5; not_0 = ~xor_6;
xor_7 = xor_2 ^ and_2; xor_8 = xor_4 ^ xor_7; not_1 = ~xor_8;
xor_9 = and_3 ^ x[0]; not_2 = ~xor_9; xor_10 = and_1 ^ x[2];
xor_11 = xor_5 ^ xor_10; xor_12 = x[1] ^ x[3]; xor_13 = xor_12 ^ and_0;
and_4 = not_0 & x[1]; and_5 = not_1 & x[0]; and_6 = not_2 & x[3];
and_7 = xor_11 & x[4]; and_8 = xor_3 & xor_13; xor_14 = and_5 ^ x[3];
xor_15 = and_7 ^ x[0]; xor_16 = and_8 ^ x[2]; xor_17 = xor_15 ^ xor_16;
xor_18 = and_6 ^ x[1]; xor_19 = xor_14 ^ xor_18; xor_20 = xor_3 ^ and_4;
y[0] = xor_17; y[1] = xor_19; y[2] = xor_20; y[3] = xor_14; y[4] = xor_15;
```

C.8 5-bit inverse-Gold class (x^{25}) (5-bit, Category A)

- **Profile**: $\delta = 2$, $L = 8$ (NL = 12), $\deg(S) = 3$, $\mathcal{D}(S) = [3^{31}]$, FP = 1, SAC deviation 0.125.
- **Cost**: 14 AND, 31 XOR, 2 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 65.44 μm^2 / 82.0 GE, delay 439.6 ps, logic depth 8; pipelined $d = 2$ DOM-indep: 88 `DFF_X1` [56 gadget + 32 sync], 646.91 μm^2 / 810.7 GE, stage delay 346.2 ps).
- **Remark**: Certified affine equivalent to x^7 (and hence to $x^{25} = (x^5)^{-1}$, $\sim_{\text{aff}} x^{25}$, with $S_{\text{ref}}(u) = u^7 \bmod (x^5 + x^2 + 1)$, $A =$ [`0x02`, `0x01`, `0x08`, `0x07`, `0x17`], $a =$ `0x01`, $B =$ [`0x0c`, `0x1b`, `0x15`, `0x0f`, `0x10`], $b =$ `0x14`), and not affine equivalent to x^{11} (x^{21}). Stage-1 seed: a power-map lookup-table node (`novel_power_map_5b`); Stage-4 post-processing reduced the archived 14/32/2 netlist to 14/31/2. Compared with the 10/37/2 circuit of [BDM26b] it uses four more AND gates and six fewer XOR gates. The inverse is quadratic ($\sim_{\text{aff}} x^5$) and re-uses the 7-AND Gold x^5 core of SM C.5 inside two affine layers (Tier 1).
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 2$, $\mathcal{D}(S^{-1}) = [2^{31}]$; cost: 7 AND, 26 XOR, 5 NOT, AND depth $D = 1$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 51.60 μm^2 / 64.7 GE, delay 485.6 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 38 `DFF_X1` [28 gadget + 10 sync], 331.97 μm^2 / 416.0 GE, stage delay 358.2 ps).

Lookup table (LUT$[0 \dots 31]$):

```
0x18, 0x01, 0x14, 0x11, 0x13, 0x1b, 0x0b, 0x16, 0x10, 0x12, 0x09, 0x04, 0x1d, 0x1c, 0x00, 0x07
0x08, 0x03, 0x0c, 0x1f, 0x1e, 0x17, 0x15, 0x0d, 0x02, 0x06, 0x0a, 0x05, 0x0e, 0x1a, 0x19, 0x0f
```

Netlist (14 AND, 31 XOR, 2 NOT, $D = 2$):

```
xor_0 = x[1] ^ x[3]; xor_1 = x[0] ^ x[3]; and_0 = x[0] & x[1];
and_1 = x[0] & x[2]; and_2 = x[0] & x[3]; and_3 = x[1] & x[3];
and_4 = x[2] & x[3]; and_5 = x[0] & x[4]; and_6 = x[1] & x[4];
and_7 = x[2] & x[4]; and_8 = x[3] & x[4]; xor_2 = x[0] ^ and_7;
xor_3 = x[1] ^ and_4; xor_4 = and_0 ^ and_3; xor_5 = and_1 ^ and_6;
xor_6 = x[0] ^ and_3; xor_7 = x[2] ^ and_6; xor_8 = and_0 ^ and_7;
xor_9 = x[1] ^ and_8; xor_10 = x[2] ^ and_5; xor_11 = and_3 ^ xor_5;
xor_12 = and_4 ^ and_8; xor_13 = and_7 ^ xor_3; xor_14 = x[4] ^ xor_2;
and_9 = xor_6 & xor_7; and_10 = xor_0 & xor_5; and_11 = xor_1 & xor_8;
and_12 = xor_2 & xor_3; and_13 = xor_9 & xor_10; xor_15 = and_2 ^ and_12;
xor_16 = x[2] ^ xor_15; xor_17 = and_5 ^ and_11; xor_18 = and_10 ^ xor_2;
xor_19 = and_13 ^ xor_4; xor_20 = xor_16 ^ xor_18; xor_21 = and_11 ^ xor_20;
xor_22 = xor_11 ^ xor_21; xor_23 = xor_16 ^ xor_17; xor_24 = xor_12 ^ xor_23;
xor_25 = xor_19 ^ xor_13; xor_26 = xor_0 ^ xor_20; xor_27 = and_9 ^ xor_15;
xor_28 = xor_17 ^ xor_19; xor_29 = xor_14 ^ xor_27; xor_30 = xor_28 ^ xor_29;
not_0 = ~xor_26; not_1 = ~xor_30; y[0] = xor_22; y[1] = xor_24; y[2] = xor_25;
y[3] = not_0; y[4] = not_1;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 31]$):

```
0x0e, 0x01, 0x18, 0x11, 0x0b, 0x1b, 0x19, 0x0f, 0x10, 0x0a, 0x1a, 0x06, 0x12, 0x17, 0x1c, 0x1f
0x08, 0x03, 0x09, 0x04, 0x02, 0x16, 0x07, 0x15, 0x00, 0x1e, 0x1d, 0x05, 0x0d, 0x0c, 0x14, 0x13
```

Inverse netlist (7 AND, 26 XOR, 5 NOT, $D = 1$):

```
xor_0 = x[1] ^ x[2]; xor_1 = x[3] ^ x[4]; xor_2 = x[3] ^ xor_0;
xor_3 = x[4] ^ xor_0; xor_4 = x[2] ^ xor_1; xor_5 = x[2] ^ x[4];
xor_6 = xor_0 ^ xor_1; xor_7 = x[0] ^ xor_0; xor_8 = x[0] ^ x[1];
xor_9 = x[2] ^ x[3]; xor_10 = x[1] ^ x[3]; not_0 = ~xor_2; not_1 = ~xor_7;
not_2 = ~xor_0; not_3 = ~xor_9; and_0 = xor_3 & x[0]; and_1 = x[3] & x[0];
and_2 = xor_3 & xor_5; and_3 = not_0 & xor_6; and_4 = not_1 & xor_4;
and_5 = not_0 & xor_8; and_6 = not_2 & not_3; xor_11 = and_0 ^ and_4;
xor_12 = and_1 ^ and_6; xor_13 = and_5 ^ xor_11; xor_14 = and_3 ^ xor_12;
xor_15 = and_2 ^ xor_13; xor_16 = x[0] ^ xor_14; xor_17 = xor_4 ^ and_1;
xor_18 = xor_13 ^ xor_17; xor_19 = x[2] ^ and_5; xor_20 = xor_14 ^ xor_19;
xor_21 = and_0 ^ xor_16; xor_22 = and_6 ^ xor_15; xor_23 = xor_10 ^ xor_22;
xor_24 = xor_1 ^ xor_15; xor_25 = xor_16 ^ xor_24; not_4 = ~xor_25;
y[0] = xor_18; y[1] = xor_20; y[2] = xor_21; y[3] = xor_23; y[4] = not_4;
```

C.9 5-bit x^{15} (inversion class) (5-bit, Category A)

- **Profile**: $\delta = 2$, $L = 12$ (NL $= 10$), $\deg(S) = 4$, $\mathcal{D}(S) = [4^{31}]$, FP $= 2$, SAC deviation 0.125.
- **Cost**: 14 AND, 45 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 86.72 μm^2 / 108.7 GE, delay 710.9 ps, logic depth 13; pipelined $d = 2$ DOM-indep: 76 `DFF_X1` [56 gadget + 20 sync], 636.27 μm^2 / 797.3 GE, stage delay 518.9 ps).
- **Remark**: The exact lookup table of x^{15} over $\mathbb{F}_2[x]/(x^5 + x^2 + 1)$, which is linearly equivalent to the inversion $x^{30} = (x^{15})^2$; 12/51/2 in [BDM26b]: two more AND gates and six fewer XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [4^{31}]$; cost: 14 AND, 57 XOR, 0 NOT, AND depth $D = 2$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 105.87 μm^2 / 132.7 GE, delay 794.1 ps, logic depth 14; pipelined $d = 2$ DOM-indep: 88 `DFF_X1` [56 gadget + 32 sync], 728.84 μm^2 / 913.3 GE, stage delay 517.0 ps).

Lookup table (LUT$[0 \dots 31]$):

```
0x00, 0x01, 0x1f, 0x15, 0x12, 0x1c, 0x0a, 0x11, 0x1d, 0x02, 0x16, 0x04, 0x18, 0x0b, 0x19, 0x10
0x09, 0x17, 0x1b, 0x14, 0x0e, 0x0c, 0x13, 0x03, 0x05, 0x08, 0x07, 0x0f, 0x1a, 0x1e, 0x06, 0x0d
```

Netlist (14 AND, 45 XOR, 0 NOT, $D = 2$):

```
a_in = in_val[4:0]; split_2 = a_in[0:0]; split_3 = a_in[1:1];
split_4 = a_in[2:2]; split_5 = a_in[3:3]; split_6 = a_in[4:4];
xor_7 = split_2 ^ split_3; xor_8 = split_2 ^ split_4; xor_9 = split_2 ^ split_5;
and_14 = split_4 & split_5; xor_21 = split_4 ^ split_5;
xor_11 = split_4 ^ split_6; and_13 = split_3 & split_6; xor_10 = split_6 ^ xor_7;
xor_12 = split_5 ^ xor_8; and_16 = xor_8 & split_6; and_15 = split_3 & xor_9;
xor_22 = split_6 ^ xor_21; and_18 = xor_9 & xor_11; xor_33 = split_5 ^ and_13;
and_17 = split_5 & xor_10; and_19 = xor_7 & xor_12; xor_23 = split_2 ^ and_16;
xor_30 = split_3 ^ xor_22; xor_20 = and_15 ^ and_17; xor_27 = split_6 ^ and_17;
xor_28 = and_19 ^ xor_23; xor_34 = and_14 ^ xor_23; xor_31 = xor_30 ^ and_14;
xor_24 = and_19 ^ xor_20; xor_26 = xor_22 ^ xor_20; xor_29 = xor_27 ^ xor_28;
xor_32 = and_16 ^ xor_31; xor_25 = and_18 ^ xor_24; xor_35 = xor_33 ^ xor_24;
xor_39 = xor_26 ^ xor_29; and_44 = xor_26 & xor_32; and_46 = xor_29 & xor_32;
xor_37 = xor_25 ^ xor_29; xor_56 = xor_25 ^ xor_32; xor_36 = xor_34 ^ xor_35;
and_47 = xor_25 & xor_39; xor_54 = xor_26 ^ and_46; xor_42 = xor_32 ^ xor_37;
xor_38 = xor_26 ^ xor_36; xor_43 = xor_29 ^ xor_36; and_45 = xor_26 & xor_36;
xor_40 = xor_32 ^ xor_38; xor_41 = xor_25 ^ xor_38; and_50 = xor_42 & xor_43;
xor_51 = xor_32 ^ and_45; and_48 = xor_25 & xor_40; and_49 = xor_37 & xor_41;
xor_57 = xor_56 ^ and_50; xor_52 = and_50 ^ xor_51; xor_59 = and_46 ^ and_48;
xor_61 = and_47 ^ and_48; xor_65 = xor_25 ^ and_49; xor_58 = xor_54 ^ xor_57;
xor_55 = xor_52 ^ xor_43; xor_60 = xor_51 ^ xor_59; xor_62 = xor_54 ^ xor_61;
xor_66 = xor_65 ^ xor_52; xor_64 = and_44 ^ xor_55; xor_63 = xor_62 ^ xor_55;
concat_67 = { xor_66, xor_64, xor_63, xor_60, xor_58 }; out_val = concat_67;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 31]$):

```
0x00, 0x01, 0x09, 0x17, 0x0b, 0x18, 0x1e, 0x1a, 0x19, 0x10, 0x06, 0x0d, 0x15, 0x1f, 0x14, 0x1b
0x0f, 0x07, 0x04, 0x16, 0x13, 0x03, 0x0a, 0x11, 0x0c, 0x0e, 0x1c, 0x12, 0x05, 0x08, 0x1d, 0x02
```

Inverse netlist (14 AND, 57 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[2] ^ x[3]; xor_1 = x[0] ^ x[1]; xor_2 = xor_0 ^ x[4];
xor_3 = x[0] ^ x[2]; xor_4 = x[0] ^ x[3]; xor_5 = x[2] ^ x[4];
xor_6 = xor_1 ^ x[4]; xor_7 = xor_0 ^ x[0]; xor_8 = xor_2 ^ x[1];
and_0 = x[2] & x[3]; and_1 = x[1] & x[4]; and_2 = xor_3 & x[4];
and_3 = xor_4 & x[1]; and_4 = xor_4 & xor_5; xor_9 = and_1 ^ x[3];
and_5 = xor_6 & x[3]; and_6 = xor_1 & xor_7; xor_10 = and_2 ^ x[0];
xor_11 = and_3 ^ and_5; xor_12 = and_5 ^ x[4]; xor_13 = and_6 ^ xor_10;
xor_14 = and_0 ^ xor_10; xor_15 = and_0 ^ xor_8; xor_16 = and_6 ^ xor_11;
xor_17 = xor_2 ^ xor_11; xor_18 = xor_12 ^ xor_13; xor_19 = and_2 ^ xor_15;
xor_20 = and_4 ^ xor_16; xor_21 = xor_16 ^ xor_9; xor_22 = xor_17 ^ xor_18;
and_7 = xor_17 & xor_19; and_8 = xor_18 & xor_19; xor_23 = xor_20 ^ xor_18;
xor_24 = xor_14 ^ xor_21; and_9 = xor_20 & xor_22; xor_25 = xor_19 ^ xor_23;
xor_26 = xor_17 ^ xor_24; xor_27 = xor_18 ^ xor_24; and_10 = xor_17 & xor_24;
xor_28 = xor_19 ^ xor_26; xor_29 = xor_20 ^ xor_26; and_11 = xor_25 & xor_27;
and_12 = xor_20 & xor_28; and_13 = xor_23 & xor_29; xor_30 = and_9 ^ and_12;
xor_31 = and_4 ^ and_6; xor_32 = and_13 ^ and_11; xor_33 = and_3 ^ x[2];
xor_34 = x[3] ^ x[4]; xor_35 = and_5 ^ and_7; xor_36 = xor_30 ^ xor_33;
xor_37 = xor_34 ^ xor_35; xor_38 = and_2 ^ x[1]; xor_39 = and_0 ^ xor_31;
xor_40 = xor_32 ^ xor_38; xor_41 = and_1 ^ xor_36; xor_42 = xor_40 ^ xor_41;
xor_43 = and_1 ^ and_13; xor_44 = xor_43 ^ xor_37; xor_45 = xor_44 ^ xor_39;
xor_46 = and_8 ^ xor_36; xor_47 = xor_46 ^ xor_37; xor_48 = and_7 ^ and_8;
xor_49 = and_10 ^ xor_48; xor_50 = xor_30 ^ xor_49; xor_51 = xor_50 ^ xor_39;
xor_52 = xor_40 ^ xor_51; xor_53 = and_8 ^ xor_11; xor_54 = and_12 ^ xor_53;
xor_55 = xor_31 ^ xor_54; xor_56 = xor_32 ^ xor_55; y[0] = xor_42;
y[1] = xor_45; y[2] = xor_47; y[3] = xor_52; y[4] = xor_56;
```

C.10 $\mathsf{YSH}_{6,1}$: 6-bit quartic, $[3^7, 4^{56}]$, $\delta = 4$, $D = 3$ (6-bit, Category B)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 24), $\deg(S) = 4$, $\mathcal{D}(S) = \mathcal{D}(S^{-1}) = [3^7, 4^{56}]$, FP = 0, SAC deviation 0.125.
- **Cost**: 9 AND, 27 XOR, 1 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 53.20 μm^2 / 66.7 GE, delay 916.7 ps, logic depth 16; pipelined $d = 2$ DOM-indep: 74 `DFF_X1` [36 gadget + 38 sync], 517.10 μm^2 / 648.0 GE, stage delay 561.1 ps).
- **Remark**: Balanced 3-round generalized Feistel permutation with an asymmetric degree-3 middle round: degree 4 at $\delta = 4$, $L = 16$ with 9 AND gates (3 at each of the three multiplicative levels) and minimum component degree $d_{\min}(S) = d_{\min}(S^{-1}) = 3$, eliminating all quadratic components ($\mathcal{D}(S) = \mathcal{D}(S^{-1}) = [3^7, 4^{56}]$, matching Dillon's APN degree spectrum), while inheriting the structural 3-round Feistel boomerang uniformity $\beta = 64$ (Section 6.6).
- **Inverse ($\mathsf{YSH}_{6,1}^{-1}$)**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [3^7, 4^{56}]$; cost: 9 AND, 27 XOR, 1 NOT, AND depth $D = 3$ (Tier 2: Exact 3-round generalized Feistel layer reversal; unmasked NanGate 45 nm: 53.20 μm^2 / 66.7 GE, delay 764.6 ps, logic depth 13; pipelined $d = 2$ DOM-indep: 78 `DFF_X1` [36 gadget + 42 sync], 535.19 μm^2 / 670.7 GE, stage delay 407.4 ps).

Lookup table (LUT[0 . . . 63]):

```
0x18, 0x32, 0x0d, 0x05, 0x3a, 0x10, 0x02, 0x2f, 0x31, 0x28, 0x19, 0x3d, 0x30, 0x08, 0x1b, 0x25
0x38, 0x20, 0x09, 0x00, 0x11, 0x2e, 0x0c, 0x1a, 0x23, 0x14, 0x2c, 0x15, 0x36, 0x07, 0x3b, 0x22
0x0a, 0x39, 0x1e, 0x2d, 0x0b, 0x26, 0x16, 0x01, 0x1f, 0x03, 0x34, 0x37, 0x3f, 0x1c, 0x33, 0x2b
0x0e, 0x12, 0x3c, 0x13, 0x04, 0x21, 0x3e, 0x0f, 0x29, 0x06, 0x27, 0x24, 0x1d, 0x2a, 0x35, 0x17
```

Netlist (9 AND, 27 XOR, 1 NOT, $D = 3$):

```
and_0 = x[0] & x[1]; and_1 = x[1] & x[2]; and_2 = x[2] & x[0];
xor_0 = x[4] ^ and_1; xor_1 = x[5] ^ and_2; xor_2 = xor_0 ^ x[0];
xor_3 = xor_0 ^ x[2]; xor_4 = xor_0 ^ x[3]; xor_5 = xor_2 ^ and_0;
xor_6 = xor_2 ^ x[2]; xor_7 = xor_4 ^ xor_5; xor_8 = xor_7 ^ x[1];
xor_9 = xor_1 ^ xor_8; and_3 = xor_2 & xor_9; not_0 = ~xor_9;
xor_10 = xor_7 ^ xor_6; xor_11 = not_0 ^ x[1]; and_4 = xor_9 & xor_10;
xor_12 = x[2] ^ and_4; xor_13 = xor_11 ^ xor_3; xor_14 = xor_12 ^ xor_2;
xor_15 = xor_13 ^ and_3; xor_16 = xor_15 ^ xor_10; and_5 = xor_16 & xor_14;
and_6 = xor_10 & xor_2; xor_17 = xor_0 ^ and_6; xor_18 = xor_17 ^ xor_9;
and_7 = xor_14 & xor_18; xor_19 = not_0 ^ and_7; and_8 = xor_18 & xor_16;
xor_20 = xor_10 ^ and_8; xor_21 = xor_19 ^ xor_16; xor_22 = xor_20 ^ xor_14;
xor_23 = xor_21 ^ xor_2; xor_24 = xor_19 ^ xor_18; xor_25 = xor_23 ^ and_5;
xor_26 = xor_25 ^ xor_18; y[0] = xor_22; y[1] = xor_26; y[2] = xor_21;
y[3] = xor_24; y[4] = xor_19; y[5] = xor_14;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0\ldots63]$):

```
0x13, 0x27, 0x06, 0x29, 0x34, 0x03, 0x39, 0x1d, 0x0d, 0x12, 0x20, 0x24, 0x16, 0x02, 0x30, 0x37
0x05, 0x14, 0x31, 0x33, 0x19, 0x1b, 0x26, 0x3f, 0x00, 0x0a, 0x17, 0x0e, 0x2d, 0x3c, 0x22, 0x28
0x11, 0x35, 0x1f, 0x18, 0x3b, 0x0f, 0x25, 0x3a, 0x09, 0x38, 0x3d, 0x2f, 0x1a, 0x23, 0x15, 0x07
0x0c, 0x08, 0x01, 0x2e, 0x2a, 0x3e, 0x1c, 0x2b, 0x10, 0x21, 0x04, 0x1e, 0x32, 0x0b, 0x36, 0x2c
```

Inverse netlist (9 AND, 27 XOR, 1 NOT, $D = 3$):

```
xor_0 = x[2] ^ x[4]; xor_1 = x[3] ^ x[4]; and_0 = x[5] & xor_0;
and_1 = x[5] & xor_1; and_2 = xor_1 & xor_0; xor_2 = x[0] ^ x[5];
xor_3 = and_2 ^ xor_2; xor_4 = and_1 ^ x[4]; not_0 = ~xor_4;
xor_5 = x[1] ^ xor_1; xor_6 = and_0 ^ xor_5; xor_7 = x[2] ^ xor_6;
and_3 = xor_7 & not_0; and_4 = xor_3 & not_0; and_5 = xor_3 & xor_7;
xor_8 = not_0 ^ xor_1; xor_9 = and_5 ^ xor_8; xor_10 = x[5] ^ xor_7;
xor_11 = and_4 ^ xor_10; xor_12 = xor_3 ^ xor_0; xor_13 = and_3 ^ xor_12;
xor_14 = xor_11 ^ xor_9; xor_15 = xor_14 ^ xor_13; xor_16 = xor_15 ^ xor_4;
xor_17 = xor_7 ^ xor_9; and_6 = xor_17 & xor_16; and_7 = xor_16 & xor_11;
and_8 = xor_17 & xor_11; xor_18 = and_7 ^ xor_9; xor_19 = xor_11 ^ xor_7;
xor_20 = xor_3 ^ xor_19; xor_21 = and_6 ^ xor_7; xor_22 = xor_20 ^ xor_21;
xor_23 = xor_9 ^ xor_22; xor_24 = xor_16 ^ xor_20; xor_25 = xor_24 ^ not_0;
xor_26 = and_8 ^ xor_25; y[0] = xor_17; y[1] = xor_16; y[2] = xor_11;
y[3] = xor_23; y[4] = xor_18; y[5] = xor_26;
```

C.11 $\mathsf{YSH}_{6,2}$: 6-bit quartic, $[3^{31}, 4^{32}]$, $\delta = 4$, $D = 3$ (6-bit, Category B)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 24), $\deg(S) = 4$, $\mathcal{D}(S) = [3^{31}, 4^{32}]$, FP = 2, SAC deviation 0.25.
- **Cost**: 27 AND, 64 XOR, 0 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 130.87 μm^2 / 164.0 GE, delay 502.4 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 172 `DFF_X1` [108 gadget + 64 sync], 1269.35 μm^2 / 1590.7 GE, stage delay 434.1 ps).
- **Remark**: Bipartite spectrum $[3^{31}, 4^{32}]$ at $\delta = 4$, $L = 16$. $\mathsf{YSH}_{6,1}$ (SM C.10) has the same $(\delta, L, \deg, D)$ with far fewer forward gates (9 AND, 27 XOR) and a $k = 3$ spectrum $[3^7, 4^{56}]$, whereas $\mathsf{YSH}_{6,2}^{-1}$ unwinds to depth $D^{-1} = 2$.
- **Inverse ($\mathsf{YSH}_{6,2}^{-1}$)**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [2^{15}, 3^{48}]$; cost: 9 AND, 37 XOR, 0 NOT, AND depth $D = 2$ (Tier 2: Output shear unwinding + 8-AND Q_{2258} quadratic inverse core; unmasked NanGate 45 nm: 68.63 μm^2 / 86.0 GE, delay 468.6 ps, logic depth 8; pipelined $d = 2$ DOM-indep: 76 `DFF_X1` [36 gadget + 40 sync], 557.54 μm^2 / 698.7 GE, stage delay 387.3 ps).

Lookup table (LUT$[0\ldots63]$):

```
0x00, 0x01, 0x07, 0x02, 0x03, 0x04, 0x05, 0x06, 0x0b, 0x08, 0x1f, 0x1e, 0x2d, 0x2a, 0x39, 0x3c
0x3d, 0x35, 0x38, 0x33, 0x16, 0x1b, 0x14, 0x1a, 0x12, 0x2e, 0x29, 0x10, 0x0c, 0x37, 0x31, 0x0f
0x19, 0x17, 0x26, 0x2c, 0x18, 0x15, 0x22, 0x2b, 0x24, 0x3a, 0x11, 0x0d, 0x3f, 0x20, 0x0e, 0x13
0x2f, 0x23, 0x34, 0x3b, 0x32, 0x3e, 0x28, 0x27, 0x36, 0x1c, 0x0a, 0x25, 0x21, 0x09, 0x1d, 0x30
```

Netlist (27 AND, 64 XOR, 0 NOT, $D = 3$):

```
a_in = in_val[5:0];
  split_2 = a_in[0:0]; split_3 = a_in[1:1]; split_4 = a_in[2:2];
  split_5 = a_in[3:3]; split_6 = a_in[4:4]; split_7 = a_in[5:5];
  and_8 = split_2 & split_3; and_9 = split_2 & split_4; and_13 = split_3 & split_4;
  paar_shared_xor_1_split_3_split_4 = split_3 ^ split_4;
  and_10 = split_2 & split_5; and_14 = split_3 & split_5;
  and_17 = split_4 & split_5; and_11 = split_2 & split_6;
  and_15 = split_3 & split_6; and_18 = split_4 & split_6;
  and_20 = split_5 & split_6; xor_34 = split_3 ^ split_6;
  xor_49 = split_5 ^ split_6; and_12 = split_2 & split_7;
  and_16 = split_3 & split_7; and_19 = split_4 & split_7;
  and_21 = split_5 & split_7; and_22 = split_6 & split_7;
  xor_43 = split_4 ^ split_7; xor_83 = split_3 ^ and_14; xor_29 = split_3 ^ and_17;
  xor_37 = split_3 ^ and_11; xor_44 = and_10 ^ and_15; xor_25 = split_2 ^ and_18;
  xor_38 = and_10 ^ and_18; xor_26 = split_7 ^ and_20; xor_32 = and_11 ^ and_12;
  xor_41 = and_12 ^ and_18; xor_50 = and_12 ^ and_14; xor_55 = split_6 ^ and_12;
```

```
paar_shared_xor_0_and_12_and_17 = and_12 ^ and_17; xor_47 = and_10 ^ and_16;
xor_57 = and_14 ^ and_16; xor_28 = split_2 ^ and_19; xor_35 = and_17 ^ and_19;
xor_52 = split_6 ^ and_19; xor_46 = split_4 ^ and_21; xor_40 = split_3 ^ and_22;
xor_65 = split_5 ^ and_22; xor_73 = split_2 ^ paar_shared_xor_1_split_3_split_4;
xor_79 = split_6 ^ paar_shared_xor_1_split_3_split_4; and_45 = xor_43 & xor_44;
and_39 = xor_37 & xor_38; and_27 = xor_25 & xor_26;
and_33 = paar_shared_xor_1_split_3_split_4 & xor_32; and_51 = xor_49 & xor_50;
xor_60 = and_18 ^ xor_57; xor_91 = and_21 ^ xor_57; and_30 = xor_28 & xor_29;
and_36 = xor_34 & xor_35; and_48 = xor_46 & xor_47; and_42 = xor_40 & xor_41;
xor_74 = xor_73 ^ and_13; xor_67 = and_20 ^ and_45; xor_66 = and_9 ^ and_39;
xor_84 = xor_83 ^ and_33; xor_61 = and_17 ^ and_51;
and_24 = split_3 & paar_shared_xor_0_and_12_and_17;
and_54 = xor_52 & paar_shared_xor_0_and_12_and_17; xor_56 = and_30 ^ xor_55;
xor_94 = xor_60 ^ and_36; xor_58 = split_7 ^ and_48; xor_62 = and_21 ^ and_42;
xor_75 = xor_74 ^ and_33; xor_85 = xor_61 ^ xor_67; xor_68 = and_24 ^ xor_61;
xor_80 = xor_79 ^ and_54; xor_59 = and_8 ^ xor_56; xor_63 = and_11 ^ xor_58;
xor_92 = xor_91 ^ xor_58; xor_69 = xor_62 ^ xor_66; xor_95 = xor_62 ^ xor_67;
xor_87 = xor_84 ^ xor_85; xor_71 = and_27 ^ xor_59; xor_72 = xor_59 ^ xor_65;
xor_70 = xor_60 ^ xor_63; xor_76 = xor_63 ^ xor_68; xor_86 = xor_56 ^ xor_69;
xor_96 = xor_68 ^ xor_95; xor_93 = xor_92 ^ xor_71; xor_81 = xor_69 ^ xor_70;
xor_89 = and_51 ^ xor_70; xor_77 = xor_76 ^ xor_72; xor_88 = xor_86 ^ xor_87;
xor_97 = xor_94 ^ xor_96; xor_82 = xor_80 ^ xor_81; xor_90 = xor_72 ^ xor_89;
xor_78 = xor_75 ^ xor_77; xor_98 = xor_71 ^ xor_97;
and_shear = xor_78 & xor_90; xor_shear = xor_82 ^ and_shear;
concat_99 = { xor_98, xor_93, xor_90, xor_88, xor_shear, xor_78 };
out_val = concat_99;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x01, 0x03, 0x04, 0x05, 0x06, 0x07, 0x02, 0x09, 0x3d, 0x3a, 0x08, 0x1c, 0x2b, 0x2e, 0x1f
0x1b, 0x2a, 0x18, 0x2f, 0x16, 0x25, 0x14, 0x21, 0x24, 0x20, 0x17, 0x15, 0x39, 0x3e, 0x0b, 0x0a
0x2d, 0x3c, 0x26, 0x31, 0x28, 0x3b, 0x22, 0x37, 0x36, 0x1a, 0x0d, 0x27, 0x23, 0x0c, 0x19, 0x30
0x3f, 0x1e, 0x34, 0x13, 0x32, 0x11, 0x38, 0x1d, 0x12, 0x0e, 0x29, 0x33, 0x0f, 0x10, 0x35, 0x2c
```

Inverse netlist (9 AND, 37 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[3] ^ x[4]; xor_1 = x[0] ^ xor_0; xor_2 = x[2] ^ x[5];
xor_3 = x[4] ^ xor_2; xor_4 = x[3] ^ xor_2; xor_5 = x[4] ^ x[5];
xor_6 = x[5] ^ xor_1; xor_7 = xor_1 ^ xor_2; xor_8 = x[2] ^ x[3];
xor_9 = x[0] ^ x[5]; xor_10 = x[0] ^ x[2]; and_0 = x[0] & x[3];
and_1 = xor_1 & xor_4; and_2 = xor_5 & xor_6; and_3 = xor_0 & xor_7;
xor_11 = x[1] ^ and_0; xor_12 = x[5] ^ xor_11; xor_13 = x[2] ^ xor_11;
xor_14 = x[3] ^ xor_12; xor_15 = xor_11 ^ xor_1; xor_16 = x[0] ^ xor_14;
xor_17 = xor_3 ^ xor_11; xor_18 = x[3] ^ xor_15; xor_19 = xor_0 ^ xor_13;
xor_20 = and_2 ^ xor_12; xor_21 = and_3 ^ xor_15; and_4 = x[3] & xor_13;
and_5 = xor_8 & xor_15; and_6 = xor_9 & xor_16; and_7 = x[0] & xor_17;
and_8 = xor_18 & xor_19; xor_22 = and_3 ^ and_7; xor_23 = and_1 ^ xor_22;
xor_24 = and_2 ^ and_4; xor_25 = and_5 ^ and_8; xor_26 = xor_10 ^ and_8;
xor_27 = xor_22 ^ xor_24; xor_28 = xor_26 ^ xor_27; xor_29 = xor_20 ^ and_7;
xor_30 = xor_3 ^ xor_23; xor_31 = xor_21 ^ and_6; xor_32 = xor_25 ^ xor_31;
xor_33 = x[5] ^ xor_24; xor_34 = xor_14 ^ and_4; xor_35 = xor_23 ^ xor_25;
xor_36 = xor_34 ^ xor_35; y[0] = xor_28; y[1] = xor_29; y[2] = xor_30;
y[3] = xor_32; y[4] = xor_33; y[5] = xor_36;
```

C.12 $\mathsf{YSH}_{6,3}$: 6-bit cubic, $\delta = 4$, 9 AND (6-bit, Category B)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 24), $\deg(S) = 3$, $\mathcal{D}(S) = [3^{63}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 9 AND, 17 XOR, 9 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 41.50 μm^2 / 52.0 GE, delay 411.0 ps, logic depth 11; pipelined $d = 2$ DOM-indep: 72 `DFF_X1` [36 gadget + 36 sync], 480.40 μm^2 / 602.0 GE, stage delay 254.4 ps).
- **Remark**: Same $\mathcal{D}(S) = [3^{63}]$ as the x^{13} class, from which the DDT and |LAT| histograms separate it. The circuit of SM C.17 has the same $(\delta, L, \deg, D)$ and 9 AND gates with two fewer XOR gates.
- **Inverse ($\mathsf{YSH}_{6,3}^{-1}$)**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [3^{31}, 4^{32}]$; cost: 9 AND, 17 XOR, 8 NOT, AND depth $D = 3$ (Tier 2: Output shear unwinding + 3-round χ_3-Feistel reversal; unmasked NanGate 45 nm: 40.96 μm^2 / 51.3 GE, delay 399.2 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 72 `DFF_X1` [36 gadget + 36 sync], 479.86 μm^2 / 601.3 GE, stage delay 317.0 ps).

Lookup table (LUT[0 . . . 63]):

```
0x00, 0x03, 0x06, 0x01, 0x05, 0x04, 0x02, 0x07, 0x19, 0x3c, 0x2b, 0x18, 0x0f, 0x0e, 0x3d, 0x2a
0x30, 0x17, 0x3a, 0x15, 0x1f, 0x38, 0x32, 0x1d, 0x08, 0x0b, 0x37, 0x2e, 0x34, 0x11, 0x2d, 0x12
0x28, 0x35, 0x27, 0x3b, 0x3f, 0x2c, 0x23, 0x31, 0x20, 0x1e, 0x1b, 0x36, 0x25, 0x33, 0x0c, 0x09
0x10, 0x29, 0x16, 0x22, 0x2f, 0x1a, 0x1c, 0x24, 0x39, 0x21, 0x0a, 0x0d, 0x14, 0x26, 0x13, 0x3e
```

Netlist (9 AND, 17 XOR, 9 NOT, $D = 3$):

```
not_0 = ~x[0]; not_1 = ~x[1]; not_2 = ~x[2]; and_0 = not_0 & x[1];
and_1 = not_1 & x[2]; and_2 = not_2 & x[0]; xor_0 = x[4] ^ and_2;
xor_1 = x[3] ^ and_1; xor_2 = x[5] ^ and_0; xor_3 = x[0] ^ xor_1;
xor_4 = x[1] ^ xor_0; xor_5 = x[2] ^ xor_2; not_3 = ~xor_5; not_4 = ~xor_3;
not_5 = ~xor_4; and_3 = not_3 & xor_3; and_4 = not_4 & xor_4;
and_5 = not_5 & xor_5; xor_6 = and_3 ^ xor_0; xor_7 = and_4 ^ xor_2;
xor_8 = and_5 ^ xor_1; xor_9 = x[0] ^ and_5; xor_10 = xor_6 ^ xor_9;
xor_11 = x[1] ^ and_3; xor_12 = x[2] ^ and_4; not_6 = ~xor_6; not_7 = ~xor_7;
not_8 = ~xor_8; and_6 = not_6 & xor_7; and_7 = not_7 & xor_8;
and_8 = not_8 & xor_6; xor_13 = xor_10 ^ and_6; xor_14 = and_8 ^ xor_13;
xor_15 = xor_11 ^ and_7; xor_16 = xor_12 ^ and_8; y[0] = xor_14; y[1] = xor_15;
y[2] = xor_16; y[3] = xor_8; y[4] = xor_6; y[5] = xor_7;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 63]$):

```
0x00, 0x03, 0x06, 0x01, 0x05, 0x04, 0x02, 0x07, 0x18, 0x2f, 0x3a, 0x19, 0x2e, 0x3b, 0x0d, 0x0c
0x30, 0x1d, 0x1f, 0x3e, 0x3c, 0x13, 0x32, 0x11, 0x0b, 0x08, 0x35, 0x2a, 0x36, 0x17, 0x29, 0x14
0x28, 0x39, 0x33, 0x26, 0x37, 0x2c, 0x3d, 0x22, 0x20, 0x31, 0x0f, 0x0a, 0x25, 0x1e, 0x1b, 0x34
0x10, 0x27, 0x16, 0x2d, 0x1c, 0x21, 0x2b, 0x1a, 0x15, 0x38, 0x12, 0x23, 0x09, 0x0e, 0x3f, 0x24
```

Inverse netlist (9 AND, 17 XOR, 8 NOT, $D = 3$):

```
xor_0 = x[2] ^ x[4]; not_0 = ~x[4]; not_1 = ~x[5]; and_0 = x[3] & x[4];
and_1 = not_0 & x[5]; and_2 = not_1 & x[3]; xor_1 = x[0] ^ and_0;
xor_2 = x[1] ^ and_2; xor_3 = and_0 ^ xor_0; xor_4 = and_1 ^ xor_1;
xor_5 = x[3] ^ xor_4; xor_6 = x[4] ^ xor_2; xor_7 = x[5] ^ xor_3;
not_2 = ~xor_7; not_3 = ~xor_5; not_4 = ~xor_6; and_3 = not_2 & xor_5;
and_4 = not_3 & xor_6; and_5 = not_4 & xor_7; xor_8 = and_3 ^ xor_2;
xor_9 = and_4 ^ xor_3; xor_10 = and_5 ^ xor_4; xor_11 = x[3] ^ and_5;
xor_12 = x[4] ^ and_3; xor_13 = x[5] ^ and_4; not_5 = ~xor_10; not_6 = ~xor_8;
not_7 = ~xor_9; and_6 = xor_8 & not_5; and_7 = xor_9 & not_6;
and_8 = xor_10 & not_7; xor_14 = xor_11 ^ and_7; xor_15 = xor_12 ^ and_8;
xor_16 = xor_13 ^ and_6; y[0] = xor_10; y[1] = xor_8; y[2] = xor_9;
y[3] = xor_14; y[4] = xor_15; y[5] = xor_16;
```

C.13 $\mathsf{YSH}_{6,4}$: 6-bit degree 5, $\delta = 8$, $L = 28$ (6-bit, Category B)

- **Profile**: $\delta = 8$, $L = 28$ (NL = 18), $\deg(S) = 5$, $\mathcal{D}(S) = [2^{31}, 5^{32}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 11 AND, 22 XOR, 1 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 47.35 μm^2 / 59.3 GE, delay 427.0 ps, logic depth 9; pipelined $d = 2$ DOM-indep: 80 `DFF_X1` [44 gadget + 36 sync], 549.56 μm^2 / 688.7 GE, stage delay 358.1 ps).
- **Remark**: Same δ and degree as the SPEEDY S-box (20/51/3 in [BDM26b]) but a different class and a higher linearity (28; SPEEDY has $L = 24$): nine fewer AND gates and 29 fewer XOR gates at the same depth. Structure and origin: the nine level-1 products are exactly those of the Q_{2258} library seed circuit (9/30, $D = 1$; the un-sheared core is affine-equivalent to x^5), followed by two chained Stage-2 shear gates at levels 2 and 3 (Section 6.5).
- **Inverse ($\mathsf{YSH}_{6,4}^{-1}$)**: $\deg(S^{-1}) = 5$, $\mathcal{D}(S^{-1}) = [3^{15}, 4^{16}, 5^{32}]$; cost: 22 AND, 49 XOR, 1 NOT, AND depth $D = 3$ (Tier 2: 18-AND $D = 2$ tower core + 4-AND $D = 3$ scalar rank-1 shift correction; unmasked NanGate 45 nm: 102.14 μm^2 / 128.0 GE, delay 632.8 ps, logic depth 12; pipelined $d = 2$ DOM-indep: 132 `DFF_X1` [88 gadget + 44 sync], 987.92 μm^2 / 1238.0 GE, stage delay 480.4 ps).

Lookup table (LUT[0 . . . 63]):

```
0x02, 0x03, 0x01, 0x06, 0x07, 0x04, 0x05, 0x00, 0x1b, 0x0a, 0x38, 0x2f, 0x1e, 0x1d, 0x2c, 0x29
0x19, 0x28, 0x1a, 0x2d, 0x14, 0x27, 0x16, 0x23, 0x26, 0x17, 0x15, 0x22, 0x2b, 0x08, 0x09, 0x3c
0x3f, 0x3e, 0x24, 0x33, 0x2a, 0x39, 0x20, 0x35, 0x34, 0x25, 0x1f, 0x18, 0x21, 0x32, 0x0b, 0x0e
0x3d, 0x1c, 0x36, 0x11, 0x30, 0x13, 0x3a, 0x0f, 0x10, 0x31, 0x3b, 0x0c, 0x0d, 0x2e, 0x37, 0x12
```

Netlist (11 AND, 22 XOR, 1 NOT, $D = 3$):

```
xor_0 = x[1] ^ x[4]; xor_1 = x[2] ^ x[5]; xor_2 = x[0] ^ x[3];
and_0 = x[0] & x[1]; and_1 = x[0] & x[2]; and_2 = x[1] & x[2];
and_3 = x[3] & x[4]; and_4 = x[3] & x[5]; and_5 = x[4] & x[5];
and_6 = xor_2 & xor_0; and_7 = xor_2 & xor_1; and_8 = xor_0 & xor_1;
xor_3 = and_0 ^ and_3; xor_4 = and_1 ^ xor_0; xor_5 = x[4] ^ and_8;
xor_6 = xor_3 ^ xor_4; xor_7 = x[3] ^ xor_5; xor_8 = and_5 ^ xor_1;
xor_9 = and_0 ^ and_6; xor_10 = and_2 ^ xor_7; xor_11 = and_2 ^ xor_2;
xor_12 = and_4 ^ xor_6; xor_13 = xor_8 ^ xor_11; xor_14 = xor_0 ^ xor_13;
xor_15 = x[5] ^ xor_10; xor_16 = xor_1 ^ xor_3; xor_17 = x[4] ^ and_1;
xor_18 = and_7 ^ xor_9; xor_19 = xor_17 ^ xor_18; xor_20 = x[5] ^ xor_9;
not_0 = ~xor_12; and_9 = xor_15 & not_0; and_10 = and_9 & xor_14;
xor_21 = xor_19 ^ and_10; y[0] = xor_14; y[1] = not_0; y[2] = xor_16;
y[3] = xor_15; y[4] = xor_21; y[5] = xor_20;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 63]$):

```
0x07, 0x02, 0x00, 0x01, 0x05, 0x06, 0x03, 0x04, 0x1d, 0x1e, 0x09, 0x2e, 0x3b, 0x3c, 0x2f, 0x37
0x38, 0x33, 0x3f, 0x35, 0x14, 0x1a, 0x16, 0x19, 0x2b, 0x10, 0x12, 0x08, 0x31, 0x0d, 0x0c, 0x2a
0x26, 0x2c, 0x1b, 0x17, 0x22, 0x29, 0x18, 0x15, 0x11, 0x0f, 0x24, 0x1c, 0x0e, 0x13, 0x3d, 0x0b
0x34, 0x39, 0x2d, 0x23, 0x28, 0x27, 0x32, 0x3e, 0x0a, 0x25, 0x36, 0x3a, 0x1f, 0x30, 0x21, 0x20
```

Inverse netlist (22 AND, 49 XOR, 1 NOT, $D = 3$):

```
not_0 = ~x[1]; xor_0 = x[3] ^ x[5]; xor_1 = not_0 ^ x[0]; xor_2 = x[3] ^ x[4];
xor_3 = x[2] ^ xor_1; xor_4 = x[4] ^ xor_0; xor_5 = x[0] ^ x[2];
xor_6 = not_0 ^ x[2]; xor_7 = x[4] ^ x[5]; xor_8 = xor_1 ^ xor_2;
xor_9 = x[2] ^ x[5]; xor_10 = x[0] ^ x[3]; xor_11 = xor_3 ^ xor_4;
xor_12 = not_0 ^ x[4]; xor_13 = xor_0 ^ xor_5; and_0 = xor_1 & xor_2;
and_1 = xor_0 & xor_5; and_2 = x[2] & x[5]; and_3 = xor_6 & xor_7;
and_4 = xor_3 & xor_4; and_5 = x[0] & x[3]; xor_14 = and_0 ^ and_5;
xor_15 = and_1 ^ and_2; xor_16 = xor_14 ^ xor_15; xor_17 = and_3 ^ xor_14;
xor_18 = and_0 ^ and_1; xor_19 = and_4 ^ xor_18; xor_20 = xor_16 ^ xor_8;
xor_21 = xor_17 ^ xor_9; xor_22 = xor_10 ^ xor_19; xor_23 = xor_21 ^ xor_22;
and_6 = xor_20 & xor_8; and_7 = xor_23 & xor_9; and_8 = xor_10 & xor_21;
xor_24 = xor_20 ^ xor_23; and_9 = xor_11 & xor_24; xor_25 = xor_20 ^ xor_21;
and_10 = xor_12 & xor_25; and_11 = xor_13 & xor_22; and_12 = xor_2 & xor_20;
and_13 = x[5] & xor_23; and_14 = x[3] & xor_21; and_15 = xor_24 & xor_4;
and_16 = x[4] & xor_25; and_17 = xor_0 & xor_22; xor_26 = and_6 ^ and_8;
xor_27 = and_12 ^ and_14; xor_28 = and_10 ^ and_9; xor_29 = xor_26 ^ xor_28;
xor_30 = and_11 ^ and_7; xor_31 = xor_26 ^ xor_30; xor_32 = and_7 ^ and_8;
xor_33 = and_9 ^ xor_32; xor_34 = and_15 ^ and_16; xor_35 = xor_27 ^ xor_34;
xor_36 = and_13 ^ and_17; xor_37 = xor_27 ^ xor_36; xor_38 = and_13 ^ and_14;
xor_39 = and_15 ^ xor_38; and_18 = x[0] & x[1]; and_19 = and_18 & x[3];
and_20 = and_19 & x[2]; and_21 = and_2 & and_19; xor_40 = and_19 ^ and_21;
xor_41 = and_20 ^ and_21; xor_42 = xor_40 ^ and_20; xor_43 = and_20 ^ xor_29;
xor_44 = xor_42 ^ xor_31; xor_45 = xor_40 ^ xor_33; xor_46 = and_20 ^ xor_35;
xor_47 = xor_41 ^ xor_37; xor_48 = xor_42 ^ xor_39; y[0] = xor_43;
y[1] = xor_44; y[2] = xor_45; y[3] = xor_46; y[4] = xor_47; y[5] = xor_48;
```

C.14 $\mathsf{YSH}_{6,5}$: 6-bit degree 5, $\delta = 6$, $L = 24$ (6-bit, Category B)

- **Profile**: $\delta = 6$, $L = 24$ (NL = 20), $\deg(S) = 5$, $\mathcal{D}(S) = [2^{31}, 5^{32}]$, FP = 1, SAC deviation 0.125.
- **Cost**: 12 AND, 31 XOR, 0 NOT, AND depth $D = 4$ (unmasked NanGate 45 nm: 62.24 μm^2 / 78.0 GE, delay 453.9 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 98 `DFF_X1` [48 gadget + 50 sync], 669.79 μm^2 / 839.3 GE, stage delay 368.8 ps).
- **Remark**: Lower δ than the SPEEDY S-box (6 vs. 8) at the same $L = 24$, matching native SPEEDY's 12-gate NAND/NOR count ($D = 3$) [LMMR21] in the (AND, XOR) basis and using eight fewer AND and 20 fewer XOR gates than its 20/51/3 ($D = 3$) circuit in [BDM26b], at depth 4 instead of 3. Structure and origin: the nine level-1 products are exactly those of the Q_{2258} library seed circuit (9/30, $D = 1$; the un-sheared core is affine-equivalent to x^5), followed by three chained Stage-2 shear gates at levels 2, 3, and 4 (Section 6.5).

- **Inverse ($\mathsf{YSH}_{6,5}^{-1}$)**: $\deg(S^{-1}) = 5$, $\mathcal{D}(S^{-1}) = [3^{15}, 4^{16}, 5^{32}]$; cost: 21 AND, 57 XOR, 0 NOT, AND depth $D = 3$ (Tier 2: 18-AND $D = 2$ tower core + 3-AND $D = 3$ scalar rank-1 shift correction; unmasked NanGate 45 nm: 113.32 μm^2 / 142.0 GE, delay 869.9 ps, logic depth 15; pipelined $d = 2$ DOM-indep: 134 `DFF_X1` [84 gadget + 50 sync], 1011.33 μm^2 / 1267.3 GE, stage delay 381.2 ps).

Lookup table (LUT$[0 \dots 63]$):

```
0x00, 0x0d, 0x3e, 0x04, 0x09, 0x37, 0x3a, 0x33, 0x2c, 0x21, 0x2e, 0x14, 0x39, 0x07, 0x36, 0x1f
0x03, 0x32, 0x3d, 0x1b, 0x2b, 0x29, 0x18, 0x2d, 0x24, 0x15, 0x26, 0x20, 0x10, 0x12, 0x3f, 0x2a
0x08, 0x19, 0x17, 0x31, 0x01, 0x23, 0x13, 0x06, 0x0b, 0x1a, 0x28, 0x0e, 0x1e, 0x3c, 0x30, 0x25
0x27, 0x0a, 0x38, 0x22, 0x0f, 0x11, 0x1d, 0x34, 0x2f, 0x02, 0x0c, 0x16, 0x3b, 0x05, 0x35, 0x1c
```

Netlist (12 AND, 31 XOR, 0 NOT, $D = 4$):

```
xor_0 = x[2] ^ x[5]; xor_1 = x[0] ^ x[3]; xor_2 = x[1] ^ x[4];
and_0 = x[0] & x[1]; and_1 = x[0] & x[2]; and_2 = x[1] & x[2];
and_3 = x[3] & x[4]; and_4 = x[3] & x[5]; and_5 = x[4] & x[5];
and_6 = xor_1 & xor_2; and_7 = xor_1 & xor_0; and_8 = xor_2 & xor_0;
xor_3 = x[1] ^ and_1; xor_4 = and_3 ^ and_4; xor_5 = and_6 ^ and_7;
xor_6 = x[3] ^ and_2; xor_7 = x[0] ^ and_5; xor_8 = and_0 ^ xor_3;
xor_9 = xor_5 ^ xor_6; xor_10 = x[4] ^ xor_4; xor_11 = xor_7 ^ xor_9;
xor_12 = x[1] ^ xor_4; xor_13 = and_0 ^ and_1; xor_14 = and_8 ^ xor_13;
xor_15 = xor_5 ^ xor_12; xor_16 = xor_8 ^ xor_0; xor_17 = xor_8 ^ xor_10;
xor_18 = xor_11 ^ xor_16; xor_19 = x[5] ^ xor_9; xor_20 = xor_14 ^ xor_19;
xor_21 = x[2] ^ xor_7; xor_22 = xor_10 ^ xor_14; xor_23 = xor_21 ^ xor_22;
xor_24 = and_3 ^ xor_3; xor_25 = xor_11 ^ xor_24; xor_26 = and_6 ^ and_8;
xor_27 = xor_3 ^ xor_4; xor_28 = xor_6 ^ xor_26; xor_29 = xor_27 ^ xor_28;
and_9 = xor_20 & xor_17; and_10 = xor_15 & and_9; and_11 = and_10 & xor_18;
xor_30 = xor_29 ^ and_11; y[0] = xor_23; y[1] = xor_17; y[2] = xor_25;
y[3] = xor_18; y[4] = xor_15; y[5] = xor_30;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x24, 0x39, 0x10, 0x03, 0x3d, 0x27, 0x0d, 0x20, 0x04, 0x31, 0x28, 0x3a, 0x01, 0x2b, 0x34
0x1c, 0x35, 0x1d, 0x26, 0x0b, 0x19, 0x3b, 0x22, 0x16, 0x21, 0x29, 0x13, 0x3f, 0x36, 0x2c, 0x0f
0x1b, 0x09, 0x33, 0x25, 0x18, 0x2f, 0x1a, 0x30, 0x2a, 0x15, 0x1f, 0x14, 0x08, 0x17, 0x0a, 0x38
0x2e, 0x23, 0x11, 0x07, 0x37, 0x3e, 0x0e, 0x05, 0x32, 0x0c, 0x06, 0x3c, 0x2d, 0x12, 0x02, 0x1e
```

Inverse netlist (21 AND, 57 XOR, 0 NOT, $D = 3$):

```
xor_0 = x[1] ^ x[4]; xor_1 = x[0] ^ x[3]; xor_2 = x[2] ^ x[5];
xor_3 = x[2] ^ x[3]; xor_4 = x[0] ^ xor_0; xor_5 = x[3] ^ x[4];
xor_6 = x[2] ^ xor_0; xor_7 = x[2] ^ x[4]; xor_8 = xor_2 ^ xor_4;
xor_9 = x[4] ^ xor_8; xor_10 = x[1] ^ xor_3; xor_11 = x[4] ^ xor_2;
xor_12 = xor_3 ^ xor_8; xor_13 = x[4] ^ x[5]; xor_14 = x[1] ^ x[5];
xor_15 = x[1] ^ xor_1; xor_16 = x[4] ^ xor_1; xor_17 = x[3] ^ xor_0;
and_0 = xor_3 & xor_8; and_1 = x[1] & x[4]; and_2 = xor_2 & xor_7;
and_3 = xor_9 & xor_10; and_4 = xor_4 & xor_5; and_5 = xor_11 & xor_6;
and_6 = x[0] & x[1]; xor_18 = x[5] ^ and_0; xor_19 = and_2 ^ xor_1;
xor_20 = and_3 ^ and_5; xor_21 = and_1 ^ xor_0; xor_22 = and_4 ^ xor_18;
xor_23 = x[1] ^ and_1; xor_24 = and_3 ^ xor_19; xor_25 = xor_20 ^ xor_21;
xor_26 = xor_18 ^ xor_19; xor_27 = xor_23 ^ xor_24; xor_28 = xor_22 ^ xor_23;
xor_29 = and_4 ^ xor_25; xor_30 = x[4] ^ xor_18; xor_31 = xor_22 ^ xor_24;
xor_32 = and_5 ^ xor_26; xor_33 = xor_20 ^ xor_30; xor_34 = xor_21 ^ xor_32;
xor_35 = and_0 ^ and_2; xor_36 = and_6 ^ xor_25; xor_37 = xor_35 ^ xor_36;
and_7 = xor_12 & xor_34; and_8 = xor_13 & xor_29; and_9 = xor_14 & xor_33;
and_10 = xor_15 & xor_31; and_11 = xor_16 & xor_27; and_12 = xor_0 & xor_28;
and_13 = xor_34 & xor_3; and_14 = xor_29 & xor_7; and_15 = xor_6 & xor_33;
and_16 = xor_31 & xor_5; and_17 = xor_17 & xor_27; and_18 = x[1] & xor_28;
and_19 = and_6 & xor_37; and_20 = and_19 & x[2]; xor_38 = and_7 ^ and_19;
xor_39 = and_11 ^ xor_38; xor_40 = and_10 ^ and_20; xor_41 = and_8 ^ xor_39;
xor_42 = and_12 ^ xor_40; xor_43 = and_13 ^ and_16; xor_44 = and_9 ^ xor_39;
xor_45 = xor_40 ^ xor_44; xor_46 = and_13 ^ and_14; xor_47 = and_17 ^ xor_41;
xor_48 = xor_46 ^ xor_47; xor_49 = and_18 ^ xor_38; xor_50 = xor_43 ^ xor_49;
xor_51 = xor_50 ^ xor_42; xor_52 = and_15 ^ and_17; xor_53 = xor_43 ^ xor_52;
xor_54 = xor_53 ^ xor_45; xor_55 = xor_41 ^ and_20; xor_56 = and_7 ^ xor_42;
y[0] = xor_48; y[1] = xor_51; y[2] = xor_54; y[3] = xor_55; y[4] = xor_56;
y[5] = xor_45;
```

C.15 $\mathsf{YSH}_{6,6}$: 6-bit quartic, $\delta = 8$, $D = 2$ (6-bit, Category B)

- **Profile**: $\delta = 8$, $L = 24$ (NL = 20), $\deg(S) = 4$, $\mathcal{D}(S) = [2^{31}, 4^{32}]$, FP = 2, SAC deviation 0.5.

- **Cost**: 10 AND, 22 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 45.75 μm^2 / 57.3 GE, delay 362.8 ps, logic depth 7; pipelined $d = 2$ DOM-indep: 66 `DFF_X1` [40 gadget + 26 sync], 475.08 μm^2 / 595.3 GE, stage delay 317.9 ps).
- **Remark**: Degree 4 at depth $D = 2$, the minimum for degree 4. Structure and origin: the nine level-1 products are exactly those of the Q_{2258} library seed circuit (9/30, $D = 1$; the un-sheared core is affine-equivalent to x^5), followed by one Stage-2 shear gate at level 2 (Section 6.5).
- **Inverse ($\mathsf{YSH}_{6,6}^{-1}$)**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [3^7, 4^{56}]$; cost: 27 AND, 50 XOR, 3 NOT, AND depth $D = 2$ (Tier 2: 18-AND $D = 2$ tower core + 9-AND parallel $D = 2$ vector directional-derivative correction; unmasked NanGate 45 nm: 110.12 μm^2 / 138.0 GE, delay 561.3 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 140 `DFF_X1` [108 gadget + 32 sync], 1081.56 μm^2 / 1355.3 GE, stage delay 479.7 ps).

Lookup table (LUT$[0 \dots 63]$):

```
0x00, 0x01, 0x07, 0x02, 0x03, 0x04, 0x05, 0x06, 0x09, 0x08, 0x2e, 0x2b, 0x3b, 0x3c, 0x14, 0x17
0x3e, 0x16, 0x39, 0x15, 0x35, 0x1b, 0x33, 0x19, 0x1a, 0x32, 0x34, 0x18, 0x29, 0x0e, 0x0f, 0x2c
0x13, 0x2a, 0x1c, 0x21, 0x10, 0x2f, 0x1e, 0x25, 0x24, 0x1d, 0x0b, 0x3f, 0x1f, 0x20, 0x38, 0x0a
0x2d, 0x3d, 0x22, 0x36, 0x26, 0x30, 0x28, 0x3a, 0x37, 0x27, 0x11, 0x0c, 0x0d, 0x12, 0x23, 0x31
```

Netlist (10 AND, 22 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[2] ^ x[5]; xor_1 = x[0] ^ x[3]; xor_2 = x[1] ^ x[4];
xor_3 = xor_0 ^ xor_1; xor_4 = xor_2 ^ xor_3; and_0 = x[0] & x[1];
and_1 = x[0] & x[2]; and_2 = x[1] & x[2]; and_3 = x[3] & x[4];
and_4 = x[3] & x[5]; and_5 = x[4] & x[5]; and_6 = xor_1 & xor_2;
and_7 = xor_1 & xor_0; and_8 = xor_2 & xor_0; xor_5 = x[4] ^ and_1;
xor_6 = and_7 ^ xor_5; xor_7 = x[1] ^ and_4; xor_8 = x[5] ^ xor_6;
xor_9 = and_0 ^ and_6; xor_10 = xor_5 ^ xor_7; xor_11 = xor_6 ^ xor_9;
xor_12 = xor_4 ^ and_5; xor_13 = xor_0 ^ xor_10; xor_14 = and_0 ^ and_3;
xor_15 = xor_10 ^ xor_14; xor_16 = x[3] ^ and_1; xor_17 = and_7 ^ and_8;
xor_18 = xor_16 ^ xor_17; and_9 = xor_8 & xor_11; xor_19 = and_2 ^ and_9;
xor_20 = xor_12 ^ xor_19; xor_21 = xor_18 ^ xor_19; y[0] = xor_20;
y[1] = xor_13; y[2] = xor_15; y[3] = xor_21; y[4] = xor_8; y[5] = xor_11;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x01, 0x03, 0x04, 0x05, 0x06, 0x07, 0x02, 0x09, 0x08, 0x2f, 0x2a, 0x3b, 0x3c, 0x1d, 0x1e
0x24, 0x3a, 0x3d, 0x20, 0x0e, 0x13, 0x11, 0x0f, 0x1b, 0x17, 0x18, 0x15, 0x22, 0x29, 0x26, 0x2c
0x2d, 0x23, 0x32, 0x3e, 0x28, 0x27, 0x34, 0x39, 0x36, 0x1c, 0x21, 0x0b, 0x1f, 0x30, 0x0a, 0x25
0x35, 0x3f, 0x19, 0x16, 0x1a, 0x14, 0x33, 0x38, 0x2e, 0x12, 0x37, 0x0c, 0x0d, 0x31, 0x10, 0x2b
```

Inverse netlist (27 AND, 50 XOR, 3 NOT, $D = 2$):

```
xor_0 = x[3] ^ x[4]; xor_1 = x[2] ^ x[5]; xor_2 = x[0] ^ xor_0;
xor_3 = x[0] ^ x[1]; xor_4 = x[1] ^ x[4]; xor_5 = x[0] ^ x[2];
xor_6 = x[5] ^ xor_0; xor_7 = x[1] ^ x[2]; xor_8 = x[4] ^ x[5];
xor_9 = xor_1 ^ xor_4; xor_10 = xor_1 ^ xor_2; xor_11 = x[3] ^ xor_1;
xor_12 = xor_11 ^ xor_3; xor_13 = x[3] ^ x[5]; and_0 = x[0] & xor_0;
and_1 = x[1] & x[4]; and_2 = x[2] & x[5]; and_3 = x[3] & xor_3;
and_4 = xor_5 & xor_6; and_5 = xor_7 & xor_8; xor_14 = and_0 ^ and_5;
xor_15 = and_1 ^ and_2; xor_16 = xor_14 ^ xor_15; xor_17 = and_3 ^ xor_14;
xor_18 = and_0 ^ and_1; xor_19 = and_4 ^ xor_18; xor_20 = xor_16 ^ xor_2;
xor_21 = xor_1 ^ xor_17; xor_22 = xor_19 ^ xor_9; xor_23 = xor_21 ^ xor_22;
and_6 = xor_2 & xor_20; and_7 = xor_1 & xor_23; and_8 = xor_21 & xor_9;
xor_24 = xor_20 ^ xor_23; and_9 = xor_10 & xor_24; xor_25 = xor_20 ^ xor_21;
and_10 = xor_12 & xor_25; and_11 = xor_22 & xor_4; and_12 = xor_0 & xor_20;
and_13 = x[5] & xor_23; and_14 = xor_21 & xor_8; and_15 = xor_24 & xor_6;
and_16 = xor_13 & xor_25; and_17 = x[4] & xor_22; xor_26 = and_11 ^ and_6;
xor_27 = and_12 ^ and_17; xor_28 = and_7 ^ and_8; xor_29 = xor_26 ^ xor_28;
xor_30 = and_9 ^ xor_26; xor_31 = and_6 ^ and_7; xor_32 = and_10 ^ xor_31;
xor_33 = and_13 ^ and_14; xor_34 = xor_27 ^ xor_33; xor_35 = and_15 ^ xor_27;
xor_36 = and_12 ^ and_13; xor_37 = and_16 ^ xor_36; and_18 = x[4] & x[5];
xor_38 = x[0] ^ x[3]; and_19 = x[1] & x[2]; and_20 = xor_38 & x[1];
and_21 = xor_38 & x[2]; not_0 = ~and_19; and_22 = not_0 & and_18;
xor_39 = xor_38 ^ x[1]; and_23 = xor_39 & and_18; xor_40 = and_23 ^ and_18;
xor_41 = xor_39 ^ and_20; not_1 = ~xor_41; and_24 = not_1 & and_18;
xor_42 = xor_38 ^ x[2]; xor_43 = xor_42 ^ and_21; not_2 = ~xor_43;
and_25 = not_2 & and_18; and_26 = and_18 & x[2]; xor_44 = and_22 ^ xor_29;
xor_45 = and_24 ^ xor_30; xor_46 = and_25 ^ xor_32; xor_47 = and_26 ^ xor_34;
xor_48 = and_23 ^ xor_35; xor_49 = xor_40 ^ xor_37; y[0] = xor_44;
y[1] = xor_45; y[2] = xor_46; y[3] = xor_47; y[4] = xor_48; y[5] = xor_49;
```

C.16 6-bit Q_{2258} class (x^5) (6-bit, Category A)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 24), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{63}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 8 AND, 34 XOR, 0 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 62.78 μm^2 / 78.7 GE, delay 527.6 ps, logic depth 9; pipelined $d = 2$ DOM-indep: 44 `DFF_X1` [32 gadget + 12 sync], 392.62 μm^2 / 492.0 GE, stage delay 354.6 ps).
- **Remark**: Certified linearly equivalent to x^5 ($\sim_{\text{aff}} x^5$, with $S_{\text{ref}}(u) = u^5$ mod $(x^6 + x + 1)$, $A =$ [`0x01`, `0x0e`, `0x17`, `0x3a`, `0x03`, `0x12`], $a =$ `0x00`, $B =$ [`0x01`, `0x11`, `0x25`, `0x36`, `0x31`, `0x1e`], $b =$ `0x00`); 8/34/1 in [BDM26b] and 8/25/1 in [BDM26a]: identical 8 AND and 34 XOR count as [BDM26b] (nine more XOR gates than [BDM26a]) at depth $D = 1$.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{63}]$; cost: 18 AND, 37 XOR, 0 NOT, AND depth $D = 2$ (Tier 3: Tower-field norm inversion $y^{38} = y^2(y^9)^4$ over $\mathbb{F}_{2^6}/\mathbb{F}_{2^3}$; unmasked NanGate 45 nm: 78.20 μm^2 / 98.0 GE, delay 473.2 ps, logic depth 9; pipelined $d = 2$ DOM-indep: 96 `DFF_X1` [72 gadget + 24 sync], 743.74 μm^2 / 932.0 GE, stage delay 480.1 ps).

Lookup table (LUT$[0 \dots 63]$):

```
0x00, 0x01, 0x07, 0x02, 0x03, 0x04, 0x05, 0x06, 0x09, 0x08, 0x2e, 0x2b, 0x3a, 0x3d, 0x1c, 0x1f
0x3f, 0x1e, 0x38, 0x1d, 0x34, 0x13, 0x32, 0x11, 0x12, 0x33, 0x35, 0x10, 0x29, 0x0e, 0x0f, 0x2c
0x1b, 0x2a, 0x14, 0x21, 0x18, 0x2f, 0x16, 0x25, 0x24, 0x15, 0x0b, 0x3e, 0x17, 0x20, 0x39, 0x0a
0x2d, 0x3c, 0x22, 0x37, 0x26, 0x31, 0x28, 0x3b, 0x36, 0x27, 0x19, 0x0c, 0x0d, 0x1a, 0x23, 0x30
```

Netlist (8 AND, 34 XOR, 0 NOT, $D = 1$):

```
xor_0 = x[3] ^ x[4]; xor_1 = x[0] ^ xor_0; xor_2 = x[1] ^ x[2];
xor_3 = x[2] ^ x[3]; xor_4 = x[4] ^ x[5]; xor_5 = x[0] ^ x[5];
xor_6 = x[1] ^ xor_1; xor_7 = x[5] ^ xor_1; xor_8 = x[5] ^ xor_3;
xor_9 = xor_4 ^ xor_6; xor_10 = xor_2 ^ xor_4; xor_11 = x[3] ^ xor_6;
xor_12 = xor_0 ^ xor_2; xor_13 = x[2] ^ xor_7; xor_14 = x[0] ^ x[2];
xor_15 = x[4] ^ xor_2; and_0 = x[3] & xor_2; and_1 = xor_8 & xor_1;
and_2 = xor_3 & xor_6; and_3 = xor_9 & xor_5; and_4 = xor_10 & x[0];
and_5 = xor_11 & xor_12; and_6 = xor_4 & xor_7; and_7 = xor_13 & xor_0;
xor_16 = and_5 ^ and_7; xor_17 = and_4 ^ and_6; xor_18 = x[1] ^ xor_17;
xor_19 = and_2 ^ xor_16; xor_20 = xor_14 ^ and_0; xor_21 = xor_16 ^ xor_17;
xor_22 = xor_20 ^ xor_21; xor_23 = xor_15 ^ and_1; xor_24 = and_6 ^ and_7;
xor_25 = xor_23 ^ xor_24; xor_26 = x[5] ^ xor_18; xor_27 = xor_6 ^ and_3;
xor_28 = xor_19 ^ xor_27; xor_29 = x[3] ^ and_1; xor_30 = xor_18 ^ xor_19;
xor_31 = xor_29 ^ xor_30; xor_32 = x[5] ^ and_0; xor_33 = and_6 ^ xor_32;
y[0] = xor_22; y[1] = xor_25; y[2] = xor_26; y[3] = xor_28; y[4] = xor_31;
y[5] = xor_33;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x01, 0x03, 0x04, 0x05, 0x06, 0x07, 0x02, 0x09, 0x08, 0x2f, 0x2a, 0x3b, 0x3c, 0x1d, 0x1e
0x1b, 0x17, 0x18, 0x15, 0x22, 0x29, 0x26, 0x2c, 0x24, 0x3a, 0x3d, 0x20, 0x0e, 0x13, 0x11, 0x0f
0x2d, 0x23, 0x32, 0x3e, 0x28, 0x27, 0x34, 0x39, 0x36, 0x1c, 0x21, 0x0b, 0x1f, 0x30, 0x0a, 0x25
0x3f, 0x35, 0x16, 0x19, 0x14, 0x1a, 0x38, 0x33, 0x12, 0x2e, 0x0c, 0x37, 0x31, 0x0d, 0x2b, 0x10
```

Inverse netlist (18 AND, 37 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[2] ^ x[5]; xor_1 = x[0] ^ x[3]; xor_2 = x[1] ^ x[4];
xor_3 = xor_0 ^ xor_1; xor_4 = x[3] ^ x[4]; xor_5 = x[0] ^ x[1];
xor_6 = x[0] ^ x[2]; xor_7 = x[3] ^ x[5]; xor_8 = x[1] ^ x[2];
xor_9 = x[4] ^ x[5]; xor_10 = xor_0 ^ xor_2; xor_11 = xor_2 ^ xor_3;
xor_12 = xor_4 ^ x[5]; and_0 = x[0] & x[3]; and_1 = x[1] & x[4];
and_2 = x[2] & x[5]; and_3 = xor_4 & xor_5; and_4 = xor_6 & xor_7;
and_5 = xor_8 & xor_9; xor_13 = and_0 ^ and_5; xor_14 = and_1 ^ and_2;
xor_15 = xor_13 ^ xor_14; xor_16 = and_3 ^ xor_13; xor_17 = and_0 ^ and_1;
xor_18 = and_4 ^ xor_17; xor_19 = xor_1 ^ xor_15; xor_20 = xor_0 ^ xor_16;
xor_21 = xor_10 ^ xor_18; xor_22 = xor_20 ^ xor_21; and_6 = xor_1 & xor_19;
and_7 = xor_0 & xor_22; and_8 = xor_10 & xor_20; xor_23 = xor_19 ^ xor_22;
and_9 = xor_3 & xor_23; xor_24 = xor_19 ^ xor_20; and_10 = xor_11 & xor_24;
and_11 = xor_2 & xor_21; and_12 = xor_19 & x[3]; and_13 = xor_22 & x[5];
and_14 = xor_9 & xor_20; and_15 = xor_7 & xor_23; and_16 = xor_12 & xor_24;
and_17 = xor_21 & x[4]; xor_25 = and_11 ^ and_6; xor_26 = and_12 ^ and_17;
```

```
xor_27 = and_7 ^ and_8; xor_28 = xor_25 ^ xor_27; xor_29 = and_9 ^ xor_25;
xor_30 = and_6 ^ and_7; xor_31 = and_10 ^ xor_30; xor_32 = and_13 ^ and_14;
xor_33 = xor_26 ^ xor_32; xor_34 = and_15 ^ xor_26; xor_35 = and_12 ^ and_13;
xor_36 = and_16 ^ xor_35; y[0] = xor_28; y[1] = xor_29; y[2] = xor_31;
y[3] = xor_33; y[4] = xor_34; y[5] = xor_36;
```

C.17 6-bit 3-round χ_3-Feistel cubic involution ($\sim_{\text{aff}}$ Feistel$_{3R}(\chi_3)$, 6-bit, Category A)

- **Profile**: $\delta = 4$, $L = 16$ (NL = 24), $\deg(S) = 3$, $\mathcal{D}(S) = [3^{63}]$, FP = 8, SAC deviation 0.5.
- **Cost**: 9 AND, 15 XOR, 9 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 38.30 μm^2 / 48.0 GE, delay 336.8 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 72 `DFF_X1` [36 gadget + 36 sync], 474.01 μm^2 / 594.0 GE, stage delay 254.4 ps).
- **Remark**: Certified linearly equivalent to the 3-round χ_3-Feistel involution Feistel$_{3R}(\chi_3)$ [CDL16, BDMD+20] ($\sim_{\text{aff}}$ `Feistel_3R_Chi3`, with $A =$ [`0x01`, `0x03`, `0x07`, `0x18`, `0x30`, `0x20`], $a =$ `0x00`, $B =$ [`0x01`, `0x03`, `0x06`, `0x38`, `0x30`, `0x20`], $b =$ `0x00`). All three forward invariants equal those of x^{13} (the class of Q_{2258}^{-1}), but it differs in inverse spectrum ($\mathcal{D}(S^{-1}) = [3^{63}]$ vs. $[2^{63}]$), boomerang uniformity ($\beta = 64$ vs. 4), and third-derivative rank multiset ($[2^{273}, 3^{840}, 4^{840}]$ vs. $[3^{189}, 4^{1764}]$ for x^{13}), certifying that it is not EA-equivalent to x^{13}, and the same circuit also computes its inverse. Compared with the 11/42/2 circuit for Q_{2258}^{-1} in [BDM26b], it uses two fewer AND gates and 27 fewer XOR gates, at one more AND level and with 9 NOT gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{63}]$; cost: 9 AND, 15 XOR, 9 NOT, AND depth $D = 3$ (Tier 1: Exact involution ($S^{-1} = S$); unmasked NanGate 45 nm: 38.30 μm^2 / 48.0 GE, delay 336.8 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 72 `DFF_X1` [36 gadget + 36 sync], 474.01 μm^2 / 594.0 GE, stage delay 254.4 ps).

Lookup table (LUT$[0 \dots 63]$):

```
0x00, 0x18, 0x30, 0x08, 0x28, 0x20, 0x10, 0x38, 0x03, 0x3d, 0x17, 0x0b, 0x35, 0x1f, 0x29, 0x21
0x06, 0x2b, 0x3b, 0x37, 0x27, 0x1a, 0x16, 0x0a, 0x01, 0x19, 0x15, 0x2e, 0x3a, 0x36, 0x22, 0x0d
0x05, 0x0f, 0x1e, 0x34, 0x3e, 0x25, 0x2f, 0x14, 0x04, 0x0e, 0x39, 0x11, 0x2c, 0x33, 0x1b, 0x26
0x02, 0x3c, 0x32, 0x2d, 0x23, 0x0c, 0x1d, 0x13, 0x07, 0x2a, 0x1c, 0x12, 0x31, 0x09, 0x24, 0x3f
```

Netlist (9 AND, 15 XOR, 9 NOT, $D = 3$):

```
not_0 = ~x[3]; not_1 = ~x[4]; not_2 = ~x[5]; and_0 = not_0 & x[4];
and_1 = not_1 & x[5]; and_2 = not_2 & x[3]; xor_0 = x[0] ^ and_1;
xor_1 = x[1] ^ and_2; xor_2 = x[2] ^ and_0; xor_3 = x[3] ^ xor_0;
xor_4 = x[4] ^ xor_1; xor_5 = x[5] ^ xor_2; not_3 = ~xor_3; not_4 = ~xor_4;
not_5 = ~xor_5; and_3 = not_3 & xor_4; and_4 = not_4 & xor_5;
and_5 = not_5 & xor_3; xor_6 = and_3 ^ xor_2; xor_7 = and_4 ^ xor_0;
xor_8 = and_5 ^ xor_1; xor_9 = x[3] ^ and_4; xor_10 = x[4] ^ and_5;
xor_11 = x[5] ^ and_3; not_6 = ~xor_7; not_7 = ~xor_6; not_8 = ~xor_8;
and_6 = not_6 & xor_8; and_7 = not_7 & xor_7; and_8 = not_8 & xor_6;
xor_12 = xor_9 ^ and_8; xor_13 = xor_10 ^ and_7; xor_14 = xor_11 ^ and_6;
y[0] = xor_12; y[1] = xor_13; y[2] = xor_14; y[3] = xor_7; y[4] = xor_8;
y[5] = xor_6;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x18, 0x30, 0x08, 0x28, 0x20, 0x10, 0x38, 0x03, 0x3d, 0x17, 0x0b, 0x35, 0x1f, 0x29, 0x21
0x06, 0x2b, 0x3b, 0x37, 0x27, 0x1a, 0x16, 0x0a, 0x01, 0x19, 0x15, 0x2e, 0x3a, 0x36, 0x22, 0x0d
0x05, 0x0f, 0x1e, 0x34, 0x3e, 0x25, 0x2f, 0x14, 0x04, 0x0e, 0x39, 0x11, 0x2c, 0x33, 0x1b, 0x26
0x02, 0x3c, 0x32, 0x2d, 0x23, 0x0c, 0x1d, 0x13, 0x07, 0x2a, 0x1c, 0x12, 0x31, 0x09, 0x24, 0x3f
```

Inverse netlist (9 AND, 15 XOR, 9 NOT, $D = 3$):

```
not_0 = ~x[3]; not_1 = ~x[4]; not_2 = ~x[5]; and_0 = not_0 & x[4];
and_1 = not_1 & x[5]; and_2 = not_2 & x[3]; xor_0 = x[0] ^ and_1;
xor_1 = x[1] ^ and_2; xor_2 = x[2] ^ and_0; xor_3 = x[3] ^ xor_0;
xor_4 = x[4] ^ xor_1; xor_5 = x[5] ^ xor_2; not_3 = ~xor_3; not_4 = ~xor_4;
not_5 = ~xor_5; and_3 = not_3 & xor_4; and_4 = not_4 & xor_5;
```

```
and_5 = not_5 & xor_3; xor_6 = and_3 ^ xor_2; xor_7 = and_4 ^ xor_0;
xor_8 = and_5 ^ xor_1; xor_9 = x[3] ^ and_4; xor_10 = x[4] ^ and_5;
xor_11 = x[5] ^ and_3; not_6 = ~xor_7; not_7 = ~xor_6; not_8 = ~xor_8;
and_6 = not_6 & xor_8; and_7 = not_7 & xor_7; and_8 = not_8 & xor_6;
xor_12 = xor_9 ^ and_8; xor_13 = xor_10 ^ and_7; xor_14 = xor_11 ^ and_6;
y[0] = xor_12; y[1] = xor_13; y[2] = xor_14; y[3] = xor_7; y[4] = xor_8;
y[5] = xor_6;
```

C.18 6-bit inversion class (6-bit, Category A)

- **Profile**: $\delta = 4$, $L = 16$ (NL $= 24$), $\deg(S) = 5$, $\mathcal{D}(S) = [5^{63}]$, FP $= 1$, SAC deviation 0.125.
- **Cost**: 21 AND, 48 XOR, 0 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 98.95 μm^2 / 124.0 GE, delay 797.5 ps, logic depth 15; pipelined $d = 2$ DOM-indep: 152 `DFF_X1` [84 gadget + 68 sync], 1064.00 μm^2 / 1333.3 GE, stage delay 367.4 ps).
- **Remark**: Certified linearly equivalent to the inversion $x^{31} \sim_{\text{aff}} x^{62}$ (with $S_{\text{ref}}(u) = u^{31} \bmod (x^6 + x + 1)$, $A =$ [`0x01`, `0x0e`, `0x17`, `0x2c`, `0x02`, `0x1c`], $a =$ `0x00`, $B =$ [`0x08`, `0x3c`, `0x16`, `0x0a`, `0x12`, `0x05`], $b =$ `0x00`); the tower-field circuit 21/63/3 of [BDM26b] has the same AND count and depth and 15 more XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 5$, $\mathcal{D}(S^{-1}) = [5^{63}]$; cost: 21 AND, 50 XOR, 0 NOT, AND depth $D = 3$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 102.14 μm^2 / 128.0 GE, delay 771.4 ps, logic depth 14; pipelined $d = 2$ DOM-indep: 152 `DFF_X1` [84 gadget + 68 sync], 1070.38 μm^2 / 1341.3 GE, stage delay 317.0 ps).

Lookup table (LUT$[0 \dots 63]$):

```
0x00, 0x08, 0x28, 0x30, 0x38, 0x10, 0x18, 0x20, 0x36, 0x06, 0x32, 0x22, 0x29, 0x21, 0x0c, 0x2c
0x1b, 0x15, 0x03, 0x3d, 0x19, 0x3a, 0x11, 0x2a, 0x12, 0x35, 0x1d, 0x02, 0x2e, 0x2f, 0x17, 0x1e
0x24, 0x31, 0x0f, 0x25, 0x04, 0x39, 0x37, 0x0d, 0x3f, 0x3c, 0x0a, 0x13, 0x1c, 0x07, 0x0b, 0x1a
0x09, 0x3e, 0x1f, 0x3b, 0x27, 0x23, 0x01, 0x0e, 0x2d, 0x14, 0x33, 0x16, 0x26, 0x2b, 0x34, 0x05
```

Netlist (21 AND, 48 XOR, 0 NOT, $D = 3$):

```
xor_0 = x[0] ^ x[1]; xor_1 = x[3] ^ x[4]; xor_2 = x[0] ^ x[2];
xor_3 = x[4] ^ x[5]; xor_4 = x[1] ^ x[2]; xor_5 = x[3] ^ x[5];
and_0 = x[3] & x[0]; and_1 = x[4] & x[1]; and_2 = x[5] & x[2];
and_3 = xor_1 & xor_0; and_4 = xor_5 & xor_2; and_5 = xor_3 & xor_4;
xor_6 = x[4] ^ and_0; xor_7 = x[3] ^ and_5; xor_8 = x[0] ^ and_2;
xor_9 = xor_6 ^ xor_7; xor_10 = x[1] ^ and_4; xor_11 = x[5] ^ xor_8;
xor_12 = x[2] ^ and_1; xor_13 = xor_6 ^ xor_10; xor_14 = xor_12 ^ xor_13;
xor_15 = and_1 ^ xor_9; xor_16 = x[2] ^ and_3; xor_17 = xor_9 ^ xor_16;
xor_18 = xor_11 ^ xor_15; xor_19 = and_3 ^ xor_10; and_6 = xor_18 & xor_14;
and_7 = xor_17 & xor_14; and_8 = xor_18 & xor_17; xor_20 = and_7 ^ xor_11;
xor_21 = and_3 ^ xor_20; xor_22 = and_6 ^ xor_7; xor_23 = and_1 ^ xor_22;
xor_24 = and_8 ^ xor_12; xor_25 = xor_21 ^ xor_24; xor_26 = and_6 ^ xor_17;
xor_27 = and_8 ^ xor_14; xor_28 = xor_20 ^ xor_15; xor_29 = and_6 ^ xor_28;
xor_30 = xor_23 ^ xor_19; xor_31 = xor_13 ^ xor_21; xor_32 = and_8 ^ xor_26;
and_9 = x[4] & xor_27; and_10 = x[1] & xor_27; and_11 = x[5] & xor_30;
and_12 = x[2] & xor_30; and_13 = x[3] & xor_31; and_14 = x[0] & xor_31;
and_15 = xor_3 & xor_32; and_16 = xor_4 & xor_32; and_17 = xor_1 & xor_25;
and_18 = xor_0 & xor_25; and_19 = xor_5 & xor_29; and_20 = xor_2 & xor_29;
xor_33 = and_13 ^ and_15; xor_34 = and_9 ^ and_19; xor_35 = and_11 ^ xor_33;
xor_36 = and_14 ^ and_16; xor_37 = and_9 ^ xor_35; xor_38 = and_13 ^ xor_34;
xor_39 = and_17 ^ xor_33; xor_40 = and_10 ^ and_12; xor_41 = xor_36 ^ xor_37;
xor_42 = xor_40 ^ xor_41; xor_43 = and_18 ^ xor_36; xor_44 = xor_39 ^ xor_43;
xor_45 = and_10 ^ and_14; xor_46 = and_20 ^ xor_38; xor_47 = xor_45 ^ xor_46;
y[0] = xor_37; y[1] = xor_39; y[2] = xor_38; y[3] = xor_42; y[4] = xor_44;
y[5] = xor_47;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 63]$):

```
0x00, 0x36, 0x1b, 0x12, 0x24, 0x3f, 0x09, 0x2d, 0x01, 0x30, 0x2a, 0x2e, 0x0e, 0x27, 0x37, 0x22
0x05, 0x16, 0x18, 0x2b, 0x39, 0x11, 0x3b, 0x1e, 0x06, 0x14, 0x2f, 0x10, 0x2c, 0x1a, 0x1f, 0x32
0x07, 0x0d, 0x0b, 0x35, 0x20, 0x23, 0x3c, 0x34, 0x02, 0x0c, 0x17, 0x3d, 0x0f, 0x38, 0x1c, 0x1d
0x03, 0x21, 0x0a, 0x3a, 0x3e, 0x19, 0x08, 0x26, 0x04, 0x25, 0x15, 0x33, 0x29, 0x13, 0x31, 0x28
```

Inverse netlist (21 AND, 50 XOR, 0 NOT, $D = 3$):

```
xor_0 = x[2] ^ x[3]; xor_1 = x[3] ^ x[4]; xor_2 = x[0] ^ x[1];
xor_3 = x[0] ^ x[2]; xor_4 = x[1] ^ x[2]; xor_5 = x[3] ^ x[5];
xor_6 = x[4] ^ x[5]; xor_7 = x[5] ^ xor_1; and_0 = xor_6 & x[0];
and_1 = xor_1 & x[1]; and_2 = xor_7 & x[2]; and_3 = xor_5 & xor_2;
and_4 = xor_3 & x[3]; and_5 = xor_4 & x[5]; xor_8 = x[4] ^ and_1;
xor_9 = x[5] ^ and_3; xor_10 = x[0] ^ and_2; xor_11 = x[1] ^ and_4;
xor_12 = and_0 ^ and_5; xor_13 = and_0 ^ xor_11; xor_14 = xor_8 ^ xor_0;
xor_15 = xor_8 ^ xor_12; xor_16 = xor_0 ^ xor_9; xor_17 = xor_12 ^ xor_16;
xor_18 = xor_13 ^ xor_14; xor_19 = and_5 ^ xor_8; xor_20 = xor_9 ^ xor_11;
xor_21 = xor_10 ^ xor_15; xor_22 = xor_19 ^ xor_20; and_6 = xor_21 & xor_18;
and_7 = xor_17 & xor_18; and_8 = xor_21 & xor_17; xor_23 = and_7 ^ xor_10;
xor_24 = xor_9 ^ xor_23; xor_25 = and_6 ^ and_8; xor_26 = and_6 ^ xor_23;
xor_27 = and_6 ^ xor_22; xor_28 = and_8 ^ xor_14; xor_29 = and_8 ^ xor_18;
xor_30 = xor_13 ^ xor_24; xor_31 = xor_15 ^ xor_26; xor_32 = xor_17 ^ xor_25;
xor_33 = xor_24 ^ xor_28; and_9 = xor_1 & xor_29; and_10 = xor_29 & x[1];
and_11 = xor_7 & xor_27; and_12 = xor_27 & x[2]; and_13 = xor_6 & xor_30;
and_14 = xor_30 & x[0]; and_15 = xor_32 & x[5]; and_16 = xor_4 & xor_32;
and_17 = xor_5 & xor_33; and_18 = xor_2 & xor_33; and_19 = xor_31 & x[3];
and_20 = xor_3 & xor_31; xor_34 = and_10 ^ and_18; xor_35 = and_9 ^ and_13;
xor_36 = and_12 ^ and_14; xor_37 = and_12 ^ xor_34; xor_38 = and_16 ^ and_20;
xor_39 = and_18 ^ and_20; xor_40 = xor_34 ^ xor_38; xor_41 = xor_36 ^ xor_39;
xor_42 = and_11 ^ and_15; xor_43 = xor_35 ^ xor_40; xor_44 = xor_42 ^ xor_43;
xor_45 = and_13 ^ and_15; xor_46 = and_17 ^ xor_37; xor_47 = xor_45 ^ xor_46;
xor_48 = and_19 ^ xor_35; xor_49 = xor_41 ^ xor_48; y[0] = xor_44;
y[1] = xor_47; y[2] = xor_49; y[3] = xor_40; y[4] = xor_37; y[5] = xor_41;
```

C.19 6-bit Dillon APN permutation (6-bit, Category A)

- **Profile**: $\delta = 2$, $L = 16$ (NL = 24), $\deg(S) = 4$, $\mathcal{D}(S) = [3^7, 4^{56}]$, FP = 1, SAC deviation 0.5.
- **Cost**: 28 AND, 68 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 138.32 μm^2 / 173.3 GE, delay 538.6 ps, logic depth 10; pipelined $d = 2$ DOM-indep: 154 `DFF_X1` [112 gadget + 42 sync], 1211.36 μm^2 / 1518.0 GE, stage delay 460.5 ps).
- **Remark**: Equal to the APN permutation [BDMW10] of our reference library; 17/75/2 in [BDM26b]: our circuit has 11 more AND gates and seven fewer XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [3^7, 4^{56}]$; cost: 28 AND, 78 XOR, 7 NOT, AND depth $D = 2$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 158.00 μm^2 / 198.0 GE, delay 819.8 ps, logic depth 15; pipelined $d = 2$ DOM-indep: 168 `DFF_X1` [112 gadget + 56 sync], 1310.32 μm^2 / 1642.0 GE, stage delay 423.5 ps).

Lookup table (LUT[0 . . . 63]):

```
0x00, 0x36, 0x30, 0x0d, 0x0f, 0x12, 0x35, 0x23, 0x19, 0x3f, 0x2d, 0x34, 0x03, 0x14, 0x29, 0x21
0x3b, 0x24, 0x02, 0x22, 0x0a, 0x08, 0x39, 0x25, 0x3c, 0x13, 0x2a, 0x0e, 0x32, 0x1a, 0x3a, 0x18
0x27, 0x1b, 0x15, 0x11, 0x10, 0x1d, 0x01, 0x3e, 0x2f, 0x28, 0x33, 0x38, 0x07, 0x2b, 0x2c, 0x26
0x1f, 0x0b, 0x04, 0x1c, 0x3d, 0x2e, 0x05, 0x31, 0x09, 0x06, 0x17, 0x20, 0x1e, 0x0c, 0x37, 0x16
```

Netlist (28 AND, 68 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[2]; xor_1 = x[2] ^ x[3]; xor_2 = x[5] ^ xor_0;
and_0 = x[0] & x[1]; and_1 = x[0] & x[2]; and_2 = x[1] & x[2];
and_3 = x[0] & x[3]; and_4 = x[1] & x[3]; and_5 = x[2] & x[3];
and_6 = x[0] & x[4]; and_7 = x[1] & x[4]; and_8 = x[2] & x[4];
and_9 = x[3] & x[4]; and_10 = x[0] & x[5]; and_11 = x[1] & x[5];
and_12 = x[2] & x[5]; and_13 = x[3] & x[5]; and_14 = x[4] & x[5];
xor_3 = and_3 ^ and_8; xor_4 = x[4] ^ and_5; xor_5 = and_6 ^ xor_3;
xor_6 = and_0 ^ and_1; xor_7 = x[0] ^ and_12; xor_8 = x[3] ^ and_8;
xor_9 = and_14 ^ xor_6; xor_10 = x[0] ^ and_0; xor_11 = and_7 ^ xor_1;
xor_12 = and_13 ^ xor_4; xor_13 = x[1] ^ xor_7; xor_14 = x[5] ^ and_2;
xor_15 = x[1] ^ and_2; xor_16 = x[3] ^ and_6; xor_17 = and_1 ^ and_6;
xor_18 = and_0 ^ and_8; xor_19 = x[1] ^ and_1; xor_20 = x[3] ^ and_10;
xor_21 = and_6 ^ and_11; xor_22 = x[3] ^ and_1; xor_23 = and_7 ^ and_11;
xor_24 = x[3] ^ and_11; xor_25 = and_7 ^ and_10; xor_26 = x[4] ^ and_13;
xor_27 = x[0] ^ and_11; xor_28 = x[3] ^ and_14; xor_29 = x[0] ^ and_14;
xor_30 = and_8 ^ and_11; xor_31 = xor_9 ^ xor_11; xor_32 = xor_12 ^ xor_31;
xor_33 = xor_2 ^ and_11; xor_34 = xor_9 ^ xor_33; xor_35 = xor_2 ^ and_5;
xor_36 = xor_3 ^ xor_35; xor_37 = and_12 ^ xor_6; xor_38 = xor_11 ^ xor_37;
```

```
xor_39 = xor_8 ^ xor_12; xor_40 = xor_13 ^ xor_39; and_15 = and_6 & xor_14;
and_16 = xor_15 & xor_16; and_17 = xor_1 & xor_17; and_18 = x[3] & xor_18;
and_19 = xor_19 & xor_20; and_20 = xor_0 & xor_21; and_21 = xor_22 & xor_23;
and_22 = xor_24 & xor_25; and_23 = xor_7 & xor_26; and_24 = xor_27 & xor_28;
and_25 = xor_29 & xor_30; and_26 = xor_10 & xor_8; and_27 = xor_10 & xor_4;
xor_41 = and_9 ^ and_26; xor_42 = x[4] ^ and_18; xor_43 = xor_42 ^ xor_5;
xor_44 = and_19 ^ xor_41; xor_45 = and_10 ^ xor_44; xor_46 = x[5] ^ and_20;
xor_47 = and_2 ^ and_21; xor_48 = and_4 ^ xor_45; xor_49 = and_23 ^ xor_48;
xor_50 = and_25 ^ xor_46; xor_51 = xor_43 ^ xor_47; xor_52 = xor_49 ^ xor_51;
xor_53 = and_15 ^ and_22; xor_54 = xor_50 ^ xor_53; xor_55 = xor_32 ^ xor_54;
xor_56 = xor_34 ^ xor_52; xor_57 = xor_36 ^ and_16; xor_58 = xor_41 ^ xor_57;
xor_59 = xor_38 ^ and_20; xor_60 = xor_52 ^ xor_59; xor_61 = xor_40 ^ and_16;
xor_62 = xor_44 ^ xor_61; xor_63 = xor_13 ^ and_17; xor_64 = and_24 ^ and_27;
xor_65 = xor_43 ^ xor_54; xor_66 = xor_63 ^ xor_64; xor_67 = xor_65 ^ xor_66;
y[0] = xor_55; y[1] = xor_56; y[2] = xor_58; y[3] = xor_60; y[4] = xor_62;
y[5] = xor_67;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 63]$):

```
0x00, 0x26, 0x12, 0x0c, 0x32, 0x36, 0x39, 0x2c, 0x15, 0x38, 0x14, 0x31, 0x3d, 0x03, 0x1b, 0x04
0x24, 0x23, 0x05, 0x19, 0x0d, 0x22, 0x3f, 0x3a, 0x1f, 0x08, 0x1d, 0x21, 0x33, 0x25, 0x3c, 0x30
0x3b, 0x0f, 0x13, 0x07, 0x11, 0x17, 0x2f, 0x20, 0x29, 0x0e, 0x1a, 0x2d, 0x2e, 0x0a, 0x35, 0x28
0x02, 0x37, 0x1c, 0x2a, 0x0b, 0x06, 0x01, 0x3e, 0x2b, 0x16, 0x1e, 0x10, 0x18, 0x34, 0x27, 0x09
```

Inverse netlist (28 AND, 78 XOR, 7 NOT, $D = 2$):

```
not_0 = ~x[0]; not_1 = ~x[2]; xor_0 = not_1 ^ x[3]; xor_1 = xor_0 ^ x[4];
xor_2 = xor_1 ^ not_0; xor_3 = x[1] ^ x[3]; xor_4 = xor_0 ^ x[1];
not_2 = ~xor_3; xor_5 = not_1 ^ x[4]; xor_6 = x[3] ^ x[5];
xor_7 = xor_2 ^ x[5]; xor_8 = xor_1 ^ x[5]; xor_9 = xor_3 ^ x[4];
and_0 = not_2 & xor_4; and_1 = xor_4 & xor_5; and_2 = not_2 & xor_5;
and_3 = xor_4 & xor_6; and_4 = not_2 & xor_6; and_5 = xor_5 & xor_6;
and_6 = xor_2 & xor_4; and_7 = not_2 & xor_2; and_8 = xor_2 & xor_5;
and_9 = xor_2 & xor_6; and_10 = xor_4 & xor_7; and_11 = not_2 & xor_7;
and_12 = xor_5 & xor_7; and_13 = xor_6 & xor_7; and_14 = xor_2 & xor_7;
xor_10 = and_0 ^ xor_4; xor_11 = and_1 ^ not_2; xor_12 = and_1 ^ xor_6;
xor_13 = and_2 ^ xor_7; xor_14 = and_2 ^ not_2; xor_15 = and_5 ^ xor_2;
xor_16 = and_6 ^ xor_6; xor_17 = and_6 ^ and_1; xor_18 = and_8 ^ and_0;
xor_19 = and_8 ^ xor_6; xor_20 = and_10 ^ xor_6; xor_21 = and_10 ^ and_7;
xor_22 = and_6 ^ and_11; xor_23 = and_11 ^ xor_4; xor_24 = and_11 ^ and_8;
xor_25 = and_7 ^ and_11; xor_26 = and_11 ^ xor_6; xor_27 = and_12 ^ xor_4;
xor_28 = and_13 ^ xor_2; xor_29 = and_14 ^ xor_6; xor_30 = and_14 ^ xor_4;
and_15 = and_6 & xor_13; and_16 = xor_14 & xor_16; and_17 = xor_8 & xor_17;
and_18 = xor_6 & xor_18; and_19 = xor_11 & xor_20; and_20 = xor_9 & xor_22;
and_21 = xor_12 & xor_25; and_22 = xor_26 & xor_21; and_23 = xor_27 & xor_28;
and_24 = xor_23 & xor_29; and_25 = xor_30 & xor_24; and_26 = xor_10 & xor_19;
and_27 = xor_10 & xor_15; xor_31 = and_16 ^ not_0; xor_32 = and_13 ^ and_19;
xor_33 = x[4] ^ x[5]; xor_34 = and_5 ^ and_12; xor_35 = and_8 ^ xor_31;
xor_36 = and_24 ^ and_17; xor_37 = and_27 ^ xor_36; xor_38 = xor_33 ^ xor_35;
xor_39 = and_3 ^ not_1; xor_40 = xor_32 ^ xor_39; xor_41 = and_9 ^ and_26;
xor_42 = xor_41 ^ xor_34; xor_43 = xor_37 ^ x[1]; xor_44 = and_14 ^ x[3];
xor_45 = and_2 ^ and_4; xor_46 = and_10 ^ and_21; xor_47 = and_15 ^ and_20;
xor_48 = and_25 ^ and_22; xor_49 = and_23 ^ xor_38; xor_50 = xor_45 ^ xor_46;
xor_51 = xor_47 ^ xor_48; xor_52 = xor_49 ^ xor_50; xor_53 = and_20 ^ not_0;
xor_54 = xor_53 ^ xor_40; xor_55 = xor_54 ^ xor_25; xor_56 = xor_44 ^ xor_55;
xor_57 = x[1] ^ x[4]; xor_58 = and_12 ^ xor_57; xor_59 = xor_58 ^ xor_40;
not_3 = ~xor_59; xor_60 = xor_32 ^ xor_38; xor_61 = xor_42 ^ xor_60;
not_4 = ~xor_61; xor_62 = and_6 ^ and_18; xor_63 = xor_31 ^ xor_62;
xor_64 = xor_42 ^ xor_63; xor_65 = xor_43 ^ xor_64; xor_66 = xor_51 ^ xor_65;
xor_67 = xor_43 ^ xor_25; xor_68 = xor_52 ^ xor_67; not_5 = ~xor_68;
xor_69 = and_0 ^ and_1; xor_70 = and_3 ^ xor_69; xor_71 = and_11 ^ xor_70;
xor_72 = and_19 ^ xor_71; xor_73 = xor_34 ^ xor_72; xor_74 = xor_73 ^ xor_37;
xor_75 = xor_44 ^ xor_74; xor_76 = xor_51 ^ xor_75; xor_77 = xor_52 ^ xor_76;
not_6 = ~xor_77; y[0] = xor_56; y[1] = not_3; y[2] = not_4; y[3] = xor_66;
y[4] = not_5; y[5] = not_6;
```

C.20 $\mathsf{YSH}_{7,1}$: 7-bit quartic, $\delta = 4$, $L = 32$ (7-bit, Category B)

- **Profile**: $\delta = 4$, $L = 32$ (NL = 48), $\deg(S) = 4$, $\mathcal{D}(S) = [2^{63}, 4^{64}]$, FP = 1, SAC deviation 0.125.
- **Cost**: 15 AND, 70 XOR, 10 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 133.00 μm^2 / 166.7 GE, delay 750.9 ps, logic depth 14; pipelined $d = 2$ DOM-indep: 88 `DFF_X1` [60 gadget + 28 sync], 786.30 μm^2 / 985.3 GE, stage delay 478.9 ps).

- **Remark**: Bipartite spectrum $[2^{63}, 4^{64}]$: its quadratic components form a hyperplane. Its AND gates are 14 at level 1 and one at level 2 (Section 6.6).
- **Inverse ($\mathsf{YSH}_{7,1}^{-1}$)**: $\deg(S^{-1}) = 5$, $\mathcal{D}(S^{-1}) = [4^7, 5^{120}]$; cost: 30 AND, 158 XOR, 12 NOT, AND depth $D = 3$ (Tier 2: Output shear unwinding + 29-AND $D = 2$ Gold inverse core; unmasked NanGate 45 nm: 290.47 μm^2 / 364.0 GE, delay 1511.7 ps, logic depth 24; pipelined $d = 2$ DOM-indep: 226 `DFF_X1` [120 gadget + 106 sync], 1851.89 μm^2 / 2320.7 GE, stage delay 759.6 ps).

Lookup table (LUT$[0 \dots 127]$):

```
0x3d, 0x7b, 0x7e, 0x34, 0x5b, 0x5d, 0x76, 0x7c, 0x5f, 0x2f, 0x4b, 0x37, 0x65, 0x55, 0x1f, 0x23
0x06, 0x56, 0x0c, 0x51, 0x3a, 0x2a, 0x5e, 0x42, 0x7d, 0x1b, 0x21, 0x4a, 0x1d, 0x3b, 0x2e, 0x04
0x79, 0x35, 0x32, 0x73, 0x25, 0x29, 0x00, 0x01, 0x69, 0x13, 0x75, 0x02, 0x68, 0x52, 0x1a, 0x2d
0x6d, 0x36, 0x6e, 0x38, 0x6a, 0x70, 0x07, 0x11, 0x64, 0x08, 0x31, 0x50, 0x3f, 0x12, 0x05, 0x24
0x7a, 0x19, 0x49, 0x26, 0x10, 0x33, 0x4d, 0x63, 0x2b, 0x7f, 0x4e, 0x16, 0x1c, 0x09, 0x17, 0x0f
0x45, 0x30, 0x3e, 0x46, 0x74, 0x41, 0x60, 0x58, 0x0d, 0x4f, 0x20, 0x6f, 0x61, 0x62, 0x22, 0x2c
0x77, 0x1e, 0x4c, 0x28, 0x27, 0x0e, 0x72, 0x57, 0x54, 0x0b, 0x39, 0x6b, 0x59, 0x47, 0x5a, 0x48
0x66, 0x18, 0x14, 0x67, 0x6c, 0x53, 0x71, 0x43, 0x5c, 0x15, 0x78, 0x3c, 0x0a, 0x03, 0x40, 0x44
```

Netlist (15 AND, 70 XOR, 10 NOT, $D = 2$):

```
not_0 = ~x[0]; xor_0 = x[1] ^ x[3]; xor_1 = x[2] ^ x[3]; not_1 = ~x[3];
xor_2 = not_0 ^ x[5]; xor_3 = x[4] ^ xor_0; xor_4 = x[2] ^ xor_0;
not_2 = ~xor_1; xor_5 = not_0 ^ not_1; xor_6 = not_1 ^ x[5];
xor_7 = not_1 ^ xor_2; xor_8 = x[0] ^ xor_3; xor_9 = x[6] ^ xor_3;
not_3 = ~xor_4; xor_10 = not_0 ^ not_2; xor_11 = not_2 ^ xor_2;
and_0 = not_2 & not_1; not_4 = ~xor_8; not_5 = ~xor_9; xor_12 = not_3 ^ xor_2;
xor_13 = not_3 ^ xor_5; xor_14 = x[5] ^ x[2]; and_1 = not_0 & x[2];
xor_15 = not_2 ^ and_0; xor_16 = not_3 ^ not_4; xor_17 = not_4 ^ xor_2;
xor_18 = not_1 ^ not_4; xor_19 = not_4 ^ xor_5; xor_20 = not_5 ^ xor_2;
xor_21 = not_0 ^ not_5; xor_22 = not_3 ^ not_5; xor_23 = not_5 ^ x[2];
and_2 = not_4 & not_5; and_3 = xor_10 & xor_12; and_4 = xor_13 & xor_11;
xor_24 = not_1 ^ and_1; xor_25 = not_4 ^ and_1; xor_26 = not_3 ^ xor_15;
xor_27 = not_2 ^ xor_16; xor_28 = not_5 ^ xor_16; and_5 = not_3 & xor_17;
and_6 = xor_18 & xor_2; and_7 = xor_19 & xor_6; and_8 = not_1 & xor_20;
and_9 = xor_16 & xor_21; and_10 = xor_11 & xor_22; and_11 = xor_13 & xor_23;
xor_29 = and_3 ^ xor_15; and_12 = xor_27 & xor_14; and_13 = xor_7 & xor_28;
xor_30 = and_2 ^ and_5; xor_31 = not_2 ^ and_5; xor_32 = not_0 ^ and_6;
xor_33 = and_8 ^ and_7; xor_34 = not_4 ^ and_8; xor_35 = and_9 ^ and_4;
xor_36 = and_10 ^ and_11; xor_37 = and_10 ^ and_13; xor_38 = and_3 ^ and_13;
xor_39 = xor_30 ^ xor_26; xor_40 = and_6 ^ xor_30; xor_41 = and_12 ^ xor_32;
xor_42 = xor_32 ^ xor_25; xor_43 = xor_33 ^ xor_24; xor_44 = and_12 ^ xor_33;
xor_45 = not_5 ^ xor_35; xor_46 = and_4 ^ xor_36; xor_47 = xor_35 ^ xor_36;
xor_48 = xor_35 ^ xor_37; xor_49 = xor_34 ^ xor_40; xor_50 = x[5] ^ xor_41;
xor_51 = xor_31 ^ xor_41; xor_52 = xor_38 ^ xor_42; xor_53 = xor_29 ^ xor_43;
xor_54 = xor_36 ^ xor_44; xor_55 = xor_29 ^ xor_45; xor_56 = xor_43 ^ xor_46;
xor_57 = xor_43 ^ xor_48; xor_58 = xor_47 ^ xor_50; xor_59 = xor_50 ^ xor_53;
xor_60 = xor_55 ^ xor_54; xor_61 = xor_55 ^ xor_49; xor_62 = xor_51 ^ xor_56;
xor_63 = xor_39 ^ xor_58; not_6 = ~xor_59; xor_64 = xor_62 ^ xor_60;
xor_65 = xor_62 ^ xor_52; xor_66 = xor_63 ^ xor_60; not_7 = ~xor_63;
xor_67 = xor_63 ^ xor_61; not_8 = ~xor_64; xor_68 = xor_57 ^ xor_66;
not_9 = ~xor_67; and_14 = not_6 & not_8; xor_69 = xor_66 ^ and_14;
y[0] = xor_69; y[1] = not_6; y[2] = not_8; y[3] = not_7; y[4] = xor_68;
y[5] = xor_65; y[6] = not_9;
```

Inverse lookup table (LUT$_{S^{-1}}[0 \dots 127]$):

```
0x26, 0x27, 0x2b, 0x7d, 0x1f, 0x3e, 0x10, 0x36, 0x39, 0x4d, 0x7c, 0x69, 0x12, 0x58, 0x65, 0x4f
0x44, 0x37, 0x3d, 0x29, 0x72, 0x79, 0x4b, 0x4e, 0x71, 0x41, 0x2e, 0x19, 0x4c, 0x1c, 0x61, 0x0e
0x5a, 0x1a, 0x5e, 0x0f, 0x3f, 0x24, 0x43, 0x64, 0x63, 0x25, 0x15, 0x48, 0x5f, 0x2f, 0x1e, 0x09
0x51, 0x3a, 0x22, 0x45, 0x03, 0x21, 0x31, 0x0b, 0x33, 0x6a, 0x14, 0x1d, 0x7b, 0x00, 0x52, 0x3c
0x7e, 0x55, 0x17, 0x77, 0x7f, 0x50, 0x53, 0x6d, 0x6f, 0x42, 0x1b, 0x0a, 0x62, 0x46, 0x4a, 0x59
0x3b, 0x13, 0x2d, 0x75, 0x68, 0x0d, 0x11, 0x67, 0x57, 0x6c, 0x6e, 0x04, 0x78, 0x05, 0x16, 0x08
0x56, 0x5c, 0x5d, 0x47, 0x38, 0x0c, 0x70, 0x73, 0x2c, 0x28, 0x34, 0x6b, 0x74, 0x30, 0x32, 0x5b
0x35, 0x76, 0x66, 0x23, 0x54, 0x2a, 0x06, 0x60, 0x7a, 0x20, 0x40, 0x01, 0x07, 0x18, 0x02, 0x49
```

Inverse netlist (30 AND, 158 XOR, 12 NOT, $D = 3$):

```
and_0 = x[1] & x[2]; xor_0 = and_0 ^ x[0]; not_0 = ~x[1]; not_1 = ~x[2];
not_2 = ~x[3]; not_3 = ~x[5]; not_4 = ~x[6]; xor_1 = not_1 ^ x[4];
xor_2 = not_3 ^ xor_0; xor_3 = not_2 ^ not_4; xor_4 = not_0 ^ not_3;
xor_5 = not_0 ^ xor_2; xor_6 = xor_1 ^ xor_3; xor_7 = xor_0 ^ x[4];
xor_8 = not_2 ^ xor_1; xor_9 = not_4 ^ xor_5; xor_10 = xor_0 ^ xor_1;
xor_11 = not_2 ^ xor_4; xor_12 = xor_2 ^ xor_3; xor_13 = not_1 ^ xor_0;
```

```
xor_14 = xor_0 ^ xor_8; xor_15 = not_2 ^ xor_5; xor_16 = not_3 ^ not_4;
xor_17 = xor_1 ^ xor_16; xor_18 = x[4] ^ xor_11; xor_19 = xor_4 ^ xor_6;
xor_20 = x[4] ^ xor_9; xor_21 = not_3 ^ xor_6; xor_22 = not_0 ^ not_4;
xor_23 = xor_22 ^ xor_7; xor_24 = not_2 ^ xor_13; xor_25 = xor_2 ^ xor_6;
xor_26 = xor_1 ^ xor_2; xor_27 = x[4] ^ xor_3; xor_28 = xor_3 ^ xor_7;
xor_29 = not_0 ^ xor_8; xor_30 = not_1 ^ xor_5; xor_31 = xor_3 ^ xor_4;
xor_32 = not_4 ^ xor_10; xor_33 = not_1 ^ xor_12; xor_34 = not_0 ^ xor_13;
xor_35 = not_1 ^ not_2; xor_36 = xor_2 ^ xor_35; and_1 = xor_14 & xor_15;
and_2 = xor_17 & xor_18; and_3 = not_3 & xor_19; and_4 = xor_8 & xor_9;
and_5 = xor_14 & xor_20; and_6 = xor_21 & xor_23; and_7 = xor_24 & xor_25;
and_8 = xor_23 & xor_25; and_9 = xor_26 & xor_27; and_10 = x[4] & xor_7;
and_11 = xor_7 & xor_10; and_12 = xor_28 & xor_4; and_13 = xor_29 & xor_30;
and_14 = not_0 & xor_2; xor_37 = and_1 ^ xor_31; xor_38 = and_9 ^ xor_37;
xor_39 = xor_32 ^ xor_38; xor_40 = and_2 ^ xor_12; xor_41 = and_4 ^ and_5;
xor_42 = xor_17 ^ xor_39; xor_43 = and_13 ^ xor_40; xor_44 = and_8 ^ xor_11;
xor_45 = and_3 ^ and_6; xor_46 = xor_41 ^ xor_45; xor_47 = and_12 ^ xor_42;
xor_48 = and_11 ^ xor_44; xor_49 = xor_41 ^ xor_43; xor_50 = and_8 ^ xor_33;
xor_51 = xor_40 ^ xor_50; xor_52 = xor_46 ^ xor_51; xor_53 = and_2 ^ and_10;
xor_54 = and_14 ^ xor_53; xor_55 = xor_46 ^ xor_54; xor_56 = xor_47 ^ xor_55;
xor_57 = xor_48 ^ xor_56; not_5 = ~xor_57; xor_58 = and_7 ^ xor_47;
xor_59 = xor_49 ^ xor_58; xor_60 = xor_34 ^ xor_42; xor_61 = xor_43 ^ xor_60;
xor_62 = xor_48 ^ xor_61; not_6 = ~xor_62; xor_63 = and_3 ^ xor_36;
xor_64 = and_7 ^ xor_63; xor_65 = and_12 ^ xor_64; xor_66 = and_14 ^ xor_65;
xor_67 = xor_39 ^ xor_66; not_7 = ~xor_67; xor_68 = and_10 ^ xor_38;
xor_69 = xor_49 ^ xor_68; not_8 = ~xor_69; xor_70 = and_4 ^ xor_14;
xor_71 = and_6 ^ xor_70; xor_72 = and_7 ^ xor_71; xor_73 = and_9 ^ xor_72;
xor_74 = xor_44 ^ xor_73; not_9 = ~xor_74; xor_75 = not_7 ^ xor_52;
xor_76 = not_6 ^ xor_52; xor_77 = not_7 ^ xor_59; xor_78 = not_8 ^ not_9;
xor_79 = not_5 ^ not_7; xor_80 = not_5 ^ not_6; xor_81 = not_8 ^ xor_59;
xor_82 = not_5 ^ xor_59; xor_83 = not_7 ^ not_8; xor_84 = not_6 ^ not_8;
xor_85 = xor_59 ^ xor_76; xor_86 = not_5 ^ xor_76; xor_87 = not_8 ^ xor_75;
xor_88 = xor_59 ^ xor_75; xor_89 = not_9 ^ xor_77; xor_90 = not_6 ^ not_7;
xor_91 = not_9 ^ xor_90; xor_92 = not_9 ^ xor_76; xor_93 = xor_52 ^ xor_78;
xor_94 = not_8 ^ xor_79; xor_95 = not_9 ^ xor_59; and_15 = not_5 & not_6;
and_16 = xor_75 & xor_80; and_17 = xor_52 & xor_81; and_18 = xor_82 & xor_83;
and_19 = xor_77 & xor_84; and_20 = xor_59 & xor_78; and_21 = not_9 & xor_85;
and_22 = xor_84 & xor_86; and_23 = not_5 & xor_87; and_24 = not_6 & xor_88;
and_25 = xor_87 & xor_89; and_26 = xor_86 & xor_91; and_27 = xor_79 & xor_92;
and_28 = xor_75 & xor_93; and_29 = xor_94 & xor_95; xor_96 = and_15 ^ and_18;
xor_97 = and_24 ^ xor_96; xor_98 = and_6 ^ and_8; xor_99 = and_28 ^ xor_97;
xor_100 = and_16 ^ and_9; xor_101 = and_17 ^ xor_98; xor_102 = not_0 ^ xor_0;
xor_103 = and_11 ^ not_4; xor_104 = and_7 ^ xor_99; xor_105 = and_13 ^ and_25;
xor_106 = and_4 ^ xor_100; xor_107 = and_1 ^ xor_101;
xor_108 = and_22 ^ and_27; xor_109 = xor_103 ^ xor_107; xor_110 = not_3 ^ x[4];
xor_111 = and_3 ^ and_26; xor_112 = and_14 ^ and_10; xor_113 = and_19 ^ and_23;
xor_114 = xor_102 ^ xor_113; xor_115 = xor_106 ^ xor_108;
xor_116 = and_2 ^ and_5; xor_117 = xor_109 ^ xor_111;
xor_118 = xor_116 ^ xor_117; xor_119 = and_29 ^ xor_104;
xor_120 = xor_110 ^ xor_119; xor_121 = and_20 ^ xor_105;
xor_122 = and_21 ^ xor_112; xor_123 = and_14 ^ xor_104;
xor_124 = xor_115 ^ xor_123; xor_125 = xor_118 ^ xor_124; not_10 = ~xor_125;
xor_126 = and_12 ^ xor_4; xor_127 = and_17 ^ xor_126;
xor_128 = xor_127 ^ xor_96; xor_129 = xor_105 ^ xor_128;
xor_130 = xor_115 ^ xor_129; xor_131 = xor_122 ^ xor_130; not_11 = ~xor_131;
xor_132 = xor_105 ^ xor_97; xor_133 = xor_114 ^ xor_132;
xor_134 = xor_118 ^ xor_133; xor_135 = not_1 ^ xor_101;
xor_136 = xor_106 ^ xor_135; xor_137 = xor_114 ^ xor_136;
xor_138 = xor_120 ^ xor_137; xor_139 = and_23 ^ xor_100;
xor_140 = xor_108 ^ xor_139; xor_141 = xor_109 ^ xor_140;
xor_142 = xor_120 ^ xor_141; xor_143 = xor_121 ^ xor_142;
xor_144 = and_1 ^ and_9; xor_145 = and_25 ^ xor_144;
xor_146 = xor_145 ^ xor_99; xor_147 = xor_102 ^ xor_146;
xor_148 = xor_110 ^ xor_147; xor_149 = xor_111 ^ xor_148;
xor_150 = xor_112 ^ xor_149; xor_151 = and_16 ^ and_3;
xor_152 = xor_151 ^ xor_98; xor_153 = xor_103 ^ xor_152;
xor_154 = xor_104 ^ xor_153; xor_155 = xor_114 ^ xor_154;
xor_156 = xor_121 ^ xor_155; xor_157 = xor_122 ^ xor_156; y[0] = not_10;
y[1] = not_11; y[2] = xor_134; y[3] = xor_138; y[4] = xor_143; y[5] = xor_150;
y[6] = xor_157;
```

C.21 7-bit swapped Gold $x^5_{\mathbf{swap(1,25)}}$ class, $\delta = 4$, $L = 20$ (7-bit, Category A)

- **Profile**: $\delta = 4$, $L = 20$ (NL = 54), $\deg(S) = 6$, $\mathcal{D}(S) = [2^{63}, 6^{64}]$, FP = 2, SAC deviation 0.0312.

- **Cost**: 51 AND, 76 XOR, 1 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 176.09 μm^2 / 220.7 GE, delay 840.4 ps, logic depth 16; pipelined $d = 2$ DOM-indep: 284 `DFF_X1` [204 gadget + 80 sync], 2070.01 μm^2 / 2594.0 GE, stage delay 802.7 ps).
- **Remark**: Certified affine equivalent to the two-point-swapped Gold permutation $x^5_{\mathrm{swap}(1,25)}$ over $\mathbb{F}_2[x]/(x^7+x+1)$ [BKRS22] ($\sim_{\mathrm{aff}}$ `GF128_Gold_x5_swap(1,25)`, with $A =$ [`0x5f`, `0x2a`, `0x76`, `0x32`, `0x2f`, `0x53`, `0x40`], $a =$ `0x30`, $B =$ [`0x17`, `0x1e`, `0x37`, `0x6e`, `0x5c`, `0x21`, `0x73`], $b =$ `0x69`). Degree 6, the maximum for a 7-bit permutation, at $\delta = 4$, $L = 20$.
- **Inverse (S^{-1}, $\sim_{\mathbf{aff}}$ $(x^5_{\mathbf{swap}(1,25)})^{-1}$)**: $\deg(S^{-1}) = 6$, $\mathcal{D}(S^{-1}) = [4^{63}, 6^{64}]$; cost: 34 AND, 164 XOR, 9 NOT, AND depth $D = 3$ (Tier 2: Parallel Transposition Inversion on 29-AND $D = 2$ Gold inverse core; unmasked NanGate 45 nm: 302.71 μm^2 / 379.3 GE, delay 1263.9 ps, logic depth 21; pipelined $d = 2$ DOM-indep: 216 `DFF_X1` [136 gadget + 80 sync], 1866.79 μm^2 / 2339.3 GE, stage delay 698.8 ps).

Lookup table (LUT[0 . . . 127]):

```
0x04, 0x5e, 0x34, 0x57, 0x73, 0x7e, 0x05, 0x53, 0x48, 0x60, 0x75, 0x64, 0x21, 0x3c, 0x5a, 0x1c
0x4c, 0x2d, 0x6f, 0x37, 0x1a, 0x4e, 0x7f, 0x12, 0x35, 0x26, 0x1b, 0x31, 0x7d, 0x5b, 0x15, 0x0a
0x47, 0x2f, 0x40, 0x11, 0x42, 0x1f, 0x03, 0x67, 0x14, 0x0e, 0x1e, 0x3d, 0x0f, 0x20, 0x43, 0x55
0x45, 0x16, 0x51, 0x3b, 0x61, 0x07, 0x33, 0x6c, 0x23, 0x02, 0x3a, 0x22, 0x19, 0x0d, 0x46, 0x6b
0x5d, 0x06, 0x30, 0x52, 0x50, 0x3e, 0x7b, 0x2c, 0x39, 0x10, 0x59, 0x49, 0x2a, 0x36, 0x0c, 0x29
0x09, 0x69, 0x77, 0x2e, 0x25, 0x70, 0x1d, 0x71, 0x58, 0x4a, 0x2b, 0x00, 0x6a, 0x4d, 0x5f, 0x41
0x68, 0x01, 0x32, 0x62, 0x17, 0x4b, 0x0b, 0x6e, 0x13, 0x08, 0x44, 0x66, 0x72, 0x5c, 0x63, 0x74
0x76, 0x24, 0x3f, 0x54, 0x28, 0x4f, 0x27, 0x79, 0x38, 0x18, 0x7c, 0x65, 0x78, 0x6d, 0x7a, 0x56
```

Netlist (51 AND, 76 XOR, 1 NOT, $D = 3$):

```
xor_0 = x[3] ^ x[4]; and_0 = x[0] & x[1]; and_1 = x[0] & x[2];
and_2 = x[1] & x[2]; and_3 = x[0] & x[3]; and_4 = x[1] & x[3];
and_5 = x[2] & x[3]; and_6 = x[0] & x[4]; and_7 = x[1] & x[4];
and_8 = x[2] & x[4]; and_9 = x[3] & x[4]; and_10 = x[0] & x[5];
and_11 = x[1] & x[5]; and_12 = x[2] & x[5]; and_13 = x[3] & x[5];
and_14 = x[4] & x[5]; and_15 = x[0] & x[6]; and_16 = x[1] & x[6];
and_17 = x[2] & x[6]; and_18 = x[3] & x[6]; and_19 = x[4] & x[6];
and_20 = x[5] & x[6]; and_21 = x[0] & and_2; and_22 = x[0] & and_5;
and_23 = x[0] & and_8; and_24 = and_0 & and_8; and_25 = and_1 & and_9;
and_26 = and_8 & x[5]; and_27 = and_9 & x[5]; and_28 = and_8 & x[6];
and_29 = and_9 & x[6]; and_30 = x[0] & and_12; and_31 = and_0 & and_12;
and_32 = and_1 & and_13; and_33 = and_1 & and_14; and_34 = and_12 & x[6];
and_35 = and_13 & x[6]; and_36 = and_14 & x[6]; and_37 = x[0] & and_17;
and_38 = and_0 & and_17; and_39 = and_1 & and_18; and_40 = and_1 & and_19;
and_41 = and_1 & and_20; and_42 = and_0 & and_26; and_43 = and_1 & and_27;
and_44 = and_0 & and_28; and_45 = and_1 & and_29; and_46 = and_0 & and_34;
and_47 = and_1 & and_35; and_48 = and_1 & and_36; and_49 = and_21 & and_36;
and_50 = and_22 & and_36; xor_1 = x[2] ^ and_11; xor_2 = and_17 ^ and_20;
xor_3 = and_12 ^ xor_2; xor_4 = and_1 ^ and_13; xor_5 = and_9 ^ xor_1;
xor_6 = x[6] ^ and_16; xor_7 = and_0 ^ and_6; xor_8 = and_3 ^ xor_3;
xor_9 = and_41 ^ and_46; xor_10 = and_39 ^ and_40; xor_11 = and_49 ^ xor_10;
xor_12 = and_38 ^ and_47; xor_13 = and_23 ^ and_24; xor_14 = xor_11 ^ xor_13;
xor_15 = and_45 ^ xor_12; xor_16 = and_21 ^ xor_15; xor_17 = and_42 ^ xor_16;
xor_18 = and_43 ^ xor_8; xor_19 = x[0] ^ and_14; xor_20 = and_30 ^ and_31;
xor_21 = and_25 ^ and_44; xor_22 = and_37 ^ and_48; xor_23 = xor_18 ^ xor_22;
xor_24 = xor_14 ^ xor_17; xor_25 = xor_9 ^ xor_24; xor_26 = and_33 ^ and_50;
xor_27 = xor_21 ^ xor_25; xor_28 = xor_5 ^ xor_7; xor_29 = and_22 ^ xor_27;
xor_30 = xor_20 ^ xor_26; xor_31 = xor_23 ^ xor_30; xor_32 = xor_29 ^ xor_31;
xor_33 = and_7 ^ xor_4; xor_34 = and_5 ^ and_19; xor_35 = and_32 ^ xor_32;
xor_36 = xor_6 ^ xor_19; xor_37 = xor_6 ^ xor_33; xor_38 = and_4 ^ xor_34;
xor_39 = x[1] ^ and_10; xor_40 = xor_28 ^ xor_37; xor_41 = and_2 ^ xor_35;
xor_42 = x[5] ^ xor_41; xor_43 = xor_36 ^ xor_0; xor_44 = x[5] ^ and_4;
xor_45 = and_8 ^ and_15; xor_46 = xor_40 ^ xor_44; xor_47 = xor_45 ^ xor_46;
xor_48 = and_5 ^ and_6; xor_49 = and_10 ^ xor_1; xor_50 = xor_19 ^ xor_33;
xor_51 = xor_48 ^ xor_49; xor_52 = xor_50 ^ xor_51; xor_53 = xor_52 ^ xor_42;
xor_54 = x[3] ^ and_2; xor_55 = and_16 ^ and_20; xor_56 = xor_4 ^ xor_5;
xor_57 = xor_38 ^ xor_54; xor_58 = xor_55 ^ xor_56; xor_59 = xor_57 ^ xor_58;
xor_60 = and_13 ^ and_17; xor_61 = and_18 ^ xor_7; xor_62 = xor_38 ^ xor_43;
xor_63 = xor_60 ^ xor_61; xor_64 = xor_62 ^ xor_63; xor_65 = x[0] ^ xor_8;
xor_66 = xor_34 ^ xor_39; xor_67 = xor_40 ^ xor_65; xor_68 = xor_66 ^ xor_67;
xor_69 = and_8 ^ and_18; xor_70 = xor_28 ^ xor_39; xor_71 = xor_69 ^ xor_70;
xor_72 = xor_71 ^ xor_35; xor_73 = x[2] ^ and_1; xor_74 = xor_43 ^ xor_73;
xor_75 = xor_74 ^ xor_42; not_0 = ~xor_59; y[0] = xor_47; y[1] = xor_53;
y[2] = not_0; y[3] = xor_64; y[4] = xor_68; y[5] = xor_72; y[6] = xor_75;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 127]$):

```
0x5b, 0x61, 0x39, 0x26, 0x00, 0x06, 0x41, 0x35, 0x69, 0x50, 0x1f, 0x66, 0x4e, 0x3d, 0x29, 0x2c
0x49, 0x23, 0x17, 0x68, 0x28, 0x1e, 0x31, 0x64, 0x79, 0x3c, 0x14, 0x1a, 0x0f, 0x56, 0x2a, 0x25
0x2d, 0x0c, 0x3b, 0x38, 0x71, 0x54, 0x19, 0x76, 0x74, 0x4f, 0x4c, 0x5a, 0x47, 0x11, 0x53, 0x21
0x42, 0x1b, 0x62, 0x36, 0x02, 0x18, 0x4d, 0x13, 0x78, 0x48, 0x3a, 0x33, 0x0d, 0x2b, 0x45, 0x72
0x22, 0x5f, 0x24, 0x2e, 0x6a, 0x30, 0x3e, 0x20, 0x08, 0x4b, 0x59, 0x65, 0x10, 0x5d, 0x15, 0x75
0x44, 0x32, 0x43, 0x07, 0x73, 0x2f, 0x7f, 0x03, 0x58, 0x4a, 0x0e, 0x1d, 0x6d, 0x40, 0x01, 0x5e
0x09, 0x34, 0x63, 0x6e, 0x0b, 0x7b, 0x6b, 0x27, 0x60, 0x51, 0x5c, 0x3f, 0x37, 0x7d, 0x67, 0x12
0x55, 0x57, 0x6c, 0x04, 0x6f, 0x0a, 0x70, 0x52, 0x7c, 0x77, 0x7e, 0x46, 0x7a, 0x1c, 0x05, 0x16
```

Inverse netlist (34 AND, 164 XOR, 9 NOT, $D = 3$):

```
not_0 = ~x[0]; not_1 = ~x[3]; not_2 = ~x[5]; not_3 = ~x[6];
xor_0 = x[1] ^ x[2]; xor_1 = not_1 ^ x[4]; xor_2 = not_0 ^ not_3;
xor_3 = not_2 ^ xor_0; xor_4 = not_0 ^ xor_1; xor_5 = x[2] ^ xor_2;
xor_6 = x[1] ^ xor_1; xor_7 = not_3 ^ xor_0; xor_8 = not_1 ^ not_2;
xor_9 = not_2 ^ xor_5; xor_10 = xor_0 ^ xor_2; xor_11 = not_2 ^ xor_1;
xor_12 = xor_1 ^ xor_7; xor_13 = not_0 ^ xor_0; xor_14 = xor_1 ^ xor_5;
xor_15 = not_1 ^ xor_3; xor_16 = xor_3 ^ xor_4; xor_17 = x[2] ^ xor_4;
xor_18 = not_3 ^ xor_8; xor_19 = not_3 ^ xor_3; xor_20 = not_0 ^ x[4];
xor_21 = xor_20 ^ xor_3; xor_22 = xor_2 ^ xor_6; xor_23 = xor_0 ^ xor_1;
xor_24 = x[4] ^ xor_9; xor_25 = not_3 ^ x[2]; xor_26 = x[4] ^ xor_0;
xor_27 = not_3 ^ xor_6; xor_28 = x[1] ^ x[4]; xor_29 = not_2 ^ xor_28;
xor_30 = not_3 ^ xor_29; xor_31 = xor_0 ^ xor_4; and_0 = xor_10 & xor_4;
and_1 = xor_0 & xor_11; and_2 = xor_12 & xor_13; and_3 = xor_14 & xor_2;
and_4 = xor_15 & xor_16; and_5 = xor_17 & xor_7; and_6 = xor_18 & xor_19;
and_7 = x[2] & xor_21; and_8 = xor_22 & xor_8; and_9 = xor_10 & xor_9;
and_10 = xor_23 & xor_24; and_11 = x[4] & xor_25; and_12 = xor_18 & xor_25;
and_13 = xor_26 & xor_6; xor_32 = and_7 ^ xor_4; xor_33 = and_5 ^ and_9;
xor_34 = and_11 ^ and_12; xor_35 = and_3 ^ xor_13; xor_36 = and_2 ^ xor_10;
xor_37 = xor_33 ^ xor_36; xor_38 = xor_32 ^ xor_35; xor_39 = and_4 ^ xor_34;
xor_40 = and_0 ^ and_10; xor_41 = xor_37 ^ xor_38; xor_42 = and_6 ^ and_8;
xor_43 = xor_11 ^ xor_40; xor_44 = and_13 ^ xor_37; xor_45 = xor_39 ^ xor_44;
xor_46 = xor_27 ^ xor_40; xor_47 = xor_41 ^ xor_46; xor_48 = xor_34 ^ xor_41;
xor_49 = xor_42 ^ xor_48; xor_50 = and_0 ^ xor_30; xor_51 = and_1 ^ xor_50;
xor_52 = xor_38 ^ xor_51; xor_53 = xor_39 ^ xor_52; xor_54 = xor_42 ^ xor_53;
xor_55 = and_6 ^ xor_32; xor_56 = xor_33 ^ xor_55; xor_57 = xor_39 ^ xor_56;
xor_58 = xor_43 ^ xor_57; xor_59 = and_2 ^ xor_0; xor_60 = and_10 ^ xor_59;
xor_61 = and_13 ^ xor_60; xor_62 = xor_35 ^ xor_61; xor_63 = and_1 ^ xor_31;
xor_64 = and_3 ^ xor_63; xor_65 = and_4 ^ xor_64; xor_66 = and_8 ^ xor_65;
xor_67 = and_9 ^ xor_66; xor_68 = and_12 ^ xor_67; xor_69 = xor_43 ^ xor_68;
xor_70 = xor_45 ^ xor_58; xor_71 = xor_45 ^ xor_54; xor_72 = xor_49 ^ xor_58;
xor_73 = xor_62 ^ xor_69; xor_74 = xor_47 ^ xor_58; xor_75 = xor_47 ^ xor_54;
xor_76 = xor_49 ^ xor_62; xor_77 = xor_47 ^ xor_49; xor_78 = xor_58 ^ xor_62;
xor_79 = xor_54 ^ xor_62; xor_80 = xor_49 ^ xor_71; xor_81 = xor_47 ^ xor_71;
xor_82 = xor_62 ^ xor_70; xor_83 = xor_49 ^ xor_70; xor_84 = xor_69 ^ xor_72;
xor_85 = xor_54 ^ xor_58; xor_86 = xor_69 ^ xor_85; xor_87 = xor_69 ^ xor_71;
xor_88 = xor_45 ^ xor_73; xor_89 = xor_62 ^ xor_74; xor_90 = xor_49 ^ xor_69;
and_14 = xor_47 & xor_54; and_15 = xor_70 & xor_75; and_16 = xor_45 & xor_76;
and_17 = xor_77 & xor_78; and_18 = xor_72 & xor_79; and_19 = xor_49 & xor_73;
and_20 = xor_69 & xor_80; and_21 = xor_79 & xor_81; and_22 = xor_47 & xor_82;
and_23 = xor_54 & xor_83; and_24 = xor_82 & xor_84; and_25 = xor_81 & xor_86;
and_26 = xor_74 & xor_87; and_27 = xor_70 & xor_88; and_28 = xor_89 & xor_90;
xor_91 = and_6 ^ not_2; xor_92 = and_15 ^ and_7; xor_93 = and_21 ^ and_26;
xor_94 = and_27 ^ and_9; xor_95 = and_10 ^ not_0; xor_96 = and_12 ^ xor_1;
xor_97 = and_0 ^ xor_92; xor_98 = and_1 ^ xor_91; xor_99 = and_19 ^ xor_93;
xor_100 = and_28 ^ and_3; xor_101 = and_16 ^ xor_94; xor_102 = xor_95 ^ xor_96;
xor_103 = and_5 ^ x[1]; xor_104 = x[2] ^ xor_99; xor_105 = and_2 ^ and_22;
xor_106 = and_23 ^ xor_97; xor_107 = and_24 ^ xor_98;
xor_108 = and_13 ^ and_20; xor_109 = xor_100 ^ xor_104;
xor_110 = xor_101 ^ xor_108; xor_111 = and_14 ^ and_17;
xor_112 = and_25 ^ xor_102; xor_113 = and_18 ^ not_3;
xor_114 = xor_103 ^ xor_105; xor_115 = xor_106 ^ xor_107;
xor_116 = xor_93 ^ xor_98; xor_117 = xor_106 ^ xor_116;
xor_118 = xor_110 ^ xor_117; xor_119 = xor_112 ^ xor_118; not_4 = ~xor_119;
xor_120 = and_8 ^ not_2; xor_121 = and_11 ^ xor_120;
xor_122 = xor_121 ^ xor_92; xor_123 = xor_102 ^ xor_122;
xor_124 = xor_109 ^ xor_123; xor_125 = xor_111 ^ xor_124;
xor_126 = and_5 ^ not_0; xor_127 = and_15 ^ xor_126;
xor_128 = and_22 ^ xor_127; xor_129 = xor_128 ^ xor_91;
xor_130 = xor_109 ^ xor_129; xor_131 = xor_110 ^ xor_130;
xor_132 = xor_113 ^ xor_131; xor_133 = and_24 ^ xor_95;
xor_134 = xor_133 ^ xor_97; xor_135 = xor_100 ^ xor_134;
xor_136 = xor_101 ^ xor_135; xor_137 = xor_111 ^ xor_136;
xor_138 = xor_114 ^ xor_137; xor_139 = and_18 ^ and_3;
xor_140 = and_19 ^ xor_139; xor_141 = xor_140 ^ xor_94;
xor_142 = xor_141 ^ xor_96; xor_143 = xor_105 ^ xor_142;
xor_144 = xor_115 ^ xor_143; not_5 = ~xor_144; xor_145 = and_20 ^ and_27;
```

```
xor_146 = xor_145 ^ xor_99; xor_147 = xor_103 ^ xor_146;
xor_148 = xor_107 ^ xor_147; xor_149 = xor_112 ^ xor_148;
xor_150 = and_10 ^ and_4; xor_151 = and_11 ^ xor_150;
xor_152 = and_16 ^ xor_151; xor_153 = and_25 ^ xor_152;
xor_154 = and_28 ^ xor_153; xor_155 = xor_1 ^ xor_154;
xor_156 = xor_104 ^ xor_155; xor_157 = xor_113 ^ xor_156;
xor_158 = xor_114 ^ xor_157; xor_159 = xor_115 ^ xor_158; not_6 = ~xor_159;
xor_160 = x[1] ^ x[5]; not_7 = ~xor_160; xor_161 = x[5] ^ x[6];
not_8 = ~xor_161; and_29 = not_0 & x[2]; and_30 = x[3] & x[4];
and_31 = not_7 & not_8; and_32 = and_29 & and_30; and_33 = and_32 & and_31;
xor_162 = and_33 ^ xor_125; xor_163 = and_33 ^ xor_138; y[0] = not_4;
y[1] = xor_162; y[2] = xor_132; y[3] = xor_163; y[4] = not_5; y[5] = xor_149;
y[6] = not_6;
```

C.22 7-bit Gold APN x^3 (7-bit, Category A)

- **Profile**: $\delta = 2$, $L = 16$ (NL = 56), $\deg(S) = 2$, $\mathcal{D}(S) = [2^{127}]$, FP = 2, SAC deviation 0.5.
- **Cost**: 14 AND, 57 XOR, 0 NOT, AND depth $D = 1$ (unmasked NanGate 45 nm: 105.87 μm^2 / 132.7 GE, delay 558.8 ps, logic depth 9; pipelined $d = 2$ DOM-indep: 72 `DFF_X1` [56 gadget + 16 sync], 656.49 μm^2 / 822.7 GE, stage delay 409.5 ps).
- **Remark**: The exact lookup table of x^3 over $\mathbb{F}_2[x]/(x^7 + x + 1)$, at depth $D = 1$; 11/45/1 in [BDM26a] and 11/41/1 (LTP = 6) in [KBJ+26]: three more AND gates and 12 (resp. 16) more XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 4$, $\mathcal{D}(S^{-1}) = [4^{127}]$; cost: 29 AND, 149 XOR, 15 NOT, AND depth $D = 2$ (Tier 3: 2-fold Gold quadratic composition $((y^5)^9)^{16}$ at $D = 2$; unmasked NanGate 45 nm: 276.64 μm^2 / 346.7 GE, delay 1191.7 ps, logic depth 22; pipelined $d = 2$ DOM-indep: 166 `DFF_X1` [116 gadget + 50 sync], 1542.80 μm^2 / 1933.3 GE, stage delay 772.0 ps).

Lookup table (LUT[0 . . . 127]):

```
0x00, 0x01, 0x08, 0x0f, 0x40, 0x55, 0x78, 0x6b, 0x0c, 0x45, 0x27, 0x68, 0x49, 0x14, 0x52, 0x09
0x60, 0x77, 0x24, 0x35, 0x3e, 0x3d, 0x4a, 0x4f, 0x44, 0x1b, 0x23, 0x7a, 0x1f, 0x54, 0x48, 0x05
0x0a, 0x33, 0x31, 0x0e, 0x26, 0x0b, 0x2d, 0x06, 0x75, 0x04, 0x6d, 0x1a, 0x5c, 0x39, 0x74, 0x17
0x2c, 0x03, 0x5b, 0x72, 0x1e, 0x25, 0x59, 0x64, 0x7b, 0x1c, 0x2f, 0x4e, 0x4c, 0x3f, 0x28, 0x5d
0x50, 0x71, 0x1d, 0x3a, 0x0d, 0x38, 0x70, 0x43, 0x36, 0x5f, 0x58, 0x37, 0x6e, 0x13, 0x30, 0x4b
0x21, 0x16, 0x20, 0x11, 0x62, 0x41, 0x53, 0x76, 0x6f, 0x10, 0x4d, 0x34, 0x29, 0x42, 0x3b, 0x56
0x66, 0x7f, 0x18, 0x07, 0x57, 0x5a, 0x19, 0x12, 0x73, 0x22, 0x2e, 0x79, 0x47, 0x02, 0x2a, 0x69
0x51, 0x5e, 0x63, 0x6a, 0x7e, 0x65, 0x7c, 0x61, 0x6c, 0x2b, 0x7d, 0x3c, 0x46, 0x15, 0x67, 0x32
```

Netlist (14 AND, 57 XOR, 0 NOT, $D = 1$):

```
xor_0 = x[0] ^ x[5]; xor_1 = x[2] ^ x[3]; xor_2 = x[0] ^ x[3];
xor_3 = x[1] ^ x[4]; xor_4 = x[1] ^ xor_0; xor_5 = x[2] ^ xor_0;
xor_6 = x[1] ^ xor_2; xor_7 = x[3] ^ x[4]; xor_8 = x[0] ^ x[6];
xor_9 = x[4] ^ xor_2; xor_10 = x[3] ^ x[5]; xor_11 = x[6] ^ xor_1;
xor_12 = x[2] ^ xor_3; xor_13 = x[5] ^ xor_1; xor_14 = x[4] ^ xor_0;
xor_15 = x[6] ^ xor_0; xor_16 = x[0] ^ x[2]; xor_17 = x[1] ^ x[6];
xor_18 = x[3] ^ xor_0; xor_19 = x[6] ^ xor_3; and_0 = x[2] & x[3];
and_1 = x[4] & x[6]; and_2 = x[0] & xor_1; and_3 = xor_7 & xor_0;
and_4 = xor_3 & xor_8; and_5 = xor_9 & xor_10; and_6 = xor_6 & xor_11;
and_7 = xor_12 & xor_13; and_8 = x[1] & xor_14; and_9 = x[3] & xor_15;
and_10 = xor_16 & xor_4; and_11 = xor_5 & xor_17; and_12 = xor_6 & xor_5;
and_13 = xor_18 & xor_19; xor_20 = and_5 ^ and_9; xor_21 = and_4 ^ and_12;
xor_22 = x[2] ^ and_0; xor_23 = and_3 ^ and_10; xor_24 = and_2 ^ xor_20;
xor_25 = and_6 ^ and_7; xor_26 = and_11 ^ xor_25; xor_27 = xor_21 ^ xor_22;
xor_28 = and_1 ^ xor_27; xor_29 = x[4] ^ xor_23; xor_30 = and_8 ^ xor_28;
xor_31 = xor_1 ^ xor_24; xor_32 = and_3 ^ xor_26; xor_33 = x[3] ^ and_11;
xor_34 = and_13 ^ xor_21; xor_35 = xor_24 ^ xor_33; xor_36 = xor_34 ^ xor_35;
xor_37 = xor_0 ^ and_0; xor_38 = and_7 ^ xor_23; xor_39 = xor_31 ^ xor_37;
xor_40 = xor_38 ^ xor_39; xor_41 = x[0] ^ and_8; xor_42 = and_12 ^ xor_31;
xor_43 = xor_32 ^ xor_41; xor_44 = xor_42 ^ xor_43; xor_45 = xor_4 ^ xor_30;
xor_46 = xor_32 ^ xor_45; xor_47 = x[6] ^ and_10; xor_48 = xor_20 ^ xor_26;
xor_49 = xor_27 ^ xor_47; xor_50 = xor_48 ^ xor_49; xor_51 = x[0] ^ and_2;
xor_52 = and_13 ^ xor_29; xor_53 = xor_51 ^ xor_52; xor_54 = x[6] ^ and_9;
xor_55 = xor_29 ^ xor_30; xor_56 = xor_54 ^ xor_55; y[0] = xor_36;
y[1] = xor_40; y[2] = xor_44; y[3] = xor_46; y[4] = xor_50; y[5] = xor_53;
y[6] = xor_56;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 127]$):

```
0x00, 0x01, 0x6d, 0x31, 0x29, 0x1f, 0x27, 0x63, 0x02, 0x0f, 0x20, 0x25, 0x08, 0x44, 0x23, 0x03
0x59, 0x53, 0x67, 0x4d, 0x0d, 0x7d, 0x51, 0x2f, 0x62, 0x66, 0x2b, 0x19, 0x39, 0x42, 0x34, 0x1c
0x52, 0x50, 0x69, 0x1a, 0x12, 0x35, 0x24, 0x0a, 0x3e, 0x5c, 0x6e, 0x79, 0x30, 0x26, 0x6a, 0x3a
0x4e, 0x22, 0x7f, 0x21, 0x5b, 0x13, 0x48, 0x4b, 0x45, 0x2d, 0x43, 0x5e, 0x7b, 0x15, 0x14, 0x3d
0x04, 0x55, 0x5d, 0x47, 0x18, 0x09, 0x7c, 0x6c, 0x1e, 0x0c, 0x16, 0x4f, 0x3c, 0x5a, 0x3b, 0x17
0x40, 0x70, 0x0e, 0x56, 0x1d, 0x05, 0x5f, 0x64, 0x4a, 0x36, 0x65, 0x32, 0x2c, 0x3f, 0x71, 0x49
0x10, 0x77, 0x54, 0x72, 0x37, 0x75, 0x60, 0x7e, 0x0b, 0x6f, 0x73, 0x07, 0x78, 0x2a, 0x4c, 0x58
0x46, 0x41, 0x33, 0x68, 0x2e, 0x28, 0x57, 0x11, 0x06, 0x6b, 0x1b, 0x38, 0x76, 0x7a, 0x74, 0x61
```

Inverse netlist (29 AND, 149 XOR, 15 NOT, $D = 2$):

```
not_0 = ~x[0]; not_1 = ~x[2]; not_2 = ~x[4]; not_3 = ~x[6];
xor_0 = not_2 ^ x[3]; xor_1 = not_3 ^ x[1]; xor_2 = not_1 ^ x[5];
xor_3 = xor_0 ^ not_1; xor_4 = not_0 ^ x[5]; xor_5 = xor_4 ^ not_3;
xor_6 = xor_2 ^ not_2; xor_7 = xor_0 ^ xor_1; xor_8 = not_1 ^ not_2;
xor_9 = not_2 ^ x[5]; xor_10 = xor_1 ^ x[3]; xor_11 = xor_3 ^ not_3;
xor_12 = xor_2 ^ not_3; xor_13 = not_1 ^ x[3]; xor_14 = not_1 ^ x[1];
xor_15 = xor_0 ^ xor_5; xor_16 = xor_3 ^ x[1]; xor_17 = not_0 ^ x[3];
xor_18 = xor_6 ^ not_0; xor_19 = xor_6 ^ x[1]; xor_20 = xor_7 ^ x[5];
xor_21 = x[1] ^ x[3]; xor_22 = xor_5 ^ x[3]; xor_23 = xor_0 ^ not_0;
xor_24 = xor_2 ^ xor_23; xor_25 = xor_1 ^ not_1; xor_26 = not_0 ^ not_1;
xor_27 = xor_1 ^ xor_3; xor_28 = xor_2 ^ x[3]; xor_29 = x[3] ^ x[5];
xor_30 = not_0 ^ x[1]; and_0 = xor_8 & xor_9; and_1 = xor_10 & not_1;
and_2 = xor_0 & xor_11; and_3 = xor_12 & xor_13; and_4 = xor_14 & xor_8;
and_5 = xor_15 & xor_7; and_6 = xor_16 & xor_17; and_7 = xor_17 & xor_7;
and_8 = xor_18 & xor_19; and_9 = xor_20 & xor_21; and_10 = xor_20 & xor_22;
and_11 = xor_24 & xor_25; and_12 = xor_26 & xor_27; and_13 = xor_28 & xor_5;
xor_31 = and_0 ^ not_0; xor_32 = and_8 ^ xor_31; xor_33 = xor_29 ^ xor_32;
xor_34 = and_1 ^ not_2; xor_35 = and_3 ^ and_4; xor_36 = not_1 ^ xor_33;
xor_37 = and_12 ^ xor_34; xor_38 = and_7 ^ not_3; xor_39 = and_2 ^ and_5;
xor_40 = xor_35 ^ xor_39; xor_41 = and_11 ^ xor_36; xor_42 = and_10 ^ xor_38;
xor_43 = xor_35 ^ xor_37; xor_44 = and_7 ^ xor_30; xor_45 = xor_34 ^ xor_44;
xor_46 = xor_40 ^ xor_45; xor_47 = and_1 ^ and_9; xor_48 = and_13 ^ xor_47;
xor_49 = xor_40 ^ xor_48; xor_50 = xor_41 ^ xor_49; xor_51 = xor_42 ^ xor_50;
not_4 = ~xor_51; xor_52 = and_6 ^ xor_41; xor_53 = xor_43 ^ xor_52;
xor_54 = xor_36 ^ x[1]; xor_55 = xor_37 ^ xor_54; xor_56 = xor_42 ^ xor_55;
not_5 = ~xor_56; xor_57 = and_2 ^ xor_1; xor_58 = and_6 ^ xor_57;
xor_59 = and_11 ^ xor_58; xor_60 = and_13 ^ xor_59; xor_61 = xor_33 ^ xor_60;
not_6 = ~xor_61; xor_62 = and_9 ^ xor_32; xor_63 = xor_43 ^ xor_62;
not_7 = ~xor_63; xor_64 = and_3 ^ xor_8; xor_65 = and_5 ^ xor_64;
xor_66 = and_6 ^ xor_65; xor_67 = and_8 ^ xor_66; xor_68 = xor_38 ^ xor_67;
not_8 = ~xor_68; xor_69 = not_6 ^ xor_46; xor_70 = not_5 ^ xor_46;
xor_71 = not_6 ^ xor_53; xor_72 = not_7 ^ not_8; xor_73 = not_4 ^ not_6;
xor_74 = not_4 ^ not_5; xor_75 = not_7 ^ xor_53; xor_76 = not_4 ^ xor_53;
xor_77 = not_6 ^ not_7; xor_78 = not_5 ^ not_7; xor_79 = xor_53 ^ xor_70;
xor_80 = not_4 ^ xor_70; xor_81 = not_7 ^ xor_69; xor_82 = xor_53 ^ xor_69;
xor_83 = not_8 ^ xor_71; xor_84 = not_5 ^ not_6; xor_85 = not_8 ^ xor_84;
xor_86 = not_8 ^ xor_70; xor_87 = xor_46 ^ xor_72; xor_88 = not_7 ^ xor_73;
xor_89 = not_8 ^ xor_53; and_14 = not_4 & not_5; and_15 = xor_69 & xor_74;
and_16 = xor_46 & xor_75; and_17 = xor_76 & xor_77; and_18 = xor_71 & xor_78;
and_19 = xor_53 & xor_72; and_20 = not_8 & xor_79; and_21 = xor_78 & xor_80;
and_22 = not_4 & xor_81; and_23 = not_5 & xor_82; and_24 = xor_81 & xor_83;
and_25 = xor_80 & xor_85; and_26 = xor_73 & xor_86; and_27 = xor_69 & xor_87;
and_28 = xor_88 & xor_89; xor_90 = and_21 ^ and_26; xor_91 = and_13 ^ not_0;
xor_92 = and_2 ^ and_22; xor_93 = and_25 ^ and_28; xor_94 = and_15 ^ and_8;
xor_95 = xor_94 ^ xor_64; xor_96 = and_1 ^ and_4; xor_97 = and_20 ^ xor_90;
xor_98 = xor_92 ^ xor_96; xor_99 = and_5 ^ and_7; xor_100 = and_9 ^ xor_91;
xor_101 = and_18 ^ xor_93; xor_102 = and_24 ^ xor_95;
xor_103 = xor_102 ^ xor_97; xor_104 = and_11 ^ and_12;
xor_105 = xor_29 ^ xor_100; xor_106 = xor_104 ^ xor_105;
xor_107 = and_12 ^ not_3; xor_108 = and_0 ^ and_10; xor_109 = xor_98 ^ xor_107;
xor_110 = and_24 ^ x[1]; xor_111 = xor_99 ^ not_2; xor_112 = and_14 ^ and_16;
xor_113 = and_17 ^ xor_112; xor_114 = xor_103 ^ xor_113;
xor_115 = xor_106 ^ xor_114; not_9 = ~xor_115; xor_116 = and_6 ^ not_3;
xor_117 = and_22 ^ xor_116; xor_118 = and_25 ^ xor_117;
xor_119 = xor_99 ^ xor_118; xor_120 = xor_103 ^ xor_119; not_10 = ~xor_120;
xor_121 = and_23 ^ and_27; xor_122 = xor_121 ^ xor_95;
xor_123 = xor_101 ^ xor_122; xor_124 = xor_106 ^ xor_123; not_11 = ~xor_124;
xor_125 = xor_8 ^ xor_90; xor_126 = xor_91 ^ xor_125;
xor_127 = xor_98 ^ xor_126; xor_128 = xor_101 ^ xor_127;
xor_129 = xor_108 ^ xor_128; not_12 = ~xor_129; xor_130 = and_13 ^ and_19;
xor_131 = xor_130 ^ xor_64; xor_132 = xor_131 ^ xor_93;
xor_133 = xor_109 ^ xor_132; xor_134 = xor_110 ^ xor_133; not_13 = ~xor_134;
xor_135 = and_27 ^ and_28; xor_136 = xor_135 ^ xor_97;
xor_137 = xor_108 ^ xor_136; xor_138 = xor_109 ^ xor_137;
xor_139 = xor_111 ^ xor_138; xor_140 = and_0 ^ and_3;
xor_141 = and_6 ^ xor_140; xor_142 = and_15 ^ xor_141;
```

```
xor_143 = and_16 ^ xor_142; xor_144 = xor_92 ^ xor_143;
xor_145 = xor_100 ^ xor_144; xor_146 = xor_101 ^ xor_145;
xor_147 = xor_110 ^ xor_146; xor_148 = xor_111 ^ xor_147; not_14 = ~xor_148;
y[0] = not_9; y[1] = not_10; y[2] = not_11; y[3] = not_12; y[4] = not_13;
y[5] = xor_139; y[6] = not_14;
```

C.23 7-bit Kasami APN x^{13} (7-bit, Category A)

- **Profile**: $\delta = 2$, $L = 16$ (NL = 56), $\deg(S) = 3$, $\mathcal{D}(S) = [3^{127}]$, FP = 2, SAC deviation 0.0625.
- **Cost**: 30 AND, 114 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 213.86 μm^2 / 268.0 GE, delay 1007.6 ps, logic depth 16; pipelined $d = 2$ DOM-indep: 148 `DFF_X1` [120 gadget + 28 sync], 1352.34 μm^2 / 1694.7 GE, stage delay 644.6 ps).
- **Remark**: The exact lookup table of x^{13} over $\mathbb{F}_2[x]/(x^7+x+1)$; 27/115/2 for x^{11}/x^{13} in [BDM26b]: three more AND gates and one fewer XOR gate.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 3$, $\mathcal{D}(S^{-1}) = [3^{127}]$; cost: 37 AND, 139 XOR, 0 NOT, AND depth $D = 2$ (Tier 3: Kasami APN inverse decomposition $y^{88} = (y^3y^8)^8$ at $D = 2$, via the Welch coset exponent $88 \equiv 11 \times 8 \pmod{127}$; unmasked NanGate 45 nm: 261.21 μm^2 / 327.3 GE, delay 1056.7 ps, logic depth 17; pipelined $d = 2$ DOM-indep: 204 `DFF_X1` [148 gadget + 56 sync], 1759.86 μm^2 / 2205.3 GE, stage delay 735.9 ps).

Lookup table (LUT[0 . . . 127]):

```
0x00, 0x01, 0x43, 0x1a, 0x65, 0x42, 0x39, 0x68, 0x3a, 0x0e, 0x26, 0x1e, 0x2f, 0x49, 0x67, 0x23
0x69, 0x64, 0x1b, 0x32, 0x5f, 0x38, 0x33, 0x5e, 0x08, 0x03, 0x74, 0x0f, 0x3f, 0x2a, 0x16, 0x5d
0x24, 0x13, 0x79, 0x7e, 0x7a, 0x34, 0x7d, 0x2d, 0x53, 0x75, 0x6c, 0x2e, 0x3e, 0x15, 0x10, 0x71
0x14, 0x2b, 0x46, 0x35, 0x51, 0x5b, 0x58, 0x30, 0x20, 0x3d, 0x41, 0x44, 0x27, 0x7b, 0x56, 0x3c
0x5a, 0x54, 0x6e, 0x55, 0x0c, 0x52, 0x40, 0x05, 0x4a, 0x17, 0x72, 0x4e, 0x06, 0x7f, 0x0d, 0x3b
0x4d, 0x21, 0x12, 0x37, 0x6d, 0x1d, 0x4b, 0x5c, 0x63, 0x6f, 0x61, 0x70, 0x28, 0x4c, 0x18, 0x4f
0x22, 0x76, 0x02, 0x0b, 0x2c, 0x77, 0x31, 0x19, 0x48, 0x6b, 0x59, 0x73, 0x1f, 0x47, 0x78, 0x07
0x50, 0x62, 0x25, 0x36, 0x60, 0x11, 0x29, 0x57, 0x1c, 0x6a, 0x09, 0x0a, 0x04, 0x45, 0x66, 0x7c
```

Netlist (30 AND, 114 XOR, 0 NOT, $D = 2$):

```
a_in = in_val[6:0]; split_2 = a_in[0:0]; split_3 = a_in[1:1];
split_4 = a_in[2:2]; split_5 = a_in[3:3]; split_6 = a_in[4:4];
split_7 = a_in[5:5]; split_8 = a_in[6:6]; xor_16 = split_3 ^ split_4;
xor_9 = split_2 ^ split_5; xor_19 = split_3 ^ split_5;
and_30 = split_3 & split_5; xor_10 = split_2 ^ split_6;
xor_12 = split_4 ^ split_6; xor_27 = split_5 ^ split_6;
paar_shared_xor_0_split_3_split_6 = split_3 ^ split_6;
xor_14 = split_4 ^ split_7; xor_17 = split_6 ^ split_7;
xor_21 = split_5 ^ split_7; xor_13 = split_7 ^ split_8;
xor_26 = split_4 ^ split_8; xor_20 = split_4 ^ xor_9; xor_22 = split_3 ^ xor_9;
xor_24 = split_8 ^ xor_9; xor_15 = split_7 ^ xor_10; xor_18 = split_4 ^ xor_10;
and_36 = xor_19 & xor_10; xor_29 = split_8 ^ xor_12; xor_28 = split_8 ^ xor_27;
and_31 = split_2 & xor_14; and_34 = xor_16 & xor_17; and_38 = xor_12 & xor_21;
xor_23 = split_2 ^ xor_13; and_32 = split_4 & xor_13; and_37 = xor_20 & split_8;
and_39 = xor_22 & xor_21; and_33 = split_3 & xor_15; and_35 = split_5 & xor_18;
and_44 = xor_15 & xor_29; and_43 = xor_22 & xor_28;
xor_25 = split_7 ^ paar_shared_xor_0_split_3_split_6;
and_41 = paar_shared_xor_0_split_3_split_6 & xor_24;
xor_67 = paar_shared_xor_0_split_3_split_6 ^ and_36; xor_54 = and_30 ^ and_34;
xor_50 = split_4 ^ and_38; and_40 = xor_10 & xor_23; xor_47 = split_2 ^ and_37;
xor_59 = and_37 ^ and_38; xor_70 = and_31 ^ and_37; xor_58 = and_32 ^ and_33;
xor_51 = and_32 ^ and_35; xor_45 = and_31 ^ and_44; and_42 = xor_25 & xor_26;
xor_48 = and_39 ^ and_41; xor_74 = and_33 ^ and_40; xor_78 = and_40 ^ and_43;
xor_52 = split_6 ^ xor_47; xor_61 = xor_58 ^ xor_59; xor_64 = split_3 ^ xor_51;
xor_72 = xor_51 ^ xor_70; xor_46 = split_7 ^ xor_45; xor_71 = and_42 ^ xor_50;
xor_75 = and_42 ^ xor_47; xor_55 = xor_48 ^ xor_54; xor_60 = and_43 ^ xor_48;
xor_68 = xor_45 ^ xor_48; xor_79 = xor_50 ^ xor_78; xor_49 = and_36 ^ xor_46;
xor_73 = xor_71 ^ xor_72; xor_76 = xor_74 ^ xor_75; xor_56 = xor_52 ^ xor_55;
xor_62 = xor_46 ^ xor_60; xor_69 = xor_67 ^ xor_68; xor_80 = xor_52 ^ xor_79;
xor_53 = split_8 ^ xor_49; xor_65 = xor_64 ^ xor_49; xor_63 = xor_61 ^ xor_62;
xor_99 = xor_69 ^ xor_73; xor_57 = xor_56 ^ xor_53; xor_77 = xor_76 ^ xor_53;
xor_84 = xor_65 ^ xor_73; xor_98 = xor_65 ^ xor_80; xor_83 = xor_63 ^ xor_73;
xor_88 = xor_63 ^ xor_65; xor_91 = xor_63 ^ xor_69; and_102 = xor_63 & xor_69;
xor_121 = xor_63 ^ xor_80; xor_100 = xor_80 ^ xor_99; xor_81 = xor_57 ^ xor_69;
```

```
xor_82 = xor_57 ^ xor_73; xor_85 = xor_77 ^ xor_80; xor_86 = xor_65 ^ xor_77;
xor_89 = xor_73 ^ xor_77; xor_93 = xor_69 ^ xor_77; xor_101 = xor_80 ^ xor_84;
xor_97 = xor_77 ^ xor_83; xor_92 = xor_65 ^ xor_81; xor_94 = xor_63 ^ xor_81;
xor_96 = xor_80 ^ xor_81; xor_87 = xor_77 ^ xor_82; xor_90 = xor_65 ^ xor_82;
and_108 = xor_91 & xor_82; xor_95 = xor_57 ^ xor_85; and_104 = xor_65 & xor_85;
and_103 = xor_57 & xor_86; and_106 = xor_88 & xor_89; and_110 = xor_84 & xor_93;
and_114 = xor_97 & xor_98; and_109 = xor_92 & xor_80; and_111 = xor_94 & xor_93;
and_115 = xor_94 & xor_100; and_113 = xor_83 & xor_96; and_105 = xor_63 & xor_87;
and_116 = xor_87 & xor_101; and_107 = xor_69 & xor_90; and_112 = xor_82 & xor_95;
xor_139 = xor_121 ^ and_104; xor_131 = and_102 ^ and_103;
xor_142 = xor_63 ^ and_103; xor_125 = xor_65 ^ and_110;
xor_119 = xor_57 ^ and_109; xor_117 = and_111 ^ and_113;
xor_122 = and_105 ^ and_114; xor_147 = and_105 ^ and_109;
xor_123 = and_108 ^ and_116; xor_126 = and_110 ^ and_116;
xor_150 = and_115 ^ xor_125; xor_124 = and_112 ^ xor_119;
xor_132 = and_106 ^ xor_119; xor_118 = xor_77 ^ xor_117;
xor_128 = and_115 ^ xor_117; xor_129 = xor_121 ^ xor_122;
xor_136 = and_107 ^ xor_122; xor_127 = xor_80 ^ xor_123;
xor_143 = xor_123 ^ xor_125; xor_140 = xor_126 ^ xor_139;
xor_120 = xor_73 ^ xor_118; xor_144 = xor_142 ^ xor_128;
xor_151 = xor_129 ^ xor_118; xor_137 = xor_126 ^ xor_136;
xor_133 = xor_131 ^ xor_127; xor_148 = xor_147 ^ xor_127;
xor_145 = xor_124 ^ xor_143; xor_130 = xor_120 ^ xor_124;
xor_134 = xor_132 ^ xor_120; xor_141 = xor_120 ^ xor_140;
xor_152 = xor_150 ^ xor_151; xor_149 = xor_128 ^ xor_148;
xor_146 = xor_144 ^ xor_145; xor_138 = xor_137 ^ xor_130;
xor_153 = xor_129 ^ xor_130; xor_135 = xor_133 ^ xor_134;
concat_154 = { xor_153, xor_152, xor_149, xor_146, xor_141, xor_138, xor_135 };
out_val = concat_154;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 127]$):

```
0x00, 0x01, 0x62, 0x19, 0x7c, 0x47, 0x4c, 0x6f, 0x18, 0x7a, 0x7b, 0x63, 0x44, 0x4e, 0x09, 0x1b
0x2e, 0x75, 0x52, 0x21, 0x30, 0x2d, 0x1e, 0x49, 0x5e, 0x67, 0x03, 0x12, 0x78, 0x55, 0x0b, 0x6c
0x38, 0x51, 0x60, 0x0f, 0x20, 0x72, 0x0a, 0x3c, 0x5c, 0x76, 0x1d, 0x31, 0x64, 0x27, 0x2b, 0x0c
0x37, 0x66, 0x13, 0x16, 0x25, 0x33, 0x73, 0x53, 0x15, 0x06, 0x08, 0x4f, 0x3f, 0x39, 0x2c, 0x1c
0x46, 0x3a, 0x05, 0x02, 0x3b, 0x7d, 0x32, 0x6d, 0x68, 0x0d, 0x48, 0x56, 0x5d, 0x50, 0x4b, 0x5f
0x70, 0x34, 0x45, 0x28, 0x41, 0x43, 0x3e, 0x77, 0x36, 0x6a, 0x40, 0x35, 0x57, 0x1f, 0x17, 0x14
0x74, 0x5a, 0x71, 0x58, 0x11, 0x04, 0x7e, 0x0e, 0x07, 0x10, 0x79, 0x69, 0x2a, 0x54, 0x42, 0x59
0x5b, 0x2f, 0x4a, 0x6b, 0x1a, 0x29, 0x61, 0x65, 0x6e, 0x22, 0x24, 0x3d, 0x7f, 0x26, 0x23, 0x4d
```

Inverse netlist (37 AND, 139 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[1] ^ x[4]; xor_1 = x[0] ^ x[5]; xor_2 = x[2] ^ x[6];
xor_3 = x[3] ^ x[5]; xor_4 = xor_2 ^ x[3]; xor_5 = xor_0 ^ x[6];
xor_6 = xor_0 ^ x[2]; xor_7 = x[3] ^ x[4]; xor_8 = xor_0 ^ xor_2;
xor_9 = x[2] ^ x[4]; xor_10 = xor_2 ^ x[4]; xor_11 = xor_1 ^ x[4];
xor_12 = x[0] ^ x[1]; xor_13 = xor_1 ^ x[1]; xor_14 = x[0] ^ x[6];
xor_15 = x[2] ^ x[3]; xor_16 = xor_7 ^ x[0]; xor_17 = xor_12 ^ x[3];
xor_18 = xor_3 ^ x[2]; xor_19 = xor_1 ^ x[6]; xor_20 = x[0] ^ x[2];
xor_21 = xor_1 ^ x[2]; xor_22 = x[1] ^ x[6]; xor_23 = xor_1 ^ x[3];
xor_24 = x[1] ^ x[5]; xor_25 = xor_5 ^ x[0]; xor_26 = xor_8 ^ x[5];
xor_27 = x[3] ^ x[6]; xor_28 = xor_3 ^ xor_6; xor_29 = xor_1 ^ xor_8;
xor_30 = xor_0 ^ x[3]; xor_31 = xor_1 ^ xor_4; xor_32 = xor_10 ^ xor_3;
xor_33 = xor_3 ^ x[6]; xor_34 = xor_11 ^ xor_4; xor_35 = xor_0 ^ xor_3;
xor_36 = xor_12 ^ xor_2; xor_37 = xor_0 ^ xor_4; xor_38 = xor_5 ^ x[5];
xor_39 = xor_14 ^ x[4]; and_0 = x[2] & x[3]; and_1 = x[4] & x[6];
and_2 = xor_15 & x[0]; and_3 = xor_1 & xor_7; and_4 = xor_0 & xor_14;
and_5 = xor_16 & xor_3; and_6 = xor_17 & xor_4; and_7 = xor_18 & xor_6;
and_8 = xor_11 & x[1]; and_9 = xor_19 & x[3]; and_10 = xor_13 & xor_20;
and_11 = xor_21 & xor_22; and_12 = xor_17 & xor_21; and_13 = xor_23 & xor_5;
xor_40 = and_7 ^ x[2]; xor_41 = and_5 ^ and_9; xor_42 = and_11 ^ and_12;
xor_43 = and_3 ^ x[0]; xor_44 = and_2 ^ x[3]; xor_45 = xor_41 ^ xor_44;
xor_46 = xor_40 ^ xor_43; xor_47 = and_4 ^ xor_42; xor_48 = and_0 ^ and_10;
xor_49 = xor_45 ^ xor_46; xor_50 = and_6 ^ and_8; xor_51 = xor_48 ^ x[6];
xor_52 = and_13 ^ xor_45; xor_53 = xor_47 ^ xor_52; xor_54 = xor_48 ^ x[5];
xor_55 = xor_49 ^ xor_54; xor_56 = xor_42 ^ xor_49; xor_57 = xor_50 ^ xor_56;
xor_58 = and_0 ^ xor_24; xor_59 = and_1 ^ xor_58; xor_60 = xor_46 ^ xor_59;
xor_61 = xor_47 ^ xor_60; xor_62 = xor_50 ^ xor_61; xor_63 = and_6 ^ xor_40;
xor_64 = xor_41 ^ xor_63; xor_65 = xor_47 ^ xor_64; xor_66 = xor_51 ^ xor_65;
xor_67 = and_2 ^ x[4]; xor_68 = and_10 ^ xor_67; xor_69 = and_13 ^ xor_68;
xor_70 = xor_43 ^ xor_69; xor_71 = and_1 ^ xor_9; xor_72 = and_3 ^ xor_71;
xor_73 = and_4 ^ xor_72; xor_74 = and_8 ^ xor_73; xor_75 = and_9 ^ xor_74;
xor_76 = and_12 ^ xor_75; xor_77 = xor_51 ^ xor_76; and_14 = xor_53 & x[0];
and_15 = xor_5 & xor_55; xor_78 = xor_53 ^ xor_55; and_16 = xor_25 & xor_78;
and_17 = xor_26 & xor_57; and_18 = xor_27 & xor_62; xor_79 = xor_57 ^ xor_62;
and_19 = xor_28 & xor_79; xor_80 = xor_53 ^ xor_57; xor_81 = xor_55 ^ xor_62;
and_20 = xor_29 & xor_80; and_21 = xor_30 & xor_81; xor_82 = xor_80 ^ xor_81;
and_22 = xor_31 & xor_82; and_23 = xor_32 & xor_66; and_24 = xor_33 & xor_70;
```

```
and_25 = xor_3 & xor_77; xor_83 = xor_66 ^ xor_70; and_26 = xor_9 & xor_83;
xor_84 = xor_66 ^ xor_77; and_27 = xor_10 & xor_84; xor_85 = xor_70 ^ xor_77;
and_28 = xor_85 & x[6]; xor_86 = xor_53 ^ xor_66; xor_87 = xor_55 ^ xor_70;
xor_88 = xor_57 ^ xor_77; and_29 = xor_34 & xor_86; and_30 = xor_35 & xor_87;
xor_89 = xor_86 ^ xor_87; and_31 = xor_36 & xor_89; and_32 = xor_37 & xor_88;
xor_90 = xor_88 ^ xor_62; and_33 = xor_6 & xor_90; xor_91 = xor_86 ^ xor_88;
xor_92 = xor_87 ^ xor_62; and_34 = xor_13 & xor_91; and_35 = xor_38 & xor_92;
xor_93 = xor_91 ^ xor_92; and_36 = xor_39 & xor_93; xor_94 = and_21 ^ and_22;
xor_95 = and_29 ^ and_30; xor_96 = and_31 ^ xor_94; xor_97 = and_24 ^ and_25;
xor_98 = and_19 ^ and_28; xor_99 = and_33 ^ xor_95; xor_100 = and_32 ^ and_36;
xor_101 = and_16 ^ xor_97; xor_102 = and_20 ^ and_35;
xor_103 = and_15 ^ xor_96; xor_104 = and_17 ^ xor_99;
xor_105 = xor_100 ^ xor_103; xor_106 = xor_98 ^ xor_101;
xor_107 = xor_106 ^ xor_102; xor_108 = and_34 ^ xor_104;
xor_109 = and_14 ^ and_34; xor_110 = and_27 ^ xor_96;
xor_111 = xor_105 ^ xor_107; xor_112 = xor_108 ^ xor_111;
xor_113 = and_19 ^ xor_99; xor_114 = xor_105 ^ xor_113;
xor_115 = and_18 ^ and_22; xor_116 = and_23 ^ xor_115;
xor_117 = and_30 ^ xor_116; xor_118 = xor_117 ^ xor_100;
xor_119 = xor_107 ^ xor_118; xor_120 = and_23 ^ and_26;
xor_121 = xor_94 ^ xor_120; xor_122 = xor_121 ^ xor_101;
xor_123 = xor_108 ^ xor_122; xor_124 = and_26 ^ xor_95;
xor_125 = xor_124 ^ xor_102; xor_126 = xor_109 ^ xor_125;
xor_127 = xor_110 ^ xor_126; xor_128 = and_18 ^ and_33;
xor_129 = xor_97 ^ xor_128; xor_130 = xor_129 ^ xor_98;
xor_131 = xor_130 ^ xor_103; xor_132 = xor_109 ^ xor_131;
xor_133 = and_20 ^ and_23; xor_134 = and_25 ^ xor_133;
xor_135 = and_32 ^ xor_134; xor_136 = xor_98 ^ xor_135;
xor_137 = xor_104 ^ xor_136; xor_138 = xor_110 ^ xor_137; y[0] = xor_112;
y[1] = xor_114; y[2] = xor_119; y[3] = xor_123; y[4] = xor_127; y[5] = xor_132;
y[6] = xor_138;
```

C.24 7-bit inversion class (7-bit, Category A)

- **Profile**: $\delta = 2$, $L = 20$ (NL = 54), $\deg(S) = 6$, $\mathcal{D}(S) = [6^{127}]$, FP = 0, SAC deviation 0.0938.
- **Cost**: 42 AND, 182 XOR, 6 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 338.35 μm^2 / 424.0 GE, delay 2043.5 ps, logic depth 33; pipelined $d = 2$ DOM-indep: 212 `DFF_X1` [168 gadget + 44 sync], 1989.68 μm^2 / 2493.3 GE, stage delay 718.5 ps).
- **Remark**: Certified affine equivalent to the inversion $x^{63} \sim_{\text{aff}} x^{126}$ (with $S_{\text{ref}}(u) = u^{63} \bmod (x^7 + x + 1)$, $A =$ [`0x03`, `0x62`, `0x39`, `0x6f`, `0x0c`, `0x10`, `0x15`], $a =$ `0x01`, $B =$ [`0x2e`, `0x5f`, `0x02`, `0x46`, `0x38`, `0x3f`, `0x7f`], $b =$ `0x16`); degree 6 at depth 3.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 6$, $\mathcal{D}(S^{-1}) = [6^{127}]$; cost: 42 AND, 229 XOR, 6 NOT, AND depth $D = 3$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 413.36 μm^2 / 518.0 GE, delay 2118.5 ps, logic depth 33; pipelined $d = 2$ DOM-indep: 306 `DFF_X1` [168 gadget + 138 sync], 2564.77 μm^2 / 3214.0 GE, stage delay 716.5 ps).

Lookup table (LUT[0 . . . 127]):

```
0x38, 0x7e, 0x6d, 0x5b, 0x17, 0x43, 0x09, 0x1d, 0x3f, 0x23, 0x57, 0x7f, 0x1a, 0x19, 0x7c, 0x00
0x7b, 0x60, 0x22, 0x37, 0x64, 0x28, 0x2d, 0x25, 0x77, 0x49, 0x16, 0x6e, 0x01, 0x54, 0x4f, 0x0e
0x30, 0x05, 0x42, 0x5d, 0x36, 0x69, 0x62, 0x59, 0x5f, 0x2f, 0x32, 0x6c, 0x58, 0x02, 0x2c, 0x10
0x46, 0x0c, 0x50, 0x08, 0x21, 0x61, 0x1c, 0x70, 0x11, 0x2b, 0x78, 0x6b, 0x4b, 0x20, 0x7d, 0x67
0x24, 0x0f, 0x18, 0x44, 0x1f, 0x48, 0x5a, 0x76, 0x63, 0x1e, 0x66, 0x51, 0x0b, 0x71, 0x27, 0x07
0x04, 0x13, 0x35, 0x14, 0x26, 0x2a, 0x0d, 0x40, 0x3c, 0x03, 0x4c, 0x5e, 0x79, 0x68, 0x1b, 0x29
0x47, 0x15, 0x4e, 0x53, 0x39, 0x3b, 0x55, 0x3a, 0x41, 0x6f, 0x73, 0x65, 0x31, 0x5c, 0x6a, 0x72
0x45, 0x0a, 0x7a, 0x33, 0x34, 0x52, 0x4d, 0x4a, 0x3d, 0x12, 0x75, 0x06, 0x74, 0x56, 0x3e, 0x2e
```

Netlist (42 AND, 182 XOR, 6 NOT, $D = 3$):

```
a_in = in_val[6:0]; split_1 = a_in[0:0]; split_2 = a_in[1:1];
split_3 = a_in[2:2]; split_4 = a_in[3:3]; split_5 = a_in[4:4];
split_6 = a_in[5:5]; split_7 = a_in[6:6];
sync_buf_split_1_to_xor_61_stg0 = split_1;
sync_buf_split_3_to_xor_65_stg0 = split_3; xor_8 = split_1 ^ split_4;
xor_13 = split_5 ^ split_6; sync_buf_split_7_to_xor_66_stg0 = split_7;
sync_buf_split_7_to_xor_54_stg0 = split_7; xor_9 = split_6 ^ xor_8;
not_15 = ~xor_8; not_14 = ~xor_13; xor_10 = split_2 ^ xor_9;
```

```
xor_11 = split_7 ^ xor_9; xor_19 = split_1 ^ not_15; xor_28 = not_15 ^ split_7;
sync_buf_not_15_to_xor_75_stg0 = not_15; xor_17 = not_14 ^ split_7;
sync_buf_not_14_to_xor_62_stg0 = not_14; sync_buf_not_14_to_xor_84_stg0 = not_14;
xor_18 = split_1 ^ xor_10; xor_20 = xor_10 ^ not_14; xor_22 = xor_10 ^ not_15;
sync_buf_xor_10_to_xor_81_stg0 = xor_10; not_12 = ~xor_11;
xor_24 = split_3 ^ xor_19; xor_30 = xor_10 ^ xor_17; xor_32 = not_15 ^ xor_17;
xor_35 = split_1 ^ xor_17; xor_37 = split_3 ^ xor_17; xor_36 = split_7 ^ xor_18;
xor_27 = not_15 ^ xor_20; xor_16 = split_1 ^ not_12; xor_21 = split_3 ^ not_12;
xor_23 = not_12 ^ split_7; xor_33 = not_12 ^ not_14; and_38 = xor_10 & not_12;
sync_buf_not_12_to_xor_56_stg0 = not_12; and_43 = xor_18 & xor_24;
and_51 = xor_36 & xor_37; and_46 = xor_27 & xor_28; xor_25 = split_7 ^ xor_16;
xor_26 = not_15 ^ xor_16; xor_29 = split_3 ^ xor_16; and_41 = xor_16 & xor_22;
xor_31 = xor_10 ^ xor_21; and_39 = xor_21 & not_14; and_40 = xor_18 & xor_21;
and_42 = xor_20 & xor_23; xor_34 = not_15 ^ xor_33;
xor_61 = sync_buf_split_1_to_xor_61_stg0 ^ and_51; and_44 = xor_19 & xor_25;
and_45 = xor_26 & xor_23; and_47 = xor_29 & xor_30; and_48 = xor_31 & xor_25;
and_49 = xor_31 & xor_32; xor_57 = and_40 ^ and_51;
xor_62 = sync_buf_not_14_to_xor_62_stg0 ^ and_42;
xor_66 = sync_buf_split_7_to_xor_66_stg0 ^ and_42; and_50 = xor_34 & xor_35;
xor_54 = sync_buf_split_7_to_xor_54_stg0 ^ and_44; xor_52 = and_46 ^ and_47;
xor_58 = and_43 ^ and_48; xor_78 = and_48 ^ xor_61; xor_55 = and_38 ^ and_49;
xor_84 = sync_buf_not_14_to_xor_84_stg0 ^ and_49;
xor_75 = sync_buf_not_15_to_xor_75_stg0 ^ and_50; xor_59 = and_39 ^ xor_54;
xor_81 = sync_buf_xor_10_to_xor_81_stg0 ^ xor_54; xor_53 = and_41 ^ xor_52;
xor_69 = xor_52 ^ xor_61; xor_63 = xor_57 ^ xor_58; xor_72 = and_50 ^ xor_55;
xor_85 = xor_57 ^ xor_84; xor_76 = xor_62 ^ xor_75; xor_64 = and_45 ^ xor_59;
xor_79 = xor_59 ^ xor_78; xor_56 = sync_buf_not_12_to_xor_56_stg0 ^ xor_53;
xor_70 = xor_62 ^ xor_69; xor_67 = xor_66 ^ xor_63; xor_82 = xor_63 ^ xor_81;
xor_73 = xor_58 ^ xor_72; xor_86 = xor_53 ^ xor_85; xor_60 = xor_55 ^ xor_56;
xor_71 = xor_64 ^ xor_70; xor_83 = xor_56 ^ xor_82; xor_74 = xor_64 ^ xor_73;
sync_buf_xor_86_to_xor_137_stg0 = xor_86;
sync_buf_xor_86_to_xor_125_stg0 = xor_86;
xor_65 = sync_buf_split_3_to_xor_65_stg0 ^ xor_60; xor_68 = xor_67 ^ xor_60;
sync_buf_xor_71_to_xor_152_stg0 = xor_71; xor_93 = xor_71 ^ xor_83;
xor_99 = xor_83 ^ xor_86; sync_buf_xor_83_to_xor_146_stg0 = xor_83;
sync_buf_xor_74_to_xor_136_stg0 = xor_74; xor_77 = xor_76 ^ xor_65;
xor_80 = xor_79 ^ xor_65; xor_89 = xor_68 ^ xor_71; xor_90 = xor_68 ^ xor_83;
sync_buf_xor_68_to_xor_132_stg0 = xor_68; xor_87 = xor_68 ^ xor_77;
xor_92 = xor_74 ^ xor_77; xor_94 = xor_77 ^ xor_86; and_109 = xor_71 & xor_77;
sync_buf_xor_77_to_xor_127_stg0 = xor_77; xor_88 = xor_80 ^ xor_86;
xor_91 = xor_71 ^ xor_80; xor_104 = xor_77 ^ xor_80;
sync_buf_xor_80_to_xor_133_stg0 = xor_80;
sync_buf_xor_80_to_xor_155_stg0 = xor_80; xor_107 = xor_86 ^ xor_89;
xor_95 = xor_74 ^ xor_90; xor_96 = xor_86 ^ xor_87; xor_97 = xor_83 ^ xor_87;
xor_100 = xor_74 ^ xor_87; and_112 = xor_87 & xor_93; xor_102 = xor_71 ^ xor_92;
and_110 = xor_92 & xor_80; and_111 = xor_89 & xor_92; xor_101 = xor_71 ^ xor_88;
xor_103 = xor_83 ^ xor_88; xor_106 = xor_68 ^ xor_88; xor_108 = xor_74 ^ xor_88;
xor_98 = xor_83 ^ xor_91; and_113 = xor_91 & xor_94; xor_105 = xor_83 ^ xor_104;
and_114 = xor_89 & xor_95; and_115 = xor_90 & xor_96; and_116 = xor_97 & xor_94;
and_119 = xor_102 & xor_96; and_118 = xor_100 & xor_101;
and_120 = xor_102 & xor_103; and_122 = xor_107 & xor_108;
and_117 = xor_98 & xor_99; xor_133 = sync_buf_xor_80_to_xor_133_stg0 ^ and_113;
xor_137 = sync_buf_xor_86_to_xor_137_stg0 ^ and_113; and_121 = xor_105 & xor_106;
xor_125 = sync_buf_xor_86_to_xor_125_stg0 ^ and_115; xor_129 = and_114 ^ and_119;
xor_126 = and_109 ^ and_120; xor_155 = sync_buf_xor_80_to_xor_155_stg0 ^ and_120;
xor_128 = and_111 ^ and_122; xor_132 = sync_buf_xor_68_to_xor_132_stg0 ^ and_122;
xor_123 = and_117 ^ and_118; xor_146 = sync_buf_xor_83_to_xor_146_stg0 ^ and_121;
xor_130 = and_110 ^ xor_125; xor_152 = sync_buf_xor_71_to_xor_152_stg0 ^ xor_125;
xor_143 = and_121 ^ xor_126; xor_134 = xor_128 ^ xor_129;
xor_156 = xor_128 ^ xor_155; xor_149 = and_119 ^ xor_132;
xor_124 = and_112 ^ xor_123; xor_140 = xor_123 ^ xor_132;
xor_147 = xor_133 ^ xor_146; xor_135 = and_116 ^ xor_130;
xor_144 = xor_129 ^ xor_143; xor_138 = xor_137 ^ xor_134;
xor_153 = xor_134 ^ xor_152; xor_150 = xor_130 ^ xor_149;
xor_127 = sync_buf_xor_77_to_xor_127_stg0 ^ xor_124; xor_157 = xor_124 ^ xor_156;
xor_141 = xor_133 ^ xor_140; xor_145 = xor_135 ^ xor_144;
xor_131 = xor_126 ^ xor_127; xor_154 = xor_127 ^ xor_153;
sync_buf_xor_157_to_xor_209_stg0 = xor_157;
sync_buf_xor_157_to_xor_196_stg0 = xor_157; xor_142 = xor_135 ^ xor_141;
sync_buf_xor_145_to_xor_223_stg0 = xor_145;
xor_136 = sync_buf_xor_74_to_xor_136_stg0 ^ xor_131; xor_139 = xor_138 ^ xor_131;
xor_170 = xor_154 ^ xor_157; sync_buf_xor_154_to_xor_203_stg0 = xor_154;
xor_164 = xor_142 ^ xor_154; sync_buf_xor_142_to_xor_204_stg0 = xor_142;
xor_148 = xor_147 ^ xor_136; xor_151 = xor_150 ^ xor_136;
xor_160 = xor_139 ^ xor_142; xor_161 = xor_139 ^ xor_154;
sync_buf_xor_139_to_xor_201_stg0 = xor_139; xor_158 = xor_139 ^ xor_148;
xor_163 = xor_145 ^ xor_148; xor_165 = xor_148 ^ xor_157;
and_180 = xor_142 & xor_148; sync_buf_xor_148_to_xor_199_stg0 = xor_148;
```

```
xor_159 = xor_151 ^ xor_157; xor_162 = xor_142 ^ xor_151;
paar_shared_xor_0_xor_148_xor_151 = xor_148 ^ xor_151;
sync_buf_xor_151_to_xor_195_stg0 = xor_151; xor_178 = xor_157 ^ xor_160;
xor_166 = xor_145 ^ xor_161; xor_167 = xor_157 ^ xor_158;
xor_168 = xor_154 ^ xor_158; xor_171 = xor_145 ^ xor_158;
and_183 = xor_158 & xor_164; xor_173 = xor_142 ^ xor_163;
and_181 = xor_163 & xor_151; and_182 = xor_160 & xor_163;
xor_172 = xor_142 ^ xor_159; xor_174 = xor_154 ^ xor_159;
xor_177 = xor_139 ^ xor_159; xor_179 = xor_145 ^ xor_159;
xor_169 = xor_154 ^ xor_162; and_184 = xor_162 & xor_165;
and_185 = xor_160 & xor_166; and_186 = xor_161 & xor_167;
and_187 = xor_168 & xor_165;
xor_199 = sync_buf_xor_148_to_xor_199_stg0 ^ and_183;
and_190 = xor_173 & xor_167; and_189 = xor_171 & xor_172;
and_191 = xor_173 & xor_174; and_193 = xor_178 & xor_179;
and_188 = xor_169 & xor_170;
xor_195 = sync_buf_xor_151_to_xor_195_stg0 ^ and_184;
xor_209 = sync_buf_xor_157_to_xor_209_stg0 ^ and_184;
xor_176 = xor_154 ^ paar_shared_xor_0_xor_148_xor_151;
sync_buf_xor_175_to_xor_218_stg0 = paar_shared_xor_0_xor_148_xor_151;
xor_196 = sync_buf_xor_157_to_xor_196_stg0 ^ and_186;
xor_206 = and_182 ^ xor_199; xor_197 = and_185 ^ and_190;
xor_200 = and_180 ^ and_191; xor_214 = and_190 ^ and_193;
xor_194 = and_188 ^ and_189;
xor_201 = sync_buf_xor_139_to_xor_201_stg0 ^ xor_195;
xor_224 = xor_195 ^ xor_199; and_192 = xor_176 & xor_177;
xor_218 = sync_buf_xor_175_to_xor_218_stg0 ^ and_191;
xor_204 = sync_buf_xor_142_to_xor_204_stg0 ^ xor_196;
xor_205 = and_181 ^ xor_196; xor_210 = xor_209 ^ xor_206;
xor_198 = and_193 ^ xor_194;
xor_223 = sync_buf_xor_145_to_xor_223_stg0 ^ xor_194;
xor_203 = sync_buf_xor_154_to_xor_203_stg0 ^ and_192;
xor_212 = and_192 ^ xor_201; xor_219 = xor_197 ^ xor_204;
xor_228 = xor_206 ^ xor_204; xor_207 = xor_201 ^ xor_205;
xor_202 = xor_197 ^ xor_198; xor_221 = and_187 ^ xor_198;
xor_226 = xor_223 ^ xor_224; xor_215 = xor_203 ^ xor_214;
xor_225 = xor_200 ^ xor_203; xor_220 = xor_218 ^ xor_219;
xor_208 = xor_200 ^ xor_202; xor_229 = xor_202 ^ xor_228;
xor_222 = xor_207 ^ xor_221; xor_216 = xor_215 ^ xor_207;
xor_227 = xor_225 ^ xor_226; xor_211 = xor_210 ^ xor_208;
xor_213 = xor_212 ^ xor_208; sync_buf_xor_229_to_concat_240_stg0 = xor_229;
not_231 = ~xor_216; xor_233 = xor_216 ^ xor_222; xor_232 = xor_220 ^ xor_227;
xor_236 = xor_211 ^ xor_227; sync_buf_xor_211_to_xor_239_stg0 = xor_211;
not_230 = ~xor_213; sync_buf_not_231_to_concat_240_stg0 = not_231;
xor_234 = xor_227 ^ xor_233; sync_buf_xor_232_to_concat_240_stg0 = xor_232;
sync_buf_xor_236_to_concat_240_stg0 = xor_236; xor_238 = not_230 ^ not_231;
sync_buf_not_230_to_concat_240_stg0 = not_230; not_235 = ~xor_234;
xor_239 = sync_buf_xor_211_to_xor_239_stg0 ^ xor_238;
sync_buf_not_235_to_concat_240_stg0 = not_235;
concat_240 = { sync_buf_xor_229_to_concat_240_stg0,
    sync_buf_xor_236_to_concat_240_stg0, sync_buf_not_235_to_concat_240_stg0,
    sync_buf_xor_232_to_concat_240_stg0, sync_buf_not_231_to_concat_240_stg0,
    sync_buf_not_230_to_concat_240_stg0, xor_239 };
out_val = concat_240;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 127]$):

```
0x0f, 0x1c, 0x2d, 0x59, 0x50, 0x21, 0x7b, 0x4f, 0x33, 0x06, 0x71, 0x4c, 0x31, 0x56, 0x1f, 0x41
0x2f, 0x38, 0x79, 0x51, 0x53, 0x61, 0x1a, 0x04, 0x42, 0x0d, 0x0c, 0x5e, 0x36, 0x07, 0x49, 0x44
0x3d, 0x34, 0x12, 0x09, 0x40, 0x17, 0x54, 0x4e, 0x15, 0x5f, 0x55, 0x39, 0x2e, 0x16, 0x7f, 0x29
0x20, 0x6c, 0x2a, 0x73, 0x74, 0x52, 0x24, 0x13, 0x00, 0x64, 0x67, 0x65, 0x58, 0x78, 0x7e, 0x08
0x57, 0x68, 0x22, 0x05, 0x43, 0x70, 0x30, 0x60, 0x45, 0x19, 0x77, 0x3c, 0x5a, 0x76, 0x62, 0x1e
0x32, 0x4b, 0x75, 0x63, 0x1d, 0x66, 0x7d, 0x0a, 0x2c, 0x27, 0x46, 0x03, 0x6d, 0x23, 0x5b, 0x28
0x11, 0x35, 0x26, 0x48, 0x14, 0x6b, 0x4a, 0x3f, 0x5d, 0x25, 0x6e, 0x3b, 0x2b, 0x02, 0x1b, 0x69
0x37, 0x4d, 0x6f, 0x6a, 0x7c, 0x7a, 0x47, 0x18, 0x3a, 0x5c, 0x72, 0x10, 0x0e, 0x3e, 0x01, 0x0b
```

Inverse netlist (42 AND, 229 XOR, 6 NOT, $D = 3$):

```
not_0 = ~x[1]; not_1 = ~x[2]; not_2 = ~x[4]; xor_0 = not_0 ^ x[5];
xor_1 = not_2 ^ x[6]; xor_2 = not_1 ^ x[0]; xor_3 = xor_0 ^ x[3];
xor_4 = xor_1 ^ x[0]; xor_5 = not_1 ^ not_2; xor_6 = xor_3 ^ not_1;
xor_7 = not_0 ^ x[6]; xor_8 = xor_2 ^ xor_3; xor_9 = xor_1 ^ x[5];
xor_10 = xor_3 ^ x[0]; xor_11 = xor_2 ^ xor_7; xor_12 = xor_6 ^ x[6];
xor_13 = xor_5 ^ x[3]; xor_14 = xor_0 ^ xor_4; xor_15 = not_2 ^ x[3];
xor_16 = xor_4 ^ not_0; xor_17 = xor_2 ^ not_0; xor_18 = xor_0 ^ xor_5;
xor_19 = xor_4 ^ x[5]; xor_20 = not_0 ^ not_1; xor_21 = xor_1 ^ xor_20;
xor_22 = xor_8 ^ x[6]; xor_23 = xor_9 ^ x[3]; xor_24 = xor_0 ^ xor_1;
xor_25 = xor_2 ^ xor_24; xor_26 = x[0] ^ x[6]; xor_27 = not_2 ^ x[0];
xor_28 = xor_0 ^ xor_27; and_0 = xor_10 & xor_11; and_1 = xor_0 & xor_6;
```

```
and_2 = xor_1 & xor_12; and_3 = xor_13 & xor_14; and_4 = xor_15 & xor_7;
and_5 = xor_16 & not_1; and_6 = xor_16 & xor_6; and_7 = xor_17 & xor_8;
xor_29 = and_2 ^ xor_2; and_8 = xor_18 & not_0; and_9 = xor_17 & xor_19;
and_10 = xor_21 & xor_22; and_11 = xor_23 & not_0; and_12 = xor_23 & xor_25;
xor_30 = and_6 ^ and_2; xor_31 = and_7 ^ not_1; xor_32 = and_7 ^ x[6];
and_13 = xor_26 & xor_9; xor_33 = and_8 ^ x[6]; xor_34 = and_3 ^ and_10;
xor_35 = and_1 ^ and_11; xor_36 = and_11 ^ xor_29; xor_37 = and_0 ^ and_12;
xor_38 = and_12 ^ not_1; xor_39 = and_13 ^ xor_28; xor_40 = and_5 ^ xor_33;
xor_41 = xor_10 ^ xor_33; xor_42 = and_4 ^ xor_34; xor_43 = xor_34 ^ xor_29;
xor_44 = xor_30 ^ xor_35; xor_45 = and_13 ^ xor_37; xor_46 = xor_30 ^ xor_38;
xor_47 = xor_31 ^ xor_39; xor_48 = and_9 ^ xor_40; xor_49 = xor_40 ^ xor_36;
xor_50 = xor_11 ^ xor_42; xor_51 = xor_31 ^ xor_43; xor_52 = xor_44 ^ xor_32;
xor_53 = xor_44 ^ xor_41; xor_54 = xor_35 ^ xor_45; xor_55 = xor_42 ^ xor_46;
xor_56 = xor_37 ^ xor_50; xor_57 = xor_48 ^ xor_51; xor_58 = xor_50 ^ xor_53;
xor_59 = xor_48 ^ xor_54; xor_60 = xor_5 ^ xor_56; xor_61 = xor_56 ^ xor_52;
xor_62 = xor_57 ^ xor_58; xor_63 = xor_58 ^ xor_55; xor_64 = xor_60 ^ xor_47;
xor_65 = xor_60 ^ xor_49; xor_66 = xor_61 ^ xor_57; xor_67 = xor_61 ^ xor_58;
xor_68 = xor_61 ^ xor_64; xor_69 = xor_59 ^ xor_64; xor_70 = xor_64 ^ xor_55;
and_14 = xor_57 & xor_64; xor_71 = xor_65 ^ xor_55; xor_72 = xor_57 ^ xor_65;
xor_73 = xor_64 ^ xor_65; xor_74 = xor_55 ^ xor_66; xor_75 = xor_59 ^ xor_67;
xor_76 = xor_55 ^ xor_68; xor_77 = xor_58 ^ xor_68; xor_78 = xor_59 ^ xor_68;
and_15 = xor_68 & xor_62; xor_79 = xor_57 ^ xor_69; and_16 = xor_65 & xor_69;
and_17 = xor_66 & xor_69; xor_80 = xor_57 ^ xor_71; xor_81 = xor_58 ^ xor_71;
xor_82 = xor_61 ^ xor_71; xor_83 = xor_59 ^ xor_71; xor_84 = xor_58 ^ xor_72;
and_18 = xor_72 & xor_70; xor_85 = xor_73 ^ xor_58; and_19 = xor_66 & xor_75;
and_20 = xor_67 & xor_76; and_21 = xor_70 & xor_77; and_22 = xor_79 & xor_76;
and_23 = xor_78 & xor_80; and_24 = xor_79 & xor_81; and_25 = xor_74 & xor_83;
and_26 = xor_84 & xor_63; xor_86 = and_18 ^ xor_65; xor_87 = and_18 ^ xor_55;
and_27 = xor_85 & xor_82; xor_88 = and_20 ^ xor_55; xor_89 = and_19 ^ and_22;
xor_90 = and_14 ^ and_24; xor_91 = and_24 ^ xor_65; xor_92 = and_17 ^ and_25;
xor_93 = and_25 ^ xor_61; xor_94 = and_26 ^ and_23; xor_95 = and_27 ^ xor_58;
xor_96 = and_16 ^ xor_88; xor_97 = xor_88 ^ xor_57; xor_98 = and_27 ^ xor_90;
xor_99 = xor_92 ^ xor_89; xor_100 = xor_92 ^ xor_91; xor_101 = and_22 ^ xor_93;
xor_102 = and_15 ^ xor_94; xor_103 = xor_94 ^ xor_93;
xor_104 = xor_86 ^ xor_95; xor_105 = and_21 ^ xor_96;
xor_106 = xor_89 ^ xor_98; xor_107 = xor_99 ^ xor_87;
xor_108 = xor_99 ^ xor_97; xor_109 = xor_96 ^ xor_101;
xor_110 = xor_102 ^ xor_64; xor_111 = xor_102 ^ xor_100;
xor_112 = xor_86 ^ xor_103; xor_113 = xor_105 ^ xor_106;
xor_114 = xor_90 ^ xor_110; xor_115 = xor_110 ^ xor_108;
xor_116 = xor_105 ^ xor_112; xor_117 = xor_114 ^ xor_59;
xor_118 = xor_114 ^ xor_107; xor_119 = xor_115 ^ xor_111;
xor_120 = xor_116 ^ xor_115; xor_121 = xor_117 ^ xor_104;
xor_122 = xor_117 ^ xor_109; xor_123 = xor_118 ^ xor_116;
xor_124 = xor_118 ^ xor_115; xor_125 = xor_118 ^ xor_121;
xor_126 = xor_113 ^ xor_121; xor_127 = xor_121 ^ xor_111;
and_28 = xor_116 & xor_121; xor_128 = xor_122 ^ xor_111;
xor_129 = xor_116 ^ xor_122; xor_130 = xor_121 ^ xor_122;
xor_131 = xor_111 ^ xor_123; xor_132 = xor_113 ^ xor_124;
xor_133 = xor_111 ^ xor_125; xor_134 = xor_115 ^ xor_125;
xor_135 = xor_113 ^ xor_125; and_29 = xor_125 & xor_120;
xor_136 = xor_116 ^ xor_126; and_30 = xor_122 & xor_126;
and_31 = xor_123 & xor_126; xor_137 = xor_116 ^ xor_128;
xor_138 = xor_115 ^ xor_128; xor_139 = xor_118 ^ xor_128;
xor_140 = xor_113 ^ xor_128; xor_141 = xor_115 ^ xor_129;
and_32 = xor_129 & xor_127; and_33 = xor_123 & xor_132;
and_34 = xor_124 & xor_133; and_35 = xor_127 & xor_134;
and_36 = xor_133 & xor_136; and_37 = xor_135 & xor_137;
and_38 = xor_136 & xor_138; and_39 = xor_131 & xor_140;
and_40 = xor_141 & xor_119; xor_142 = xor_130 ^ xor_115;
and_41 = xor_142 & xor_139; xor_143 = and_13 ^ not_2;
xor_144 = and_24 ^ and_38; xor_145 = and_28 ^ and_2; xor_146 = and_18 ^ and_7;
xor_147 = and_32 ^ xor_146; xor_148 = and_11 ^ xor_143;
xor_149 = xor_145 ^ xor_148; xor_150 = and_36 ^ x[6];
xor_151 = and_20 ^ xor_144; xor_152 = and_17 ^ x[3]; xor_153 = and_27 ^ and_5;
xor_154 = and_37 ^ and_9; xor_155 = and_14 ^ and_29;
xor_156 = and_40 ^ xor_154; xor_157 = xor_149 ^ xor_151;
xor_158 = and_35 ^ xor_150; xor_159 = and_4 ^ not_1; xor_160 = and_16 ^ and_31;
xor_161 = and_22 ^ xor_159; xor_162 = and_8 ^ not_0; xor_163 = and_30 ^ and_12;
xor_164 = and_21 ^ xor_147; xor_165 = and_33 ^ and_34;
xor_166 = xor_163 ^ xor_165; xor_167 = and_39 ^ xor_156;
xor_168 = xor_155 ^ xor_152; xor_169 = xor_153 ^ xor_157;
xor_170 = xor_158 ^ xor_160; xor_171 = xor_169 ^ xor_170;
xor_172 = and_15 ^ and_0; xor_173 = and_41 ^ xor_164;
xor_174 = xor_168 ^ xor_94; xor_175 = xor_172 ^ xor_174;
xor_176 = xor_162 ^ x[5]; xor_177 = and_19 ^ xor_166;
xor_178 = and_6 ^ xor_161; xor_179 = and_25 ^ xor_167;
xor_180 = xor_171 ^ x[0]; xor_181 = xor_175 ^ xor_180;
```

```
xor_182 = xor_177 ^ xor_181; xor_183 = xor_171 ^ xor_173;
xor_184 = xor_176 ^ xor_183; not_3 = ~xor_184; xor_185 = and_8 ^ xor_156;
xor_186 = xor_157 ^ xor_185; xor_187 = xor_173 ^ xor_186;
xor_188 = xor_175 ^ xor_187; xor_189 = xor_178 ^ xor_188;
xor_190 = and_27 ^ xor_149; xor_191 = xor_190 ^ xor_152;
xor_192 = xor_158 ^ xor_191; xor_193 = xor_161 ^ xor_192;
xor_194 = xor_176 ^ xor_193; xor_195 = xor_177 ^ xor_194;
xor_196 = xor_195 ^ xor_34; xor_197 = xor_179 ^ xor_196; not_4 = ~xor_197;
xor_198 = not_0 ^ x[3]; xor_199 = and_1 ^ xor_198; xor_200 = and_5 ^ xor_199;
xor_201 = and_30 ^ xor_200; xor_202 = xor_143 ^ xor_201;
xor_203 = xor_147 ^ xor_202; xor_204 = xor_203 ^ xor_150;
xor_205 = xor_204 ^ xor_151; xor_206 = xor_160 ^ xor_205;
xor_207 = xor_206 ^ xor_94; xor_208 = xor_207 ^ xor_34;
xor_209 = xor_178 ^ xor_208; not_5 = ~xor_209; xor_210 = and_13 ^ xor_26;
xor_211 = and_17 ^ xor_210; xor_212 = and_22 ^ xor_211;
xor_213 = xor_144 ^ xor_212; xor_214 = xor_153 ^ xor_213;
xor_215 = xor_155 ^ xor_214; xor_216 = xor_162 ^ xor_215;
xor_217 = xor_164 ^ xor_216; xor_218 = xor_166 ^ xor_217;
xor_219 = xor_167 ^ xor_218; xor_220 = and_31 ^ and_6;
xor_221 = and_33 ^ xor_220; xor_222 = and_36 ^ xor_221;
xor_223 = and_38 ^ xor_222; xor_224 = xor_145 ^ xor_223;
xor_225 = xor_147 ^ xor_224; xor_226 = xor_153 ^ xor_225;
xor_227 = xor_168 ^ xor_226; xor_228 = xor_179 ^ xor_227; y[0] = xor_182;
y[1] = not_3; y[2] = xor_189; y[3] = not_4; y[4] = not_5; y[5] = xor_219;
y[6] = xor_228;
```

C.25 7-bit power map x^7 (7-bit, Category A)

- **Profile**: $\delta = 6$, $L = 40$ (NL = 44), $\deg(S) = 3$, $\mathcal{D}(S) = [3^{127}]$, FP = 2, SAC deviation 0.125.
- **Cost**: 56 AND, 98 XOR, 0 NOT, AND depth $D = 2$ (unmasked NanGate 45 nm: 215.99 μm^2 / 270.7 GE, delay 772.4 ps, logic depth 14; pipelined $d = 2$ DOM-indep: 280 `DFF_X1` [224 gadget + 56 sync], 2174.82 μm^2 / 2725.3 GE, stage delay 765.9 ps).
- **Remark**: The exact lookup table of x^7 over $\mathbb{F}_2[x]/(x^7+x+1)$; 30/140/2 in [BDM26b]: 26 more AND gates and 42 fewer XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 5$, $\mathcal{D}(S^{-1}) = [5^{127}]$; cost: 51 AND, 220 XOR, 15 NOT, AND depth $D = 3$ (Tier 3: 3-stage power-map factorization $y^{109} = ((y^5)^5)^{64}(y^5)^{32}$ at $D = 3$; unmasked NanGate 45 nm: 413.36 μm^2 / 518.0 GE, delay 1904.0 ps, logic depth 33; pipelined $d = 2$ DOM-indep: 284 `DFF_X1` [204 gadget + 80 sync], 2537.11 μm^2 / 3179.3 GE, stage delay 976.9 ps).

Lookup table (LUT[0 . . . 127]):

```
0x00, 0x01, 0x03, 0x7c, 0x05, 0x2e, 0x07, 0x30, 0x0f, 0x70, 0x72, 0x0d, 0x09, 0x7d, 0x50, 0x46
0x11, 0x4c, 0x13, 0x62, 0x15, 0x1e, 0x17, 0x52, 0x1b, 0x7f, 0x04, 0x32, 0x73, 0x60, 0x49, 0x6a
0x33, 0x71, 0x57, 0x40, 0x35, 0x44, 0x25, 0x63, 0x3f, 0x1f, 0x22, 0x29, 0x39, 0x0b, 0x75, 0x0e
0x2d, 0x76, 0x02, 0x5e, 0x0c, 0x18, 0x56, 0x27, 0x16, 0x68, 0x23, 0x24, 0x58, 0x48, 0x3d, 0x36
0x55, 0x7e, 0x10, 0x1a, 0x7a, 0x51, 0x43, 0x2b, 0x5f, 0x45, 0x4f, 0x0a, 0x6f, 0x54, 0x26, 0x20
0x41, 0x2f, 0x21, 0x3c, 0x66, 0x74, 0x7b, 0x78, 0x4b, 0x53, 0x1d, 0x08, 0x1c, 0x59, 0x12, 0x38
0x77, 0x2a, 0x19, 0x4e, 0x06, 0x42, 0x61, 0x4d, 0x14, 0x64, 0x28, 0x2c, 0x79, 0x31, 0x69, 0x37
0x3a, 0x67, 0x3b, 0x3e, 0x65, 0x5d, 0x6c, 0x6e, 0x6b, 0x5c, 0x5b, 0x4a, 0x47, 0x34, 0x5a, 0x6d
```

Netlist (56 AND, 98 XOR, 0 NOT, $D = 2$):

```
xor_0 = x[0] ^ x[5]; and_0 = x[0] & x[1]; and_1 = x[0] & x[2];
and_2 = x[1] & x[2]; and_3 = x[0] & x[3]; and_4 = x[1] & x[3];
and_5 = x[2] & x[3]; and_6 = x[0] & x[4]; and_7 = x[1] & x[4];
and_8 = x[2] & x[4]; and_9 = x[3] & x[4]; and_10 = x[0] & x[5];
and_11 = x[1] & x[5]; and_12 = x[2] & x[5]; and_13 = x[3] & x[5];
and_14 = x[4] & x[5]; and_15 = x[0] & x[6]; and_16 = x[1] & x[6];
and_17 = x[2] & x[6]; and_18 = x[3] & x[6]; and_19 = x[4] & x[6];
and_20 = x[5] & x[6]; and_21 = x[0] & and_2; and_22 = x[0] & and_4;
and_23 = x[0] & and_5; and_24 = x[1] & and_5; and_25 = x[0] & and_7;
and_26 = x[0] & and_8; and_27 = x[1] & and_8; and_28 = x[0] & and_9;
and_29 = x[1] & and_9; and_30 = x[2] & and_9; and_31 = x[0] & and_11;
and_32 = x[0] & and_12; and_33 = x[1] & and_12; and_34 = x[0] & and_13;
and_35 = x[1] & and_13; and_36 = x[2] & and_13; and_37 = x[0] & and_14;
and_38 = x[1] & and_14; and_39 = x[2] & and_14; and_40 = x[3] & and_14;
```

```
and_41 = x[0] & and_16; and_42 = x[0] & and_17; and_43 = x[1] & and_17;
and_44 = x[0] & and_18; and_45 = x[1] & and_18; and_46 = x[2] & and_18;
and_47 = x[0] & and_19; and_48 = x[1] & and_19; and_49 = x[2] & and_19;
and_50 = x[3] & and_19; and_51 = x[0] & and_20; and_52 = x[1] & and_20;
and_53 = x[2] & and_20; and_54 = x[3] & and_20; and_55 = x[4] & and_20;
xor_1 = and_0 ^ and_4; xor_2 = and_3 ^ and_22; xor_3 = xor_1 ^ xor_2;
xor_4 = and_43 ^ xor_3; xor_5 = and_41 ^ and_53; xor_6 = and_26 ^ xor_4;
xor_7 = and_30 ^ and_44; xor_8 = and_31 ^ and_45; xor_9 = and_6 ^ xor_6;
xor_10 = and_11 ^ xor_7; xor_11 = xor_5 ^ xor_9; xor_12 = x[6] ^ and_33;
xor_13 = and_28 ^ xor_10; xor_14 = x[3] ^ and_14; xor_15 = and_46 ^ xor_11;
xor_16 = and_42 ^ and_54; xor_17 = and_1 ^ xor_8; xor_18 = xor_16 ^ xor_17;
xor_19 = and_37 ^ xor_12; xor_20 = and_32 ^ and_47; xor_21 = and_10 ^ and_29;
xor_22 = and_25 ^ and_55; xor_23 = xor_13 ^ xor_19; xor_24 = and_35 ^ xor_14;
xor_25 = and_15 ^ and_17; xor_26 = xor_18 ^ xor_25; xor_27 = and_24 ^ and_51;
xor_28 = x[5] ^ and_40; xor_29 = and_34 ^ xor_15; xor_30 = and_39 ^ xor_26;
xor_31 = and_12 ^ xor_24; xor_32 = and_48 ^ xor_27; xor_33 = x[2] ^ and_9;
xor_34 = xor_23 ^ xor_33; xor_35 = and_13 ^ xor_21; xor_36 = and_36 ^ xor_35;
xor_37 = and_21 ^ xor_30; xor_38 = and_5 ^ xor_31; xor_39 = x[1] ^ xor_38;
xor_40 = and_16 ^ xor_22; xor_41 = and_52 ^ xor_29; xor_42 = and_50 ^ xor_34;
xor_43 = and_19 ^ xor_42; xor_44 = and_18 ^ xor_43; xor_45 = x[4] ^ and_20;
xor_46 = xor_36 ^ xor_39; xor_47 = xor_20 ^ xor_45; xor_48 = and_38 ^ xor_40;
xor_49 = xor_28 ^ xor_37; xor_50 = xor_0 ^ and_2; xor_51 = and_7 ^ and_8;
xor_52 = xor_50 ^ xor_51; xor_53 = xor_52 ^ and_23; xor_54 = and_27 ^ and_43;
xor_55 = and_49 ^ xor_5; xor_56 = xor_8 ^ xor_27; xor_57 = xor_44 ^ xor_46;
xor_58 = xor_47 ^ xor_53; xor_59 = xor_54 ^ xor_55; xor_60 = xor_56 ^ xor_57;
xor_61 = xor_58 ^ xor_59; xor_62 = xor_60 ^ xor_61; xor_63 = and_41 ^ xor_3;
xor_64 = xor_13 ^ xor_46; xor_65 = xor_48 ^ xor_49; xor_66 = xor_63 ^ xor_64;
xor_67 = xor_65 ^ xor_66; xor_68 = and_16 ^ and_39; xor_69 = and_55 ^ xor_24;
xor_70 = xor_32 ^ xor_41; xor_71 = xor_44 ^ xor_68; xor_72 = xor_69 ^ xor_70;
xor_73 = xor_71 ^ xor_72; xor_74 = xor_14 ^ and_38; xor_75 = and_49 ^ xor_7;
xor_76 = xor_20 ^ xor_26; xor_77 = xor_41 ^ xor_74; xor_78 = xor_75 ^ xor_76;
xor_79 = xor_77 ^ xor_78; xor_80 = and_51 ^ xor_12; xor_81 = xor_22 ^ xor_28;
xor_82 = xor_29 ^ xor_47; xor_83 = xor_80 ^ xor_81; xor_84 = xor_82 ^ xor_83;
xor_85 = and_11 ^ and_23; xor_86 = and_29 ^ and_30; xor_87 = and_33 ^ xor_6;
xor_88 = xor_32 ^ xor_49; xor_89 = xor_85 ^ xor_86; xor_90 = xor_87 ^ xor_88;
xor_91 = xor_89 ^ xor_90; xor_92 = and_21 ^ and_52; xor_93 = and_54 ^ xor_11;
xor_94 = xor_21 ^ xor_23; xor_95 = xor_48 ^ xor_92; xor_96 = xor_93 ^ xor_94;
xor_97 = xor_95 ^ xor_96; y[0] = xor_62; y[1] = xor_67; y[2] = xor_73;
y[3] = xor_79; y[4] = xor_84; y[5] = xor_91; y[6] = xor_97;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 127]$):

```
0x00, 0x01, 0x32, 0x02, 0x1a, 0x04, 0x64, 0x06, 0x5b, 0x0c, 0x4b, 0x2d, 0x34, 0x0b, 0x2f, 0x08
0x42, 0x10, 0x5e, 0x12, 0x68, 0x14, 0x38, 0x16, 0x35, 0x62, 0x43, 0x18, 0x5c, 0x5a, 0x15, 0x29
0x4f, 0x52, 0x2a, 0x3a, 0x3b, 0x26, 0x4e, 0x37, 0x6a, 0x2b, 0x61, 0x47, 0x6b, 0x30, 0x05, 0x51
0x07, 0x6d, 0x1b, 0x20, 0x7d, 0x24, 0x3f, 0x6f, 0x5f, 0x2c, 0x70, 0x72, 0x53, 0x3e, 0x73, 0x28
0x23, 0x50, 0x65, 0x46, 0x25, 0x49, 0x0f, 0x7c, 0x3d, 0x1e, 0x7b, 0x58, 0x11, 0x67, 0x63, 0x4a
0x0e, 0x45, 0x17, 0x59, 0x4d, 0x40, 0x36, 0x22, 0x3c, 0x5d, 0x7e, 0x7a, 0x79, 0x75, 0x33, 0x48
0x1d, 0x66, 0x13, 0x27, 0x69, 0x74, 0x54, 0x71, 0x39, 0x6e, 0x1f, 0x78, 0x76, 0x7f, 0x77, 0x4c
0x09, 0x21, 0x0a, 0x1c, 0x55, 0x2e, 0x31, 0x60, 0x57, 0x6c, 0x44, 0x56, 0x03, 0x0d, 0x41, 0x19
```

Inverse netlist (51 AND, 220 XOR, 15 NOT, $D = 3$):

```
not_0 = ~x[0]; not_1 = ~x[2]; not_2 = ~x[4]; not_3 = ~x[6];
xor_0 = not_2 ^ x[3]; xor_1 = not_3 ^ x[1]; xor_2 = not_1 ^ x[5];
xor_3 = xor_0 ^ not_1; xor_4 = not_0 ^ x[5]; xor_5 = xor_4 ^ not_3;
xor_6 = xor_2 ^ not_2; xor_7 = xor_0 ^ xor_1; xor_8 = not_1 ^ not_2;
xor_9 = not_2 ^ x[5]; xor_10 = xor_1 ^ x[3]; xor_11 = xor_3 ^ not_3;
xor_12 = xor_2 ^ not_3; xor_13 = not_1 ^ x[3]; xor_14 = not_1 ^ x[1];
xor_15 = xor_0 ^ xor_5; xor_16 = xor_3 ^ x[1]; xor_17 = not_0 ^ x[3];
xor_18 = xor_6 ^ not_0; xor_19 = xor_6 ^ x[1]; xor_20 = xor_7 ^ x[5];
xor_21 = x[1] ^ x[3]; xor_22 = xor_5 ^ x[3]; xor_23 = xor_0 ^ not_0;
xor_24 = xor_2 ^ xor_23; xor_25 = xor_1 ^ not_1; xor_26 = not_0 ^ not_1;
xor_27 = xor_1 ^ xor_3; xor_28 = xor_2 ^ x[3]; xor_29 = x[3] ^ x[5];
xor_30 = not_0 ^ x[1]; and_0 = xor_8 & xor_9; and_1 = xor_10 & not_1;
and_2 = xor_0 & xor_11; and_3 = xor_12 & xor_13; and_4 = xor_14 & xor_8;
and_5 = xor_15 & xor_7; and_6 = xor_16 & xor_17; and_7 = xor_17 & xor_7;
and_8 = xor_18 & xor_19; and_9 = xor_20 & xor_21; and_10 = xor_20 & xor_22;
and_11 = xor_24 & xor_25; and_12 = xor_26 & xor_27; and_13 = xor_28 & xor_5;
xor_31 = and_0 ^ not_0; xor_32 = and_8 ^ xor_31; xor_33 = xor_29 ^ xor_32;
xor_34 = and_1 ^ not_2; xor_35 = and_3 ^ and_4; xor_36 = not_1 ^ xor_33;
xor_37 = and_12 ^ xor_34; xor_38 = and_7 ^ not_3; xor_39 = and_2 ^ and_5;
xor_40 = xor_35 ^ xor_39; xor_41 = and_11 ^ xor_36; xor_42 = and_10 ^ xor_38;
xor_43 = xor_35 ^ xor_37; xor_44 = and_7 ^ xor_30; xor_45 = xor_34 ^ xor_44;
xor_46 = xor_40 ^ xor_45; xor_47 = and_1 ^ and_9; xor_48 = and_13 ^ xor_47;
xor_49 = xor_40 ^ xor_48; xor_50 = xor_41 ^ xor_49; xor_51 = xor_42 ^ xor_50;
not_4 = ~xor_51; xor_52 = and_6 ^ xor_41; xor_53 = xor_43 ^ xor_52;
xor_54 = xor_36 ^ x[1]; xor_55 = xor_37 ^ xor_54; xor_56 = xor_42 ^ xor_55;
not_5 = ~xor_56; xor_57 = and_2 ^ xor_1; xor_58 = and_6 ^ xor_57;
```

```
xor_59 = and_11 ^ xor_58; xor_60 = and_13 ^ xor_59; xor_61 = xor_33 ^ xor_60;
xor_62 = and_9 ^ xor_32; xor_63 = xor_43 ^ xor_62; not_6 = ~xor_63;
xor_64 = and_3 ^ xor_8; xor_65 = and_5 ^ xor_64; xor_66 = and_6 ^ xor_65;
xor_67 = and_8 ^ xor_66; xor_68 = xor_38 ^ xor_67; not_7 = ~xor_68;
not_8 = ~xor_46; not_9 = ~xor_53; xor_69 = not_5 ^ xor_61;
xor_70 = not_9 ^ not_4; xor_71 = xor_68 ^ xor_69; xor_72 = not_6 ^ not_8;
xor_73 = not_9 ^ xor_61; xor_74 = not_6 ^ xor_61; xor_75 = not_4 ^ xor_71;
xor_76 = not_4 ^ not_5; xor_77 = not_9 ^ not_5; xor_78 = xor_68 ^ xor_72;
xor_79 = xor_68 ^ xor_76; xor_80 = not_9 ^ xor_71; xor_81 = not_9 ^ not_6;
xor_82 = xor_68 ^ xor_81; xor_83 = xor_71 ^ xor_72; xor_84 = xor_69 ^ xor_70;
xor_85 = not_5 ^ not_8; xor_86 = xor_72 ^ xor_73; xor_87 = xor_70 ^ xor_74;
xor_88 = not_6 ^ xor_75; xor_89 = not_5 ^ xor_78; xor_90 = not_9 ^ xor_69;
xor_91 = xor_72 ^ xor_90; xor_92 = xor_68 ^ xor_70; xor_93 = not_9 ^ not_8;
xor_94 = xor_70 ^ xor_71; xor_95 = not_6 ^ xor_77; and_14 = xor_73 & xor_74;
and_15 = not_9 & xor_79; and_16 = xor_69 & xor_80; and_17 = xor_77 & xor_82;
and_18 = xor_70 & xor_73; and_19 = xor_75 & xor_83; and_20 = xor_84 & xor_85;
and_21 = xor_75 & xor_85; and_22 = xor_86 & xor_87; and_23 = xor_76 & xor_88;
and_24 = xor_88 & xor_89; and_25 = xor_91 & xor_92; and_26 = xor_93 & xor_94;
and_27 = xor_78 & xor_95; xor_96 = and_14 ^ not_8; xor_97 = and_22 ^ xor_96;
xor_98 = not_5 ^ not_6; xor_99 = xor_97 ^ xor_98; xor_100 = and_15 ^ xor_61;
xor_101 = and_17 ^ and_18; xor_102 = not_9 ^ xor_99;
xor_103 = and_26 ^ xor_100; xor_104 = and_21 ^ xor_68;
xor_105 = and_16 ^ and_19; xor_106 = xor_101 ^ xor_105;
xor_107 = and_25 ^ xor_102; xor_108 = and_24 ^ xor_104;
xor_109 = xor_101 ^ xor_103; xor_110 = not_4 ^ not_8;
xor_111 = and_21 ^ xor_110; xor_112 = xor_111 ^ xor_100;
xor_113 = xor_106 ^ xor_112; xor_114 = and_15 ^ and_23;
xor_115 = and_27 ^ xor_114; xor_116 = xor_106 ^ xor_115;
xor_117 = xor_107 ^ xor_116; xor_118 = xor_108 ^ xor_117; not_10 = ~xor_118;
xor_119 = and_20 ^ xor_107; xor_120 = xor_109 ^ xor_119;
xor_121 = not_4 ^ xor_102; xor_122 = xor_103 ^ xor_121;
xor_123 = xor_108 ^ xor_122; not_11 = ~xor_123; xor_124 = not_4 ^ xor_68;
xor_125 = and_16 ^ xor_124; xor_126 = and_20 ^ xor_125;
xor_127 = and_25 ^ xor_126; xor_128 = and_27 ^ xor_127;
xor_129 = xor_128 ^ xor_99; not_12 = ~xor_129; xor_130 = and_23 ^ xor_97;
xor_131 = xor_109 ^ xor_130; not_13 = ~xor_131; xor_132 = and_17 ^ xor_73;
xor_133 = and_19 ^ xor_132; xor_134 = and_20 ^ xor_133;
xor_135 = and_22 ^ xor_134; xor_136 = xor_104 ^ xor_135; not_14 = ~xor_136;
xor_137 = not_4 ^ xor_53; xor_138 = not_6 ^ not_7; xor_139 = not_4 ^ xor_138;
xor_140 = not_10 ^ xor_120; xor_141 = not_11 ^ not_12;
xor_142 = not_13 ^ not_14; and_28 = xor_113 & xor_46;
and_29 = xor_84 & xor_140; xor_143 = xor_113 ^ xor_140;
xor_144 = xor_84 ^ xor_46; and_30 = xor_143 & xor_144;
and_31 = xor_139 & xor_141; and_32 = xor_142 & xor_98;
xor_145 = xor_141 ^ xor_142; xor_146 = xor_139 ^ xor_98;
and_33 = xor_145 & xor_146; xor_147 = xor_113 ^ xor_141;
xor_148 = xor_140 ^ xor_142; xor_149 = xor_139 ^ xor_46;
and_34 = xor_147 & xor_149; and_35 = xor_148 & xor_87;
xor_150 = xor_147 ^ xor_148; xor_151 = xor_149 ^ xor_87;
and_36 = xor_150 & xor_151; and_37 = not_10 & xor_137;
and_38 = not_11 & xor_138; and_39 = not_13 & not_5; xor_152 = not_10 ^ not_11;
xor_153 = xor_137 ^ xor_138; and_40 = xor_152 & xor_153;
xor_154 = not_10 ^ not_13; xor_155 = not_5 ^ xor_137;
and_41 = xor_154 & xor_155; xor_156 = not_11 ^ not_13;
xor_157 = not_5 ^ xor_138; and_42 = xor_156 & xor_157;
xor_158 = not_10 ^ xor_113; xor_159 = not_11 ^ xor_140;
xor_160 = not_13 ^ xor_141; xor_161 = xor_137 ^ xor_46;
xor_162 = xor_138 ^ xor_84; xor_163 = not_5 ^ xor_139;
and_43 = xor_158 & xor_161; and_44 = xor_159 & xor_162;
xor_164 = xor_158 ^ xor_159; xor_165 = xor_161 ^ xor_162;
and_45 = xor_164 & xor_165; and_46 = xor_160 & xor_163;
xor_166 = xor_142 ^ xor_160; xor_167 = xor_163 ^ xor_98;
and_47 = xor_166 & xor_167; xor_168 = xor_158 ^ xor_160;
xor_169 = xor_142 ^ xor_159; xor_170 = xor_161 ^ xor_163;
xor_171 = xor_162 ^ xor_98; and_48 = xor_168 & xor_170;
and_49 = xor_169 & xor_171; xor_172 = xor_168 ^ xor_169;
xor_173 = xor_170 ^ xor_171; and_50 = xor_172 & xor_173;
xor_174 = and_29 ^ and_43; xor_175 = and_34 ^ and_47;
xor_176 = and_42 ^ xor_174; xor_177 = and_28 ^ and_31;
xor_178 = and_33 ^ and_38; xor_179 = xor_175 ^ xor_178;
xor_180 = and_35 ^ and_39; xor_181 = and_45 ^ xor_176;
xor_182 = and_32 ^ and_41; xor_183 = and_44 ^ xor_177;
xor_184 = and_36 ^ and_48; xor_185 = and_50 ^ xor_179;
xor_186 = xor_184 ^ xor_181; xor_187 = xor_185 ^ xor_186;
xor_188 = and_49 ^ xor_180; xor_189 = and_30 ^ and_46;
xor_190 = and_40 ^ xor_183; xor_191 = and_31 ^ and_44;
xor_192 = xor_187 ^ xor_191; xor_193 = xor_188 ^ xor_192;
xor_194 = xor_189 ^ xor_193; xor_195 = and_37 ^ xor_187;
```

```
xor_196 = xor_188 ^ xor_179; xor_197 = xor_190 ^ xor_196;
xor_198 = and_28 ^ and_29; xor_199 = and_36 ^ xor_198;
xor_200 = and_40 ^ xor_199; xor_201 = xor_175 ^ xor_200;
xor_202 = xor_201 ^ xor_180; xor_203 = xor_202 ^ xor_182;
xor_204 = xor_189 ^ xor_203; xor_205 = and_35 ^ xor_176;
xor_206 = xor_177 ^ xor_205; xor_207 = xor_206 ^ xor_182;
xor_208 = and_30 ^ and_32; xor_209 = and_33 ^ xor_208;
xor_210 = and_37 ^ xor_209; xor_211 = xor_210 ^ xor_181;
xor_212 = xor_190 ^ xor_211; xor_213 = and_34 ^ and_37;
xor_214 = and_38 ^ xor_213; xor_215 = and_46 ^ xor_214;
xor_216 = and_48 ^ xor_215; xor_217 = xor_174 ^ xor_216;
xor_218 = xor_217 ^ xor_182; xor_219 = xor_218 ^ xor_183; y[0] = xor_194;
y[1] = xor_195; y[2] = xor_197; y[3] = xor_204; y[4] = xor_207; y[5] = xor_212;
y[6] = xor_219;
```

C.26 $\mathsf{YSH}_{8,1}$: 8-bit degree 7, $\delta = 6$, $L = 56$ (8-bit, Category B)

- **Profile**: $\delta = 6$, $L = 56$ (NL = 100), $\deg(S) = 7$, $\mathcal{D}(S) = [6^1, 7^{254}]$, FP = 1, SAC deviation 0.0781.
- **Cost**: 43 AND, 107 XOR, 7 NOT, AND depth $D = 4$ (unmasked NanGate 45 nm: 220.25 μm^2 / 276.0 GE, delay 1529.9 ps, logic depth 25; pipelined $d = 2$ DOM-indep: 236 `DFF_X1` [172 gadget + 64 sync], 1869.98 μm^2 / 2343.3 GE, stage delay 574.7 ps).
- **Remark**: Degree 7, the maximum for an 8-bit permutation, at $\delta = 6$, $L = 56$. The AES circuits of Table 4 are better in every metric.
- **Inverse ($\mathsf{YSH}_{8,1}^{-1}$)**: $\deg(S^{-1}) = 7$, $\mathcal{D}(S^{-1}) = [6^1, 7^{254}]$; cost: 43 AND, 109 XOR, 10 NOT, AND depth $D = 4$ (Tier 2: Output shear unwinding + 42-AND $D = 3$ $\mathbb{F}_{2^8}$ inversion core; unmasked NanGate 45 nm: 225.04 μm^2 / 282.0 GE, delay 1380.5 ps, logic depth 24; pipelined $d = 2$ DOM-indep: 264 `DFF_X1` [172 gadget + 92 sync], 2004.58 μm^2 / 2512.0 GE, stage delay 578.1 ps).

Lookup table (LUT[0 . . . 255]):

```
0xc1, 0x7c, 0x5c, 0xb9, 0x2c, 0x83, 0x98, 0x02, 0xea, 0xe9, 0x88, 0x15, 0x10, 0xf9, 0xe8, 0xc9
0x84, 0x12, 0xae, 0xd3, 0x86, 0x85, 0x9c, 0x1a, 0x60, 0x1b, 0x74, 0x3e, 0x1d, 0x5b, 0x8e, 0x54
0x3f, 0x48, 0xcb, 0x16, 0x29, 0xff, 0xe0, 0xf8, 0xb5, 0xc4, 0xca, 0x69, 0xfa, 0xcf, 0xbe, 0xee
0xec, 0x64, 0x68, 0xe1, 0x06, 0xb0, 0x0c, 0xd1, 0x05, 0x20, 0x50, 0xe4, 0xa2, 0x49, 0x0f, 0xa3
0x7e, 0xfd, 0x4b, 0xed, 0x5a, 0x45, 0x1c, 0x18, 0x38, 0x63, 0x04, 0xf2, 0x35, 0x95, 0xb7, 0xde
0xab, 0x6e, 0x0d, 0x9d, 0xc5, 0xdf, 0xe6, 0x52, 0x81, 0xc2, 0x8c, 0x37, 0x4e, 0x9e, 0x32, 0x4a
0x1f, 0xf4, 0x72, 0x4d, 0xf6, 0xdc, 0x3c, 0xb1, 0x56, 0x41, 0x5d, 0x3a, 0x17, 0xa9, 0x0b, 0xd8
0x44, 0x5f, 0x3d, 0x1e, 0x4c, 0x80, 0x27, 0x26, 0x59, 0x2a, 0x6b, 0x2b, 0x46, 0x36, 0xdb, 0x33
0xba, 0x90, 0x11, 0x53, 0x76, 0xb4, 0x2f, 0x8a, 0xe3, 0x55, 0x30, 0x3b, 0xf1, 0x40, 0x89, 0xad
0xd5, 0xd2, 0x42, 0x14, 0x61, 0xb6, 0xd9, 0x22, 0x19, 0xcc, 0xa4, 0xf7, 0xef, 0x96, 0x67, 0x6a
0xc8, 0x24, 0xc0, 0x9f, 0x62, 0xa0, 0x75, 0xc6, 0x8f, 0x7a, 0xe7, 0x94, 0xf0, 0x9a, 0xda, 0x43
0x66, 0x0e, 0xfc, 0xb8, 0x47, 0x78, 0xc7, 0xd6, 0xfe, 0x21, 0x71, 0xf5, 0xaf, 0x51, 0x77, 0x79
0x2e, 0x07, 0x93, 0x58, 0xb2, 0xd4, 0x13, 0x82, 0x28, 0x57, 0x73, 0x6d, 0xd7, 0x25, 0xa5, 0x5e
0xa7, 0x97, 0x2d, 0x65, 0x6c, 0x99, 0xbd, 0x8b, 0xf3, 0xbf, 0xeb, 0x7f, 0x00, 0x39, 0x87, 0x08
0x6f, 0x92, 0x8d, 0xe2, 0xc3, 0x0a, 0x70, 0xaa, 0x34, 0x7d, 0xce, 0x09, 0x91, 0xdd, 0x23, 0xfb
0x9b, 0xac, 0xbc, 0xd0, 0xbb, 0xa6, 0x31, 0x7b, 0xe5, 0xa8, 0x4f, 0x01, 0xb3, 0xa1, 0xcd, 0x03
```

Netlist (43 AND, 107 XOR, 7 NOT, $D = 4$):

```
a_in = in_val[7:0];
  split_1 = a_in[0:0]; split_2 = a_in[1:1]; split_3 = a_in[2:2];
  split_4 = a_in[3:3]; split_5 = a_in[4:4]; split_6 = a_in[5:5];
  split_7 = a_in[6:6]; split_8 = a_in[7:7]; xor_15 = split_1 ^ split_3;
  xor_11 = split_4 ^ split_5; xor_9 = split_1 ^ split_6;
  xor_12 = split_5 ^ split_6; xor_19 = split_2 ^ split_7;
  xor_16 = split_8 ^ xor_15; xor_21 = split_2 ^ xor_11; xor_10 = split_7 ^ xor_9;
  xor_13 = split_8 ^ xor_12; xor_23 = split_3 ^ xor_12; xor_20 = split_8 ^ xor_19;
  not_17 = ~xor_16; xor_18 = split_3 ^ xor_10; xor_22 = xor_11 ^ xor_10;
  not_14 = ~xor_13; paar_shared_xor_10_xor_21_xor_23 = xor_21 ^ xor_23;
  and_31 = xor_20 & xor_10; xor_54 = xor_20 ^ xor_10;
  paar_shared_xor_7_xor_20_xor_21 = xor_20 ^ xor_21;
  paar_shared_xor_11_not_17_xor_10 = xor_10 ^ not_17; and_25 = xor_23 & xor_18;
  xor_57 = xor_23 ^ xor_18; paar_shared_xor_6_not_17_xor_18 = not_17 ^ xor_18;
  and_24 = xor_22 & not_17; xor_56 = xor_22 ^ not_17;
  paar_shared_xor_5_xor_22_xor_23 = xor_22 ^ xor_23;
  paar_shared_xor_9_xor_20_xor_22 = xor_20 ^ xor_22; and_32 = xor_21 & not_14;
```

```
xor_55 = xor_21 ^ not_14; paar_shared_xor_8_not_14_xor_10 = xor_10 ^ not_14;
paar_shared_xor_12_not_14_xor_18 = not_14 ^ xor_18; xor_58 = xor_54 ^ xor_55;
xor_59 = xor_55 ^ xor_56;
and_28 = paar_shared_xor_5_xor_22_xor_23 & paar_shared_xor_6_not_17_xor_18;
and_42 = paar_shared_xor_9_xor_20_xor_22 & paar_shared_xor_11_not_17_xor_10;
xor_44 = paar_shared_xor_9_xor_20_xor_22 ^ paar_shared_xor_10_xor_21_xor_23;
xor_60 = xor_58 ^ xor_56; xor_61 = xor_59 ^ xor_57;
and_35 = paar_shared_xor_7_xor_20_xor_21 & paar_shared_xor_8_not_14_xor_10;
and_43 = paar_shared_xor_10_xor_21_xor_23 & paar_shared_xor_12_not_14_xor_18;
xor_45 = paar_shared_xor_11_not_17_xor_10 ^ paar_shared_xor_12_not_14_xor_18;
xor_29 = and_28 ^ and_24; xor_30 = and_28 ^ and_25; xor_36 = and_35 ^ and_31;
xor_37 = and_35 ^ and_32; and_46 = xor_44 & xor_45; xor_47 = and_46 ^ and_42;
xor_48 = and_46 ^ and_43; xor_51 = xor_37 ^ xor_47; xor_53 = xor_30 ^ xor_47;
xor_49 = xor_47 ^ xor_48; xor_63 = xor_51 ^ xor_61; xor_65 = xor_53 ^ xor_55;
xor_50 = xor_36 ^ xor_49; xor_52 = xor_29 ^ xor_49; and_79 = xor_18 & xor_63;
and_109 = xor_23 & xor_63; and_67 = xor_65 & xor_63; and_86 = not_14 & xor_65;
and_116 = xor_21 & xor_65; paar_shared_xor_4_xor_63_xor_65 = xor_63 ^ xor_65;
xor_62 = xor_50 ^ xor_60; xor_64 = xor_52 ^ xor_58; and_78 = not_17 & xor_62;
and_108 = xor_22 & xor_62; paar_shared_xor_2_xor_62_xor_63 = xor_62 ^ xor_63;
and_66 = xor_64 & xor_62; and_85 = xor_10 & xor_64; and_115 = xor_20 & xor_64;
paar_shared_xor_1_xor_64_xor_65 = xor_64 ^ xor_65;
paar_shared_xor_3_xor_62_xor_64 = xor_62 ^ xor_64;
and_97 = paar_shared_xor_12_not_14_xor_18 & paar_shared_xor_4_xor_63_xor_65;
and_127 = paar_shared_xor_10_xor_21_xor_23 & paar_shared_xor_4_xor_63_xor_65;
and_82 = paar_shared_xor_6_not_17_xor_18 & paar_shared_xor_2_xor_62_xor_63;
and_112 = paar_shared_xor_5_xor_22_xor_23 & paar_shared_xor_2_xor_62_xor_63;
and_70 = paar_shared_xor_1_xor_64_xor_65 & paar_shared_xor_2_xor_62_xor_63;
and_89 = paar_shared_xor_8_not_14_xor_10 & paar_shared_xor_1_xor_64_xor_65;
and_119 = paar_shared_xor_7_xor_20_xor_21 & paar_shared_xor_1_xor_64_xor_65;
xor_75 = paar_shared_xor_3_xor_62_xor_64 ^ paar_shared_xor_4_xor_63_xor_65;
and_96 = paar_shared_xor_11_not_17_xor_10 & paar_shared_xor_3_xor_62_xor_64;
and_126 = paar_shared_xor_9_xor_20_xor_22 & paar_shared_xor_3_xor_62_xor_64;
xor_83 = and_82 ^ and_78; xor_84 = and_82 ^ and_79; xor_113 = and_112 ^ and_108;
xor_114 = and_112 ^ and_109; xor_71 = and_70 ^ and_66; xor_72 = and_70 ^ and_67;
xor_90 = and_89 ^ and_85; xor_91 = and_89 ^ and_86; xor_120 = and_119 ^ and_115;
xor_121 = and_119 ^ and_116; and_100 = xor_45 & xor_75;
and_130 = xor_44 & xor_75; xor_76 = xor_71 ^ xor_75;
xor_77 = xor_72 ^ paar_shared_xor_4_xor_63_xor_65; xor_101 = and_100 ^ and_96;
xor_102 = and_100 ^ and_97; xor_131 = and_130 ^ and_126;
xor_132 = and_130 ^ and_127; paar_shared_xor_0_xor_76_xor_77 = xor_76 ^ xor_77;
xor_105 = xor_91 ^ xor_101; xor_107 = xor_84 ^ xor_101;
xor_103 = xor_101 ^ xor_102; xor_135 = xor_121 ^ xor_131;
xor_137 = xor_114 ^ xor_131; xor_133 = xor_131 ^ xor_132;
and_139 = xor_105 & xor_76; and_146 = xor_107 & xor_76;
xor_104 = xor_90 ^ xor_103; xor_106 = xor_83 ^ xor_103;
and_153 = xor_135 & xor_76; and_160 = xor_137 & xor_76;
xor_134 = xor_120 ^ xor_133; xor_136 = xor_113 ^ xor_133;
and_138 = xor_104 & xor_77; xor_140 = xor_104 ^ xor_105;
and_145 = xor_106 & xor_77; xor_147 = xor_106 ^ xor_107;
and_152 = xor_134 & xor_77; xor_154 = xor_134 ^ xor_135;
and_159 = xor_136 & xor_77; xor_161 = xor_136 ^ xor_137;
and_142 = xor_140 & paar_shared_xor_0_xor_76_xor_77;
and_149 = xor_147 & paar_shared_xor_0_xor_76_xor_77;
and_156 = xor_154 & paar_shared_xor_0_xor_76_xor_77;
and_163 = xor_161 & paar_shared_xor_0_xor_76_xor_77; xor_143 = and_142 ^ and_138;
xor_144 = and_142 ^ and_139; xor_150 = and_149 ^ and_145;
xor_151 = and_149 ^ and_146; xor_157 = and_156 ^ and_152;
xor_158 = and_156 ^ and_153; xor_164 = and_163 ^ and_159;
xor_165 = and_163 ^ and_160; xor_168 = xor_143 ^ xor_144;
xor_174 = xor_143 ^ xor_150; xor_178 = xor_150 ^ xor_151;
xor_167 = xor_158 ^ xor_150; xor_177 = xor_157 ^ xor_164;
xor_166 = xor_165 ^ xor_151; xor_172 = xor_157 ^ xor_168;
xor_170 = xor_167 ^ xor_168; xor_179 = xor_177 ^ xor_178;
xor_169 = xor_157 ^ xor_166; xor_180 = xor_166 ^ xor_167; not_173 = ~xor_172;
xor_171 = xor_164 ^ xor_170; xor_183 = xor_166 ^ xor_170;
xor_175 = xor_174 ^ xor_169; xor_181 = xor_158 ^ xor_169; not_176 = ~xor_171;
xor_185 = xor_169 ^ xor_171; not_184 = ~xor_183; not_182 = ~xor_181;
and_188 = xor_175 & not_176; not_186 = ~xor_185; xor_189 = not_173 ^ and_188;
concat_190 = { not_186, not_184, not_182, xor_180, xor_179, not_176, xor_175, xor_189 };
out_val = concat_190;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 255]$):

```
0xdc, 0xfb, 0x07, 0xff, 0x4a, 0x38, 0x34, 0xc1, 0xdf, 0xeb, 0xe5, 0x6e, 0x36, 0x52, 0xb1, 0x3e
0x0c, 0x82, 0x11, 0xc6, 0x93, 0x0b, 0x23, 0x6c, 0x47, 0x98, 0x17, 0x19, 0x46, 0x1c, 0x73, 0x60
0x39, 0xb9, 0x97, 0xee, 0xa1, 0xcd, 0x77, 0x76, 0xc8, 0x24, 0x79, 0x7b, 0x04, 0xd2, 0xc0, 0x86
0x8a, 0xf6, 0x5e, 0x7f, 0xe8, 0x4c, 0x7d, 0x5b, 0x48, 0xdd, 0x6b, 0x8b, 0x66, 0x72, 0x1b, 0x20
0x8d, 0x69, 0x92, 0xaf, 0x70, 0x45, 0x7c, 0xb4, 0x21, 0x3d, 0x5f, 0x42, 0x74, 0x63, 0x5c, 0xfa
0x3a, 0xbd, 0x57, 0x83, 0x1f, 0x89, 0x68, 0xc9, 0xc3, 0x78, 0x44, 0x1d, 0x02, 0x6a, 0xcf, 0x71
```

```
0x18, 0x94, 0xa4, 0x49, 0x31, 0xd3, 0xb0, 0x9e, 0x32, 0x2b, 0x9f, 0x7a, 0xd4, 0xcb, 0x51, 0xe0
0xe6, 0xba, 0x62, 0xca, 0x1a, 0xa6, 0x84, 0xbe, 0xb5, 0xbf, 0xa9, 0xf7, 0x01, 0xe9, 0x40, 0xdb
0x75, 0x58, 0xc7, 0x05, 0x10, 0x15, 0x14, 0xde, 0x0a, 0x8e, 0x87, 0xd7, 0x5a, 0xe2, 0x1e, 0xa8
0x81, 0xec, 0xe1, 0xc2, 0xab, 0x4d, 0x9d, 0xd1, 0x06, 0xd5, 0xad, 0xf0, 0x16, 0x53, 0x5d, 0xa3
0xa5, 0xfd, 0x3c, 0x3f, 0x9a, 0xce, 0xf5, 0xd0, 0xf9, 0x6d, 0xe7, 0x50, 0xf1, 0x8f, 0x12, 0xbc
0x35, 0x67, 0xc4, 0xfc, 0x85, 0x28, 0x95, 0x4e, 0xb3, 0x03, 0x80, 0xf4, 0xf2, 0xd6, 0x2e, 0xd9
0xa2, 0x00, 0x59, 0xe4, 0x29, 0x54, 0xa7, 0xb6, 0xa0, 0x0f, 0x2a, 0x22, 0x99, 0xfe, 0xea, 0x2d
0xf3, 0x37, 0x91, 0x13, 0xc5, 0x90, 0xb7, 0xcc, 0x6f, 0x96, 0xae, 0x7e, 0x65, 0xed, 0x4f, 0x55
0x26, 0x33, 0xe3, 0x88, 0x3b, 0xf8, 0x56, 0xaa, 0x0e, 0x09, 0x08, 0xda, 0x30, 0x43, 0x2f, 0x9c
0xac, 0x8c, 0x4b, 0xd8, 0x61, 0xbb, 0x64, 0x9b, 0x27, 0x0d, 0x2c, 0xef, 0xb2, 0x41, 0xb8, 0x25
```

Inverse netlist (43 AND, 109 XOR, 10 NOT, $D = 4$):

```
and_0 = x[1] & x[2]; xor_0 = and_0 ^ x[0]; not_0 = ~xor_0; not_1 = ~x[2];
not_2 = ~x[5]; not_3 = ~x[6]; not_4 = ~x[7]; xor_1 = not_2 ^ x[4];
xor_2 = not_3 ^ x[1]; xor_3 = not_0 ^ not_4; xor_4 = x[3] ^ xor_2;
xor_5 = not_1 ^ xor_1; xor_6 = not_4 ^ xor_4; xor_7 = xor_1 ^ xor_3;
xor_8 = not_0 ^ not_1; xor_9 = not_1 ^ xor_3; xor_10 = not_0 ^ x[1];
xor_11 = x[1] ^ x[3]; xor_12 = not_1 ^ not_2; xor_13 = not_1 ^ x[1];
xor_14 = x[4] ^ xor_13; xor_15 = x[3] ^ xor_7; xor_16 = xor_5 ^ xor_10;
xor_17 = xor_2 ^ xor_8; xor_18 = not_3 ^ x[3]; xor_19 = xor_18 ^ xor_8;
xor_20 = not_2 ^ x[1]; xor_21 = xor_20 ^ xor_3; xor_22 = not_0 ^ not_2;
xor_23 = xor_5 ^ xor_6; xor_24 = xor_1 ^ xor_4; xor_25 = not_4 ^ xor_2;
xor_26 = xor_12 ^ xor_6; xor_27 = xor_3 ^ xor_5; xor_28 = xor_4 ^ xor_9;
xor_29 = not_2 ^ xor_10; xor_30 = xor_11 ^ xor_3; xor_31 = x[3] ^ xor_12;
and_1 = xor_14 & xor_7; and_2 = xor_15 & xor_16; and_3 = xor_17 & xor_19;
and_4 = xor_21 & xor_9; and_5 = x[3] & xor_22; and_6 = xor_23 & xor_24;
xor_32 = and_1 ^ and_5; xor_33 = and_2 ^ and_5; and_7 = xor_25 & xor_26;
and_8 = xor_27 & xor_5; xor_34 = and_4 ^ and_7; xor_35 = and_3 ^ and_7;
and_9 = xor_28 & xor_6; xor_36 = and_8 ^ and_9; xor_37 = and_6 ^ and_9;
xor_38 = xor_35 ^ xor_36; xor_39 = xor_33 ^ xor_36; xor_40 = xor_36 ^ xor_37;
xor_41 = xor_29 ^ xor_38; xor_42 = xor_11 ^ xor_39; xor_43 = xor_34 ^ xor_40;
xor_44 = xor_32 ^ xor_40; and_10 = xor_16 & xor_41; and_11 = xor_15 & xor_41;
and_12 = xor_41 & xor_42; and_13 = xor_19 & xor_42; and_14 = xor_17 & xor_42;
xor_45 = xor_41 ^ xor_42; xor_46 = xor_30 ^ xor_43; xor_47 = xor_31 ^ xor_44;
and_15 = xor_14 & xor_46; and_16 = xor_46 & xor_7; xor_48 = xor_41 ^ xor_46;
and_17 = xor_46 & xor_47; and_18 = xor_21 & xor_47; and_19 = xor_47 & xor_9;
xor_49 = xor_42 ^ xor_47; xor_50 = xor_46 ^ xor_47; and_20 = xor_24 & xor_45;
and_21 = xor_23 & xor_45; and_22 = xor_22 & xor_48; and_23 = x[3] & xor_48;
and_24 = xor_48 & xor_49; and_25 = xor_26 & xor_49; and_26 = xor_25 & xor_49;
xor_51 = xor_45 ^ xor_50; and_27 = xor_27 & xor_50; and_28 = xor_5 & xor_50;
xor_52 = and_15 ^ and_22; xor_53 = and_22 ^ and_10; xor_54 = and_16 ^ and_23;
xor_55 = and_11 ^ and_23; xor_56 = and_17 ^ and_24; xor_57 = and_12 ^ and_24;
xor_58 = and_18 ^ and_25; xor_59 = and_13 ^ and_25; xor_60 = and_19 ^ and_26;
xor_61 = and_14 ^ and_26; and_29 = xor_28 & xor_51; and_30 = xor_6 & xor_51;
xor_62 = xor_51 ^ xor_56; xor_63 = xor_45 ^ xor_57; xor_64 = and_27 ^ and_29;
xor_65 = and_20 ^ and_29; xor_66 = and_28 ^ and_30; xor_67 = and_21 ^ and_30;
xor_68 = xor_62 ^ xor_63; xor_69 = xor_59 ^ xor_64; xor_70 = xor_53 ^ xor_64;
xor_71 = xor_64 ^ xor_65; xor_72 = xor_61 ^ xor_66; xor_73 = xor_55 ^ xor_66;
xor_74 = xor_66 ^ xor_67; and_31 = xor_62 & xor_69; and_32 = xor_62 & xor_70;
xor_75 = xor_58 ^ xor_71; xor_76 = xor_52 ^ xor_71; and_33 = xor_62 & xor_72;
and_34 = xor_62 & xor_73; xor_77 = xor_60 ^ xor_74; xor_78 = xor_54 ^ xor_74;
and_35 = xor_63 & xor_75; xor_79 = xor_69 ^ xor_75; and_36 = xor_63 & xor_76;
xor_80 = xor_70 ^ xor_76; and_37 = xor_63 & xor_77; xor_81 = xor_72 ^ xor_77;
and_38 = xor_63 & xor_78; xor_82 = xor_73 ^ xor_78; and_39 = xor_68 & xor_79;
and_40 = xor_68 & xor_80; and_41 = xor_68 & xor_81; and_42 = xor_68 & xor_82;
xor_83 = and_35 ^ and_38; xor_84 = and_37 ^ and_42; xor_85 = and_32 ^ xor_83;
xor_86 = and_40 ^ xor_84; xor_87 = and_34 ^ and_36; xor_88 = and_39 ^ xor_85;
xor_89 = and_31 ^ xor_85; xor_90 = and_33 ^ xor_86; xor_91 = and_41 ^ xor_86;
xor_92 = xor_87 ^ xor_89; xor_93 = and_36 ^ and_42; xor_94 = xor_88 ^ xor_93;
xor_95 = xor_88 ^ xor_90; xor_96 = and_36 ^ and_39; xor_97 = xor_83 ^ xor_96;
xor_98 = xor_91 ^ xor_97; not_5 = ~xor_98; xor_99 = xor_89 ^ xor_90;
not_6 = ~xor_99; xor_100 = and_31 ^ and_35; xor_101 = xor_87 ^ xor_100;
xor_102 = xor_101 ^ xor_91; not_7 = ~xor_102; xor_103 = and_32 ^ and_41;
xor_104 = xor_103 ^ xor_84; xor_105 = xor_104 ^ xor_87; not_8 = ~xor_105;
xor_106 = and_33 ^ and_38; xor_107 = and_41 ^ xor_106;
xor_108 = and_42 ^ xor_107; not_9 = ~xor_108; y[0] = xor_92; y[1] = xor_94;
y[2] = xor_95; y[3] = not_5; y[4] = not_6; y[5] = not_7; y[6] = not_8;
y[7] = not_9;
```

C.27 $\mathbf{YSH_{8,2}}$: 8-bit degree 7, $\delta = 10$, $L = 72$, 28 AND (8-bit, Category B)

- **Profile**: $\delta = 10$, $L = 72$ (NL = 92), $\deg(S) = 7$, $\mathcal{D}(S) = [6^3, 7^{252}]$, FP = 0, SAC deviation 0.1094.

- **Cost**: 28 AND, 61 XOR, 1 NOT, AND depth $D = 6$ (unmasked NanGate 45 nm: 127.68 μm^2 / 160.0 GE, delay 969.0 ps, logic depth 17; pipelined $d = 2$ DOM-indep: 208 `DFF_X1` [112 gadget + 96 sync], 1433.74 μm^2 / 1796.7 GE, stage delay 275.1 ps).
- **Remark**: A six-round generalized Feistel network over single-bit branches (with $4, 6, 4, 4, 4, 6$ AND gates per level) that Stage 2 evolved from the Canright tower-field AES seed (36/136/4 [Can05]; Section 6.5): 8 fewer AND and 75 fewer XOR gates than the seed at two more AND levels and the relaxed profile $(\delta, L) = (10, 72)$. It realizes the 8-bit $(10, 72, 7)$ profile with 28 AND gates where the ANF-synthesized archive representatives of the same profile need 239 (Table 11), with $\beta = 22$ and no fixed point; $\mathsf{YSH}_{8,1}$ and the AES circuits remain better in δ and L. Stage-2 AND erasure applied to this circuit produces the AND-versus-δ family of Table 8, whose 18-AND member is $\mathsf{YSH}_{8,3}$ (SM C.28).
- **Inverse ($\mathsf{YSH}_{8,2}^{-1}$)**: $\deg(S^{-1}) = 7$, $\mathcal{D}(S^{-1}) = [6^3, 7^{252}]$; cost: 28 AND, 84 XOR, 2 NOT, AND depth $D = 6$ (Tier 2: Exact 6-round generalized Feistel layer reversal; unmasked NanGate 45 nm: 164.92 μm^2 / 206.7 GE, delay 1524.5 ps, logic depth 28; pipelined $d = 2$ DOM-indep: 296 `DFF_X1` [112 gadget + 184 sync], 1905.62 μm^2 / 2388.0 GE, stage delay 431.9 ps).

Lookup table (LUT$[0 \ldots 255]$):

```
0x32, 0x24, 0x45, 0xdc, 0x07, 0xb7, 0xde, 0x62, 0xd1, 0xc4, 0xcb, 0x0c, 0xbb, 0x2e, 0x3f, 0x8f
0x03, 0x00, 0xd4, 0xf4, 0x15, 0xd6, 0xee, 0x67, 0x70, 0x83, 0x63, 0x89, 0xa1, 0xb5, 0x81, 0xc9
0x51, 0xbc, 0x55, 0xe0, 0xf7, 0x2f, 0x64, 0xa4, 0x22, 0xf9, 0x97, 0x9a, 0x6f, 0xeb, 0xae, 0x93
0x86, 0x92, 0xab, 0x23, 0xf1, 0x05, 0x3b, 0x1b, 0x88, 0xb3, 0xd3, 0xb4, 0xda, 0xb1, 0x38, 0x1c
0x0a, 0xea, 0x9c, 0x04, 0xa5, 0xd8, 0x3a, 0xac, 0x95, 0x71, 0x19, 0x9e, 0xdd, 0x0b, 0xc7, 0xf6
0x6b, 0xba, 0x6a, 0x8c, 0x7b, 0x08, 0xd0, 0xe9, 0xb8, 0x7e, 0x76, 0xcf, 0xfe, 0x9d, 0x2c, 0x17
0x1a, 0xe6, 0x2d, 0xbd, 0xe1, 0x96, 0x12, 0xc6, 0x31, 0x11, 0x21, 0x50, 0x99, 0x4b, 0x8b, 0x9b
0xfc, 0xff, 0x20, 0x42, 0x28, 0x61, 0xad, 0xb0, 0x1e, 0xc0, 0x8a, 0x5e, 0xd9, 0x34, 0x7f, 0xe4
0x10, 0x4f, 0x98, 0xa7, 0x2a, 0x0e, 0xd2, 0xa8, 0xc1, 0xfd, 0x36, 0x06, 0x8e, 0x7d, 0x8d, 0x14
0x6d, 0xa6, 0xf5, 0xb6, 0xa9, 0x84, 0x13, 0x0f, 0xaf, 0x5f, 0xe5, 0x41, 0xbf, 0x39, 0xd7, 0x57
0x2b, 0xb9, 0x56, 0x47, 0x82, 0x1d, 0x53, 0xf2, 0x66, 0x27, 0x4d, 0xfb, 0x02, 0x26, 0xa2, 0xca
0xe8, 0xe3, 0x79, 0x65, 0x59, 0xf3, 0x44, 0xfa, 0x6e, 0x0d, 0xf8, 0x75, 0x80, 0xed, 0x18, 0x01
0x54, 0x5b, 0x74, 0x40, 0xdb, 0x91, 0x30, 0x4a, 0xce, 0x52, 0x09, 0xef, 0x87, 0x7a, 0xec, 0xc5
0x68, 0x7c, 0x6c, 0x5c, 0xe2, 0xaa, 0x77, 0x9f, 0x3d, 0xc2, 0x72, 0x69, 0xc3, 0x43, 0x5a, 0xcd
0xbe, 0xd5, 0xb2, 0x60, 0x25, 0x33, 0x4e, 0xdf, 0x16, 0x58, 0x49, 0xa3, 0x35, 0x78, 0x37, 0x73
0x94, 0x48, 0x90, 0x1f, 0x46, 0x3c, 0xf0, 0x4c, 0x3e, 0x5d, 0x29, 0xcc, 0x85, 0xa0, 0xe7, 0xc8
```

Netlist (28 AND, 61 XOR, 1 NOT, $D = 6$):

```
xor_0 = x[0] ^ x[4]; and_0 = x[4] & x[5]; xor_1 = x[3] ^ x[5];
and_1 = x[4] & x[6]; and_2 = x[5] & x[6]; xor_2 = x[2] ^ x[6];
and_3 = x[6] & x[7]; xor_3 = x[1] ^ x[7]; xor_4 = xor_0 ^ and_0;
xor_5 = xor_1 ^ and_2; xor_6 = xor_2 ^ and_3; xor_7 = xor_3 ^ and_1;
xor_8 = x[7] ^ xor_4; and_4 = xor_4 & xor_5; xor_9 = x[4] ^ xor_5;
xor_10 = xor_5 ^ xor_4; and_5 = xor_4 & xor_6; and_6 = xor_6 & xor_5;
xor_11 = x[6] ^ xor_6; and_7 = xor_4 & xor_7; and_8 = xor_7 & xor_6;
and_9 = xor_7 & xor_5; xor_12 = x[5] ^ xor_7; xor_13 = and_4 ^ and_6;
xor_14 = and_7 ^ and_5; xor_15 = and_5 ^ and_8; xor_16 = and_4 ^ and_9;
xor_17 = xor_12 ^ xor_13; xor_18 = xor_11 ^ xor_14; xor_19 = xor_8 ^ xor_15;
xor_20 = xor_9 ^ xor_16; xor_21 = xor_6 ^ xor_17; and_10 = xor_18 & xor_4;
xor_22 = xor_7 ^ xor_18; and_11 = xor_17 & xor_18; and_12 = xor_18 & xor_19;
and_13 = xor_17 & xor_19; not_0 = ~xor_20; xor_23 = xor_10 ^ and_10;
xor_24 = xor_21 ^ and_11; xor_25 = xor_22 ^ and_13; xor_26 = not_0 ^ xor_19;
xor_27 = xor_19 ^ xor_23; xor_28 = xor_17 ^ xor_24; and_14 = xor_25 & xor_23;
xor_29 = xor_4 ^ xor_25; and_15 = xor_25 & xor_24; xor_30 = xor_26 ^ and_12;
and_16 = xor_24 & xor_30; and_17 = xor_23 & xor_30; xor_31 = xor_18 ^ xor_30;
xor_32 = xor_30 ^ xor_25; xor_33 = and_14 ^ and_16; xor_34 = and_15 ^ and_16;
xor_35 = and_15 ^ and_17; xor_36 = and_14 ^ and_17; xor_37 = xor_28 ^ xor_33;
xor_38 = xor_27 ^ xor_34; xor_39 = xor_31 ^ xor_35; xor_40 = xor_29 ^ xor_36;
xor_41 = xor_24 ^ xor_38; and_18 = xor_39 & xor_38; and_19 = xor_39 & xor_25;
xor_42 = xor_37 ^ xor_39; and_20 = xor_38 & xor_40; and_21 = xor_40 & xor_25;
xor_43 = xor_23 ^ xor_40; xor_44 = xor_32 ^ and_19; xor_45 = xor_42 ^ and_18;
xor_46 = xor_41 ^ and_21; xor_47 = xor_43 ^ and_20; xor_48 = xor_38 ^ xor_44;
and_22 = xor_44 & xor_45; xor_49 = xor_40 ^ xor_45; and_23 = xor_46 & xor_44;
xor_50 = xor_39 ^ xor_46; and_24 = xor_46 & xor_45; and_25 = xor_46 & xor_47;
and_26 = xor_47 & xor_44; xor_51 = xor_25 ^ xor_47; and_27 = xor_47 & xor_45;
xor_52 = and_23 ^ and_24; xor_53 = and_25 ^ and_22; xor_54 = and_23 ^ and_26;
xor_55 = and_26 ^ and_27; xor_56 = xor_49 ^ xor_52; xor_57 = xor_50 ^ xor_53;
```

```
xor_58 = xor_48 ^ xor_54; xor_59 = xor_51 ^ xor_55; xor_60 = xor_45 ^ xor_56;
y[0] = xor_47; y[1] = xor_44; y[2] = xor_60; y[3] = xor_57; y[4] = xor_58;
y[5] = xor_56; y[6] = xor_59; y[7] = xor_46;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 255]$):

```
0x11, 0xbf, 0xac, 0x10, 0x43, 0x35, 0x8b, 0x04, 0x55, 0xca, 0x40, 0x4d, 0x0b, 0xb9, 0x85, 0x97
0x80, 0x69, 0x66, 0x96, 0x8f, 0x14, 0xe8, 0x5f, 0xbe, 0x4a, 0x60, 0x37, 0x3f, 0xa5, 0x78, 0xf3
0x72, 0x6a, 0x28, 0x33, 0x01, 0xe4, 0xad, 0xa9, 0x74, 0xfa, 0x84, 0xa0, 0x5e, 0x62, 0x0d, 0x25
0xc6, 0x68, 0x00, 0xe5, 0x7d, 0xec, 0x8a, 0xee, 0x3e, 0x9d, 0x46, 0x36, 0xf5, 0xd8, 0xf8, 0x0e
0xc3, 0x9b, 0x73, 0xdd, 0xb6, 0x02, 0xf4, 0xa3, 0xf1, 0xea, 0xc7, 0x6d, 0xf7, 0xaa, 0xe6, 0x81
0x6b, 0x20, 0xc9, 0xa6, 0xc0, 0x22, 0xa2, 0x9f, 0xe9, 0xb4, 0xde, 0xc1, 0xd3, 0xf9, 0x7b, 0x99
0xe3, 0x75, 0x07, 0x1a, 0x26, 0xb3, 0xa8, 0x17, 0xd0, 0xdb, 0x52, 0x50, 0xd2, 0x90, 0xb8, 0x2c
0x18, 0x49, 0xda, 0xef, 0xc2, 0xbb, 0x5a, 0xd6, 0xed, 0xb2, 0xcd, 0x54, 0xd1, 0x8d, 0x59, 0x7e
0xbc, 0x1e, 0xa4, 0x19, 0x95, 0xfc, 0x30, 0xcc, 0x38, 0x1b, 0x7a, 0x6e, 0x53, 0x8e, 0x8c, 0x0f
0xf2, 0xc5, 0x31, 0x2f, 0xf0, 0x48, 0x65, 0x2a, 0x82, 0x6c, 0x2b, 0x6f, 0x42, 0x5d, 0x4b, 0xd7
0xfd, 0x1c, 0xae, 0xeb, 0x27, 0x44, 0x91, 0x83, 0x87, 0x94, 0xd5, 0x32, 0x47, 0x76, 0x2e, 0x98
0x77, 0x3d, 0xe2, 0x39, 0x3b, 0x1d, 0x93, 0x05, 0x58, 0xa1, 0x51, 0x0c, 0x21, 0x63, 0xe0, 0x9c
0x79, 0x88, 0xd9, 0xdc, 0x09, 0xcf, 0x67, 0x4e, 0xff, 0x1f, 0xaf, 0x0a, 0xfb, 0xdf, 0xc8, 0x5b
0x56, 0x08, 0x86, 0x3a, 0x12, 0xe1, 0x15, 0x9e, 0x45, 0x7c, 0x3c, 0xc4, 0x03, 0x4c, 0x06, 0xe7
0x23, 0x64, 0xd4, 0xb1, 0x7f, 0x9a, 0x61, 0xfe, 0xb0, 0x57, 0x41, 0x2d, 0xce, 0xbd, 0x16, 0xcb
0xf6, 0x34, 0xa7, 0xb5, 0x13, 0x92, 0x4f, 0x24, 0xba, 0x29, 0xb7, 0xab, 0x70, 0x89, 0x5c, 0x71
```

Inverse netlist (28 AND, 84 XOR, 2 NOT, $D = 6$):

```
xor_0 = x[2] ^ x[5]; xor_1 = x[1] ^ x[7]; and_0 = x[1] & xor_0;
and_1 = x[7] & x[1]; and_2 = x[7] & xor_0; and_3 = x[7] & x[0];
and_4 = x[0] & x[1]; and_5 = x[0] & xor_0; xor_2 = x[6] ^ and_5;
xor_3 = and_4 ^ xor_2; xor_4 = x[0] ^ xor_3; xor_5 = x[3] ^ and_3;
xor_6 = and_0 ^ xor_5; xor_7 = x[4] ^ and_1; xor_8 = x[7] ^ xor_6;
xor_9 = and_4 ^ xor_7; xor_10 = x[2] ^ and_1; xor_11 = and_2 ^ xor_10;
xor_12 = xor_2 ^ xor_7; xor_13 = x[1] ^ xor_9; xor_14 = x[0] ^ xor_12;
and_6 = xor_8 & xor_13; and_7 = xor_8 & xor_4; and_8 = xor_13 & xor_11;
and_9 = xor_11 & xor_4; xor_15 = x[5] ^ and_6; xor_16 = and_9 ^ xor_9;
xor_17 = and_8 ^ xor_12; xor_18 = x[0] ^ and_8; xor_19 = xor_11 ^ xor_18;
xor_20 = and_7 ^ xor_4; xor_21 = xor_16 ^ xor_1; xor_22 = x[1] ^ xor_20;
xor_23 = and_9 ^ xor_17; and_10 = xor_4 & xor_19; and_11 = xor_4 & xor_21;
and_12 = xor_21 & xor_22; and_13 = xor_19 & xor_22; xor_24 = and_10 ^ and_13;
xor_25 = and_7 ^ and_11; xor_26 = x[2] ^ and_12; xor_27 = xor_26 ^ xor_15;
xor_28 = x[1] ^ xor_25; xor_29 = xor_24 ^ xor_4; xor_30 = xor_8 ^ xor_28;
xor_31 = and_13 ^ xor_30; xor_32 = and_2 ^ xor_26; xor_33 = xor_27 ^ xor_16;
xor_34 = and_4 ^ xor_32; xor_35 = x[4] ^ xor_34; xor_36 = and_10 ^ xor_8;
xor_37 = xor_29 ^ xor_11; xor_38 = x[1] ^ and_11; xor_39 = xor_4 ^ xor_31;
xor_40 = and_12 ^ xor_17; xor_41 = xor_38 ^ xor_18; xor_42 = and_10 ^ xor_6;
xor_43 = x[1] ^ xor_33; xor_44 = xor_35 ^ xor_41; xor_45 = xor_42 ^ xor_43;
xor_46 = and_7 ^ xor_36; and_14 = xor_39 & xor_37; and_15 = xor_45 & xor_39;
and_16 = xor_39 & xor_44; and_17 = xor_45 & xor_44; xor_47 = and_15 ^ xor_27;
xor_48 = and_14 ^ xor_24; xor_49 = x[7] ^ and_17; xor_50 = xor_29 ^ xor_47;
xor_51 = xor_3 ^ xor_48; xor_52 = and_17 ^ xor_31; xor_53 = xor_47 ^ xor_36;
xor_54 = and_8 ^ xor_51; xor_55 = and_12 ^ and_14; xor_56 = xor_14 ^ xor_55;
xor_57 = xor_27 ^ and_14; xor_58 = xor_23 ^ xor_57; and_18 = xor_37 & xor_54;
and_19 = xor_37 & xor_53; and_20 = xor_53 & xor_54; and_21 = xor_37 & xor_52;
and_22 = xor_52 & xor_53; and_23 = xor_52 & xor_54; xor_59 = and_18 ^ xor_25;
xor_60 = and_19 ^ and_21; xor_61 = and_16 ^ and_23; xor_62 = xor_59 ^ xor_49;
xor_63 = xor_59 ^ xor_61; xor_64 = xor_28 ^ xor_60; xor_65 = and_20 ^ xor_62;
xor_66 = xor_50 ^ xor_64; xor_67 = and_22 ^ xor_40; xor_68 = xor_35 ^ xor_63;
xor_69 = xor_24 ^ xor_65; xor_70 = and_19 ^ xor_67; xor_71 = xor_33 ^ xor_69;
xor_72 = xor_48 ^ xor_68; xor_73 = xor_38 ^ xor_70; xor_74 = xor_24 ^ xor_73;
xor_75 = and_17 ^ xor_70; xor_76 = xor_39 ^ xor_60; not_0 = ~xor_72;
and_24 = not_0 & xor_71; and_25 = not_0 & xor_66; and_26 = xor_71 & xor_66;
and_27 = xor_66 & xor_74; xor_77 = xor_63 ^ and_24; xor_78 = xor_77 ^ xor_56;
xor_79 = and_25 ^ xor_46; xor_80 = xor_75 ^ xor_79; xor_81 = and_27 ^ xor_76;
xor_82 = xor_65 ^ and_26; xor_83 = xor_82 ^ xor_58; not_1 = ~xor_78;
y[0] = not_1; y[1] = xor_80; y[2] = xor_81; y[3] = xor_83; y[4] = not_0;
y[5] = xor_71; y[6] = xor_66; y[7] = xor_74;
```

C.28 $\mathrm{YSH}_{8,3}$: 8-bit degree 7, $\delta = 14$, $L = 72$, 18 AND (8-bit, Category B)

- **Profile**: $\delta = 14$, $L = 72$ (NL = 92), $\deg(S) = 7$, $\mathcal{D}(S) = [4^1, 5^2, 6^{60}, 7^{192}]$, FP = 0, SAC deviation 0.1094.
- **Cost**: 18 AND, 71 XOR, 2 NOT, AND depth $D = 5$ (unmasked NanGate 45 nm: 133.53 μm^2 / 167.3 GE, delay 1084.7 ps, logic depth 18; pipelined $d = 2$ DOM-indep: 160 `DFF_X1` [72 gadget + 88 sync], 1142.74 μm^2 / 1432.0 GE, stage delay 431.6 ps).

- **Remark**: The 18-AND member of the Stage-2 AND-erasure family of $\mathsf{YSH}_{8,2}$ (Table 8; Section 6.5): seeded with $\mathsf{YSH}_{8,2}$ itself as an archive elite (28/62, $D = 6$), AND erasure removed ten AND gates and one level ($3, 5, 3, 4, 3$ AND gates per level) and moved the profile from $(\delta, L) = (10, 72)$ to $(14, 72)$ while keeping degree 7, $\beta = 38$, and no fixed point. The 63 components of degree at most 6 form a 6-dimensional subspace ($k = 6$ split $[4^1, 5^2, 6^{60}, 7^{192}]$, nested over the 2-dimensional subspace of the three components of degree at most 5); the inverse has $\mathcal{D}(S^{-1}) = [5^1, 6^{62}, 7^{192}]$. As a pipelined $d = 2$ DOM-indep module it is the smallest 8-bit circuit of the benchmark set ($1,142.74\,\mu\text{m}^2$, 72 gadget flip-flops), although its unmasked area ($133.53\,\mu\text{m}^2$) exceeds that of $\mathsf{YSH}_{8,2}$ because of its ten additional XOR gates. Stage-4 post-processing reduced the archived 18/73/2 netlist to 18/71/2; the inverse (Tier 2 backward unrolling) is one level deeper than S.
- **Inverse ($\mathsf{YSH}_{8,3}^{-1}$)**: $\deg(S^{-1}) = 7$, $\mathcal{D}(S^{-1}) = [5^1, 6^{62}, 7^{192}]$; cost: 18 AND, 77 XOR, 8 NOT, AND depth $D = 6$ (Tier 2: Structural backward unrolling with linear-span closure (one level deeper than the forward circuit); unmasked NanGate 45 nm: $146.30\,\mu\text{m}^2$ / 183.3 GE, delay 1389.2 ps, logic depth 28; pipelined $d = 2$ DOM-indep: 274 `DFF_X1` [72 gadget + 202 sync], $1680.59\,\mu\text{m}^2$ / 2106.0 GE, stage delay 438.1 ps).

Lookup table (LUT$[0 \ldots 255]$):

```
0xcb, 0xa8, 0xb4, 0x5a, 0x79, 0x77, 0xc2, 0xe6, 0x67, 0x90, 0xfd, 0x6b, 0xf8, 0xb9, 0x8c, 0x5c
0x20, 0x08, 0x24, 0x3b, 0x59, 0x3e, 0xdc, 0x75, 0xee, 0x21, 0x8b, 0xff, 0x4a, 0x33, 0x22, 0x89
0xf2, 0x66, 0x74, 0xc5, 0xf5, 0x48, 0x2f, 0x46, 0x01, 0x99, 0xcd, 0xc4, 0xc0, 0x94, 0xdb, 0xb0
0xa7, 0x54, 0x11, 0xea, 0xbb, 0xa0, 0xbd, 0x09, 0x02, 0xd6, 0x8a, 0xd2, 0x81, 0x56, 0x78, 0x38
0xe2, 0xb8, 0x7d, 0xbe, 0x5f, 0xc7, 0x62, 0x1e, 0x53, 0x64, 0x5d, 0x40, 0xde, 0x44, 0x91, 0x93
0xf9, 0xa2, 0x50, 0xad, 0x2c, 0x37, 0x03, 0x2e, 0xec, 0x7c, 0xae, 0xf0, 0x73, 0xb6, 0x16, 0xf1
0xb3, 0x17, 0x69, 0x86, 0x00, 0xba, 0xc3, 0x35, 0x55, 0xd1, 0x10, 0x34, 0x6d, 0x82, 0x07, 0x88
0xf3, 0xd4, 0x9d, 0x51, 0x60, 0x36, 0x8f, 0x80, 0xd3, 0x2a, 0x14, 0x42, 0x29, 0xda, 0x6f, 0xa5
0xfa, 0x25, 0x1c, 0xef, 0x72, 0xc6, 0x31, 0x1b, 0xa6, 0xc9, 0xbf, 0x4f, 0x5b, 0x30, 0xe9, 0x7a
0xfb, 0xe3, 0xa1, 0x19, 0x04, 0x3d, 0xdf, 0xaa, 0xe7, 0x0c, 0xb5, 0x2d, 0xfc, 0xbc, 0x71, 0xc8
0xac, 0xfe, 0x9f, 0x9b, 0xf7, 0x0e, 0xdd, 0xb2, 0x85, 0x83, 0xca, 0x05, 0xe1, 0x15, 0x18, 0x4c
0x23, 0x0d, 0x70, 0x13, 0x95, 0xb1, 0x47, 0xcc, 0x3a, 0x0b, 0x39, 0x9e, 0x84, 0x7f, 0x97, 0x52
0x2b, 0x1a, 0xa4, 0xab, 0x4e, 0xf6, 0x76, 0x6e, 0x7e, 0x1d, 0xed, 0xb7, 0x68, 0xc1, 0xaf, 0x98
0xa9, 0x43, 0xd0, 0xe0, 0x3c, 0xe5, 0x41, 0x45, 0x61, 0x32, 0x1f, 0x8d, 0x8e, 0x92, 0xa3, 0x87
0x3f, 0x06, 0x6a, 0x63, 0x65, 0xd5, 0x6c, 0x58, 0xd8, 0xe4, 0x12, 0x49, 0xeb, 0x7b, 0x4b, 0x4d
0x57, 0xcf, 0x26, 0xd9, 0xce, 0xf4, 0x0f, 0x27, 0x9c, 0x96, 0x5e, 0x28, 0xe8, 0x0a, 0x9a, 0xd7
```

Netlist (18 AND, 71 XOR, 2 NOT, $D = 5$):

```
xor_0 = x[0] ^ x[4]; and_0 = x[4] & x[5]; xor_1 = x[3] ^ x[5];
and_1 = x[4] & x[6]; xor_2 = x[5] ^ x[6]; xor_3 = x[2] ^ x[6];
and_2 = x[6] & x[7]; xor_4 = x[1] ^ x[7]; xor_5 = xor_0 ^ and_0;
xor_6 = xor_1 ^ xor_2; xor_7 = xor_3 ^ and_2; xor_8 = xor_4 ^ and_1;
xor_9 = x[7] ^ xor_5; and_3 = xor_5 & xor_6; xor_10 = x[4] ^ xor_6;
xor_11 = xor_6 ^ xor_5; and_4 = xor_5 & xor_7; and_5 = xor_7 & xor_6;
xor_12 = x[6] ^ xor_7; and_6 = xor_5 & xor_8; and_7 = xor_8 & xor_7;
xor_13 = xor_8 ^ xor_6; xor_14 = x[5] ^ xor_8; xor_15 = and_3 ^ and_5;
xor_16 = and_6 ^ and_4; xor_17 = and_4 ^ and_7; xor_18 = and_3 ^ xor_13;
xor_19 = xor_12 ^ xor_16; xor_20 = xor_9 ^ xor_17; xor_21 = xor_10 ^ xor_18;
xor_22 = xor_14 ^ xor_15; xor_23 = xor_19 ^ xor_5; xor_24 = xor_8 ^ xor_19;
xor_25 = xor_19 ^ xor_20; not_0 = ~xor_21; and_8 = xor_22 & xor_19;
xor_26 = xor_22 ^ xor_20; xor_27 = xor_7 ^ xor_22; xor_28 = xor_11 ^ xor_23;
xor_29 = not_0 ^ xor_20; xor_30 = xor_20 ^ xor_28; xor_31 = xor_24 ^ xor_26;
xor_32 = xor_27 ^ and_8; and_9 = xor_31 & xor_28; xor_33 = xor_5 ^ xor_31;
xor_34 = xor_29 ^ xor_25; xor_35 = xor_31 ^ xor_32; xor_36 = xor_22 ^ xor_32;
xor_37 = xor_32 ^ xor_34; and_10 = xor_28 & xor_34; xor_38 = xor_34 ^ xor_31;
xor_39 = and_9 ^ xor_37; xor_40 = xor_35 ^ and_10; xor_41 = and_9 ^ and_10;
xor_42 = xor_36 ^ xor_39; xor_43 = xor_30 ^ xor_38; xor_44 = xor_33 ^ xor_41;
xor_45 = not_0 ^ xor_40; xor_46 = xor_32 ^ xor_43; and_11 = xor_43 & xor_44;
and_12 = xor_44 & xor_31; xor_47 = xor_28 ^ xor_44; and_13 = xor_45 & xor_43;
and_14 = xor_45 & xor_31; xor_48 = xor_42 ^ xor_45; xor_49 = xor_38 ^ and_14;
xor_50 = xor_46 ^ and_12; xor_51 = xor_47 ^ and_11; xor_52 = xor_43 ^ xor_49;
xor_53 = xor_48 ^ and_13; xor_54 = xor_45 ^ xor_50; xor_55 = xor_50 ^ xor_49;
xor_56 = xor_50 ^ xor_51; xor_57 = xor_31 ^ xor_51; and_15 = xor_51 & xor_49;
xor_58 = xor_51 ^ xor_49; xor_59 = xor_44 ^ xor_53; and_16 = xor_50 & xor_53;
xor_60 = xor_51 ^ xor_53; and_17 = xor_49 & xor_53; xor_61 = xor_55 ^ and_15;
xor_62 = xor_55 ^ and_16; xor_63 = and_15 ^ xor_60; xor_64 = xor_56 ^ and_17;
xor_65 = xor_52 ^ xor_61; xor_66 = xor_59 ^ xor_62; xor_67 = xor_57 ^ xor_63;
xor_68 = xor_54 ^ xor_64; xor_69 = xor_53 ^ xor_66; not_1 = ~xor_68;
xor_70 = xor_69 ^ xor_58; y[0] = xor_58; y[1] = xor_49; y[2] = xor_70;
y[3] = not_1; y[4] = xor_65; y[5] = xor_66; y[6] = xor_67; y[7] = xor_56;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 255]$):

```
0x64, 0x28, 0x38, 0x56, 0x94, 0xab, 0xe1, 0x6e, 0x11, 0x37, 0xfd, 0xb9, 0x99, 0xb1, 0xa5, 0xf6
0x6a, 0x32, 0xea, 0xb3, 0x7a, 0xad, 0x5e, 0x61, 0xae, 0x93, 0xc1, 0x87, 0x82, 0xc9, 0x47, 0xda
0x10, 0x19, 0x1e, 0xb0, 0x12, 0x81, 0xf2, 0xf7, 0xfb, 0x7c, 0x79, 0xc0, 0x54, 0x9b, 0x57, 0x26
0x8d, 0x86, 0xd9, 0x1d, 0x6b, 0x67, 0x75, 0x55, 0x3f, 0xba, 0xb8, 0x13, 0xd4, 0x95, 0x15, 0xe0
0x4b, 0xd6, 0x7b, 0xd1, 0x4d, 0xd7, 0x27, 0xb6, 0x25, 0xeb, 0x1c, 0xee, 0xaf, 0xef, 0xc4, 0x8b
0x52, 0x73, 0xbf, 0x48, 0x31, 0x68, 0x3d, 0xf0, 0xe7, 0x14, 0x03, 0x8c, 0x0f, 0x4a, 0xfa, 0x44
0x74, 0xd8, 0x46, 0xe3, 0x49, 0xe4, 0x21, 0x08, 0xcc, 0x62, 0xe2, 0x0b, 0xe6, 0x6c, 0xc7, 0x7e
0xb2, 0x9e, 0x84, 0x5c, 0x22, 0x17, 0xc6, 0x05, 0x3e, 0x04, 0x8f, 0xed, 0x59, 0x42, 0xc8, 0xbd
0x77, 0x3c, 0x6d, 0xa9, 0xbc, 0xa8, 0x63, 0xdf, 0x6f, 0x1f, 0x3a, 0x1a, 0x0e, 0xdb, 0xdc, 0x76
0x09, 0x4e, 0xdd, 0x4f, 0x2d, 0xb4, 0xf9, 0xbe, 0xcf, 0x29, 0xfe, 0xa3, 0xf8, 0x72, 0xbb, 0xa2
0x35, 0x92, 0x51, 0xde, 0xc2, 0x7f, 0x88, 0x30, 0x01, 0xd0, 0x97, 0xc3, 0xa0, 0x53, 0x5a, 0xce
0x2f, 0xb5, 0xa7, 0x60, 0x02, 0x9a, 0x5d, 0xcb, 0x41, 0x0d, 0x65, 0x34, 0x9d, 0x36, 0x43, 0x8a
0x2c, 0xcd, 0x06, 0x66, 0x2b, 0x23, 0x85, 0x45, 0x9f, 0x89, 0xaa, 0x00, 0xb7, 0x2a, 0xf4, 0xf1
0xd2, 0x69, 0x3b, 0x78, 0x71, 0xe5, 0x39, 0xff, 0xe8, 0xf3, 0x7d, 0x2e, 0x16, 0xa6, 0x4c, 0x96
0xd3, 0xac, 0x40, 0x91, 0xe9, 0xd5, 0x07, 0x98, 0xfc, 0x8e, 0x33, 0xec, 0x58, 0xca, 0x18, 0x83
0x5b, 0x5f, 0x20, 0x70, 0xf5, 0x24, 0xc5, 0xa4, 0x0c, 0x50, 0x80, 0x90, 0x9c, 0x0a, 0xa1, 0x1b
```

Inverse netlist (18 AND, 77 XOR, 8 NOT, $D = 6$):

```
xor_0 = x[0] ^ x[5]; xor_1 = x[2] ^ x[7]; xor_2 = x[0] ^ x[1];
xor_3 = x[2] ^ xor_0; xor_4 = x[6] ^ xor_3; xor_5 = x[7] ^ xor_2;
xor_6 = x[4] ^ xor_5; xor_7 = x[4] ^ x[6]; and_0 = xor_2 & x[1];
and_1 = xor_5 & xor_3; and_2 = x[1] & xor_3; xor_8 = x[3] ^ and_2;
xor_9 = and_0 ^ xor_4; xor_10 = and_1 ^ xor_1; xor_11 = xor_8 ^ xor_2;
xor_12 = and_0 ^ xor_6; xor_13 = x[1] ^ xor_9; xor_14 = and_0 ^ xor_8;
not_0 = ~xor_11; and_3 = xor_12 & xor_10; and_4 = xor_10 & xor_9;
and_5 = not_0 & xor_12; and_6 = not_0 & xor_9; xor_15 = x[4] ^ and_6;
xor_16 = xor_8 ^ xor_15; xor_17 = and_1 ^ and_3; xor_18 = x[1] ^ xor_17;
xor_19 = and_4 ^ xor_16; xor_20 = xor_1 ^ xor_18; xor_21 = x[0] ^ xor_19;
xor_22 = and_4 ^ xor_18; xor_23 = and_3 ^ xor_1; xor_24 = xor_19 ^ xor_20;
xor_25 = x[1] ^ xor_23; xor_26 = x[0] ^ xor_20; xor_27 = and_6 ^ xor_13;
xor_28 = and_5 ^ xor_23; xor_29 = xor_10 ^ and_4; and_7 = xor_9 & xor_26;
and_8 = xor_26 & xor_27; xor_30 = x[6] ^ and_7; xor_31 = and_0 ^ and_8;
xor_32 = and_5 ^ xor_30; xor_33 = xor_30 ^ xor_0; xor_34 = x[0] ^ xor_32;
xor_35 = xor_17 ^ xor_34; xor_36 = x[7] ^ xor_33; xor_37 = and_1 ^ xor_31;
xor_38 = xor_36 ^ xor_37; xor_39 = x[5] ^ xor_31; xor_40 = and_7 ^ xor_22;
xor_41 = and_6 ^ and_8; xor_42 = x[6] ^ xor_39; xor_43 = and_8 ^ xor_15;
xor_44 = xor_22 ^ xor_42; xor_45 = xor_31 ^ xor_21; xor_46 = xor_40 ^ xor_41;
xor_47 = xor_31 ^ xor_24; xor_48 = xor_0 ^ xor_46; xor_49 = and_5 ^ xor_48;
xor_50 = and_6 ^ xor_34; xor_51 = xor_14 ^ xor_50; not_1 = ~xor_47;
not_2 = ~xor_51; not_3 = ~xor_45; and_9 = xor_38 & not_1;
and_10 = xor_38 & xor_49; and_11 = not_2 & not_3; xor_52 = and_11 ^ xor_32;
xor_53 = and_9 ^ xor_16; xor_54 = x[2] ^ xor_53; xor_55 = xor_52 ^ xor_21;
xor_56 = xor_35 ^ xor_54; xor_57 = xor_39 ^ xor_53; not_4 = ~xor_55;
and_12 = xor_38 & not_4; and_13 = not_4 & not_1; and_14 = xor_49 & not_4;
xor_58 = and_10 ^ and_12; xor_59 = xor_52 ^ xor_58; xor_60 = and_12 ^ and_14;
xor_61 = xor_31 ^ xor_59; xor_62 = xor_33 ^ xor_60; xor_63 = and_13 ^ xor_44;
xor_64 = xor_43 ^ xor_62; xor_65 = and_9 ^ xor_63; xor_66 = xor_8 ^ xor_65;
xor_67 = xor_25 ^ xor_64; xor_68 = and_5 ^ xor_60; xor_69 = xor_29 ^ xor_68;
xor_70 = xor_45 ^ xor_58; xor_71 = xor_24 ^ xor_59; not_5 = ~xor_66;
not_6 = ~xor_71; and_15 = xor_56 & not_5; and_16 = xor_56 & xor_61;
and_17 = xor_61 & xor_67; xor_72 = and_15 ^ xor_28; xor_73 = xor_57 ^ xor_72;
xor_74 = and_16 ^ xor_7; xor_75 = xor_74 ^ xor_69; xor_76 = and_17 ^ xor_70;
not_7 = ~xor_76; y[0] = xor_73; y[1] = xor_75; y[2] = not_7; y[3] = not_6;
y[4] = xor_56; y[5] = not_5; y[6] = xor_61; y[7] = xor_67;
```

C.29 8-bit AES class at $D = 3$ (8-bit, Category A)

- **Profile**: $\delta = 4$, $L = 32$ (NL = 112), $\deg(S) = 7$, $\mathcal{D}(S) = [7^{255}]$, FP = 3, SAC deviation 0.0625.
- **Cost**: 42 AND, 101 XOR, 4 NOT, AND depth $D = 3$ (unmasked NanGate 45 nm: 208.01 μm^2 / 260.7 GE, delay 1516.4 ps, logic depth 24; pipelined $d = 2$ DOM-indep: 216 `DFF_X1` [168 gadget + 48 sync], 1748.15 μm^2 / 2190.7 GE, stage delay 574.7 ps).
- **Remark**: Certified affine equivalent to the FIPS-197 AES S-box ($\sim_{\mathrm{aff}}$ AES, with $A = I_8$, $a =$ `0x00`, $B =$ [`0x08`, `0x04`, `0x02`, `0x01`, `0x80`, `0x40`, `0x20`, `0x10`], $b =$ `0x0f`); replacing the output assignments by `y[7..0] = (xor_94, not_1, not_0, xor_100, xor_98, xor_95, ~xor_99, ~xor_96)` yields the exact FIPS-197 AES lookup table at the identical gate cost (42 AND, 101 XOR, 4 NOT, $D = 3$). Depth 3

is the minimum for degree 7 (Section 3). The depth-4 circuit 33/84/4 of [BDM26b] uses nine fewer AND gates and 17 fewer XOR gates.

- **Inverse (S^{-1})**: $\deg(S^{-1}) = 7$, $\mathcal{D}(S^{-1}) = [7^{255}]$; cost: 42 AND, 106 XOR, 4 NOT, AND depth $D = 3$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 215.99 μm^2 / 270.7 GE, delay 1337.5 ps, logic depth 22; pipelined $d = 2$ DOM-indep: 216 `DFF_X1` [168 gadget + 48 sync], 1764.11 μm^2 / 2210.7 GE, stage delay 576.4 ps).

Lookup table (LUT[0 . . . 255]):

```
0x63, 0xec, 0xe1, 0xe2, 0xfb, 0x62, 0x60, 0x35, 0xcf, 0x07, 0x61, 0x42, 0xf8, 0xb1, 0x52, 0xe9
0x3a, 0x1b, 0x36, 0xe4, 0xfa, 0xa6, 0x21, 0xff, 0x54, 0xbd, 0x5b, 0x50, 0x9c, 0x5d, 0xeb, 0x3f
0xd1, 0xf4, 0x93, 0x49, 0xc9, 0xc0, 0xf1, 0x3c, 0xcd, 0x55, 0x75, 0xf7, 0xe7, 0xbe, 0xc7, 0x85
0x0d, 0x31, 0x43, 0x33, 0x8e, 0x99, 0x05, 0x9a, 0x01, 0x8b, 0x1f, 0x7b, 0x72, 0x41, 0xdb, 0xe5
0x06, 0x13, 0x4c, 0x8a, 0x82, 0x68, 0xaa, 0x5f, 0xab, 0xc2, 0xb9, 0xd3, 0x46, 0x73, 0x40, 0x1d
0xa3, 0xb7, 0x0f, 0x74, 0x4f, 0xfc, 0xd7, 0xa2, 0x6a, 0x32, 0xd8, 0xc6, 0x2a, 0x2c, 0xae, 0x30
0xbf, 0x70, 0x5a, 0xf2, 0x23, 0x24, 0xc3, 0x15, 0x25, 0xf6, 0x0b, 0xe0, 0xaf, 0xcc, 0x90, 0x5e
0xa7, 0x53, 0x2f, 0x10, 0x9b, 0x94, 0xce, 0xf5, 0xdc, 0xd9, 0xba, 0x47, 0x8f, 0xf0, 0xf3, 0xbb
0x34, 0x0c, 0x83, 0x7c, 0xa0, 0x91, 0x2d, 0x81, 0x3d, 0x51, 0xe8, 0xc4, 0x6d, 0xa4, 0x86, 0xe3
0x6f, 0x17, 0x20, 0xbc, 0x4b, 0x4a, 0x9f, 0x1e, 0x29, 0x78, 0xde, 0x8d, 0xb8, 0xa8, 0x02, 0xb2
0x7f, 0xcb, 0xca, 0x0a, 0x26, 0x09, 0x4d, 0xac, 0x3b, 0xb3, 0x5c, 0x6b, 0x97, 0x95, 0x7d, 0xe6
0x71, 0x3e, 0xc1, 0x64, 0x14, 0xb5, 0x28, 0x56, 0x6c, 0xa9, 0xfd, 0x7a, 0x65, 0xea, 0x58, 0x0e
0xda, 0xee, 0x45, 0x48, 0x8c, 0x59, 0xdd, 0x39, 0x7e, 0xb4, 0xed, 0x80, 0x22, 0xd4, 0x12, 0x1a
0xef, 0xc8, 0xd5, 0x69, 0x2e, 0x03, 0xf9, 0x08, 0x67, 0xc5, 0xa1, 0xd6, 0x19, 0x37, 0x84, 0x98
0x77, 0xfe, 0x9e, 0x87, 0x66, 0xb6, 0x18, 0x9d, 0x92, 0x88, 0x11, 0x76, 0x38, 0xa5, 0x4e, 0xb0
0x1c, 0x57, 0x16, 0x04, 0xd0, 0x79, 0x2b, 0x6e, 0x27, 0x96, 0x44, 0x00, 0xdf, 0xad, 0xd2, 0x89
```

Netlist (42 AND, 101 XOR, 4 NOT, $D = 3$):

```
xor_0 = x[0] ^ x[5]; xor_1 = x[0] ^ x[6]; xor_2 = x[2] ^ x[7];
xor_3 = x[4] ^ x[7]; xor_4 = x[3] ^ xor_0; xor_5 = x[7] ^ xor_0;
xor_6 = x[2] ^ xor_1; xor_7 = x[1] ^ xor_2; xor_8 = xor_1 ^ xor_3;
xor_9 = x[4] ^ xor_4; xor_10 = x[6] ^ xor_4; xor_11 = xor_1 ^ xor_5;
xor_12 = xor_5 ^ xor_6; and_0 = xor_8 & xor_5; xor_13 = xor_8 ^ xor_5;
xor_14 = xor_2 ^ xor_9; xor_15 = xor_7 ^ xor_9; and_1 = xor_9 & xor_6;
xor_16 = xor_9 ^ xor_6; xor_17 = xor_9 ^ xor_8; xor_18 = xor_7 ^ xor_10;
xor_19 = xor_1 ^ xor_14; xor_20 = xor_14 ^ xor_6; and_2 = xor_15 & xor_14;
and_3 = xor_18 & xor_1; xor_21 = xor_18 ^ xor_1; xor_22 = xor_18 ^ xor_15;
xor_23 = xor_18 ^ xor_8; xor_24 = x[1] ^ xor_13; and_4 = xor_17 & xor_12;
xor_25 = xor_21 ^ x[1]; xor_26 = xor_11 ^ xor_20; xor_27 = xor_24 ^ xor_16;
and_5 = xor_7 & xor_20; xor_28 = and_4 ^ and_0; xor_29 = and_4 ^ and_1;
xor_30 = xor_25 ^ xor_13; and_6 = xor_22 & xor_19; and_7 = xor_23 & xor_11;
xor_31 = xor_23 ^ xor_7; xor_32 = and_6 ^ and_3; xor_33 = and_6 ^ and_2;
and_8 = xor_31 & xor_26; xor_34 = and_8 ^ and_7; xor_35 = and_8 ^ and_5;
xor_36 = xor_33 ^ xor_34; xor_37 = xor_29 ^ xor_34; xor_38 = xor_34 ^ xor_35;
xor_39 = xor_36 ^ xor_27; xor_40 = xor_37 ^ x[1]; xor_41 = xor_32 ^ xor_38;
xor_42 = xor_28 ^ xor_38; and_9 = xor_6 & xor_39; and_10 = xor_9 & xor_39;
and_11 = xor_40 & xor_39; and_12 = xor_14 & xor_40; and_13 = xor_15 & xor_40;
xor_43 = xor_39 ^ xor_40; xor_44 = xor_41 ^ xor_30; xor_45 = xor_42 ^ xor_25;
and_14 = xor_5 & xor_44; and_15 = xor_8 & xor_44; xor_46 = xor_44 ^ xor_39;
and_16 = xor_45 & xor_44; and_17 = xor_1 & xor_45; and_18 = xor_18 & xor_45;
xor_47 = xor_45 ^ xor_40; xor_48 = xor_44 ^ xor_45; and_19 = xor_20 & xor_43;
and_20 = xor_7 & xor_43; and_21 = xor_12 & xor_46; and_22 = xor_17 & xor_46;
and_23 = xor_47 & xor_46; and_24 = xor_19 & xor_47; and_25 = xor_22 & xor_47;
xor_49 = xor_48 ^ xor_43; and_26 = xor_11 & xor_48; and_27 = xor_23 & xor_48;
xor_50 = and_21 ^ and_14; xor_51 = and_21 ^ and_9; xor_52 = and_22 ^ and_15;
xor_53 = and_22 ^ and_10; xor_54 = and_23 ^ and_16; xor_55 = and_23 ^ and_11;
xor_56 = and_24 ^ and_17; xor_57 = and_24 ^ and_12; xor_58 = and_25 ^ and_18;
xor_59 = and_25 ^ and_13; and_28 = xor_26 & xor_49; and_29 = xor_31 & xor_49;
xor_60 = xor_54 ^ xor_49; xor_61 = xor_55 ^ xor_43; xor_62 = and_28 ^ and_26;
xor_63 = and_28 ^ and_19; xor_64 = and_29 ^ and_27; xor_65 = and_29 ^ and_20;
xor_66 = xor_60 ^ xor_61; xor_67 = xor_57 ^ xor_62; xor_68 = xor_51 ^ xor_62;
xor_69 = xor_62 ^ xor_63; xor_70 = xor_59 ^ xor_64; xor_71 = xor_53 ^ xor_64;
xor_72 = xor_64 ^ xor_65; and_30 = xor_67 & xor_60; and_31 = xor_68 & xor_60;
xor_73 = xor_56 ^ xor_69; xor_74 = xor_50 ^ xor_69; and_32 = xor_70 & xor_60;
and_33 = xor_71 & xor_60; xor_75 = xor_58 ^ xor_72; xor_76 = xor_52 ^ xor_72;
and_34 = xor_73 & xor_61; xor_77 = xor_73 ^ xor_67; and_35 = xor_74 & xor_61;
xor_78 = xor_74 ^ xor_68; and_36 = xor_75 & xor_61; xor_79 = xor_75 ^ xor_70;
and_37 = xor_76 & xor_61; xor_80 = xor_76 ^ xor_71; and_38 = xor_77 & xor_66;
and_39 = xor_78 & xor_66; and_40 = xor_79 & xor_66; and_41 = xor_80 & xor_66;
xor_81 = and_38 ^ and_34; xor_82 = and_38 ^ and_30; xor_83 = and_39 ^ and_35;
xor_84 = and_39 ^ and_31; xor_85 = and_40 ^ and_36; xor_86 = and_40 ^ and_32;
xor_87 = and_41 ^ and_37; xor_88 = and_41 ^ and_33; xor_89 = xor_81 ^ xor_83;
xor_90 = xor_85 ^ xor_82; xor_91 = xor_85 ^ xor_83; xor_92 = xor_87 ^ xor_81;
xor_93 = xor_88 ^ xor_84; xor_94 = xor_86 ^ xor_88; not_0 = ~xor_89;
xor_95 = xor_84 ^ xor_90; xor_96 = xor_82 ^ xor_92; xor_97 = xor_92 ^ xor_93;
```

```
not_1 = ~xor_93; not_2 = ~xor_95; xor_98 = xor_86 ^ xor_97;
xor_99 = xor_83 ^ xor_97; not_3 = ~xor_98; xor_100 = xor_91 ^ xor_98;
y[0] = not_3; y[1] = not_2; y[2] = xor_99; y[3] = xor_96; y[4] = xor_94;
y[5] = not_1; y[6] = not_0; y[7] = xor_100;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \ldots 255]$):

```
0xfb, 0x38, 0x9e, 0xd5, 0xf3, 0x36, 0x40, 0x09, 0xd7, 0xa5, 0xa3, 0x6a, 0x81, 0x30, 0xbf, 0x52
0x73, 0xea, 0xce, 0x41, 0xb4, 0x67, 0xf2, 0x91, 0xe6, 0xdc, 0xcf, 0x11, 0xf0, 0x4f, 0x97, 0x3a
0x92, 0x16, 0xcc, 0x64, 0x65, 0x68, 0xa4, 0xf8, 0xb6, 0x98, 0x5c, 0xf6, 0x5d, 0x86, 0xd4, 0x72
0x5f, 0x31, 0x59, 0x33, 0x80, 0x07, 0x12, 0xdd, 0xec, 0xc7, 0x10, 0xa8, 0x27, 0x88, 0xb1, 0x1f
0x4e, 0x3d, 0x0b, 0x32, 0xfa, 0xc2, 0x4c, 0x7b, 0xc3, 0x23, 0x95, 0x94, 0x42, 0xa6, 0xee, 0x54
0x1b, 0x89, 0x0e, 0x71, 0x18, 0x29, 0xb7, 0xf1, 0xbe, 0xc5, 0x62, 0x1a, 0xaa, 0x1d, 0x6f, 0x47
0x06, 0x0a, 0x05, 0x00, 0xb3, 0xbc, 0xe4, 0xd8, 0x45, 0xd3, 0x58, 0xab, 0xb8, 0x8c, 0xf7, 0x90
0x61, 0xb0, 0x3c, 0x4d, 0x53, 0x2a, 0xeb, 0xe0, 0x99, 0xf5, 0xbb, 0x3b, 0x83, 0xae, 0xc8, 0xa0
0xcb, 0x87, 0x44, 0x82, 0xde, 0x2f, 0x8e, 0xe3, 0xe9, 0xff, 0x43, 0x39, 0xc4, 0x9b, 0x34, 0x7c
0x6e, 0x85, 0xe8, 0x22, 0x75, 0xad, 0xf9, 0xac, 0xdf, 0x35, 0x37, 0x74, 0x1c, 0xe7, 0xe2, 0x96
0x84, 0xda, 0x57, 0x50, 0x8d, 0xed, 0x15, 0x70, 0x9d, 0xb9, 0x46, 0x48, 0xa7, 0xfd, 0x5e, 0x6c
0xef, 0x0d, 0x9f, 0xa9, 0xc9, 0xb5, 0xe5, 0x51, 0x9c, 0x4a, 0x7a, 0x7f, 0x93, 0x19, 0x2d, 0x60
0x25, 0xb2, 0x49, 0x66, 0x8b, 0xd9, 0x5b, 0x2e, 0xd1, 0x24, 0xa2, 0xa1, 0x6d, 0x28, 0x76, 0x08
0xf4, 0x20, 0xfe, 0x4b, 0xcd, 0xd2, 0xdb, 0x56, 0x5a, 0x79, 0xc0, 0x3e, 0x78, 0xc6, 0x9a, 0xfc
0x6b, 0x02, 0x03, 0x8f, 0x13, 0x3f, 0xaf, 0x2c, 0x8a, 0x0f, 0xbd, 0x1e, 0x01, 0xca, 0xc1, 0xd0
0x7d, 0x26, 0x63, 0x7e, 0x21, 0x77, 0x69, 0x2b, 0x0c, 0xd6, 0x14, 0x04, 0x55, 0xba, 0xe1, 0x17
```

Inverse netlist (42 AND, 106 XOR, 4 NOT, $D = 3$):

```
not_0 = ~x[0]; not_1 = ~x[1]; not_2 = ~x[5]; not_3 = ~x[6];
xor_0 = not_1 ^ not_2; xor_1 = x[2] ^ x[4]; xor_2 = xor_0 ^ not_3;
xor_3 = not_0 ^ x[3]; xor_4 = not_0 ^ x[2]; xor_5 = xor_1 ^ x[7];
xor_6 = not_2 ^ not_3; xor_7 = xor_2 ^ not_0; xor_8 = not_3 ^ x[4];
xor_9 = not_2 ^ x[4]; xor_10 = xor_3 ^ not_1; xor_11 = xor_3 ^ x[7];
xor_12 = xor_4 ^ xor_6; xor_13 = xor_1 ^ xor_2; xor_14 = xor_2 ^ x[7];
xor_15 = xor_9 ^ x[7]; xor_16 = x[2] ^ x[3]; xor_17 = xor_16 ^ x[7];
xor_18 = xor_17 ^ xor_6; xor_19 = xor_1 ^ xor_7; xor_20 = not_1 ^ x[7];
xor_21 = xor_20 ^ xor_4; xor_22 = xor_0 ^ x[2]; xor_23 = xor_8 ^ not_1;
xor_24 = xor_0 ^ x[4]; xor_25 = xor_3 ^ xor_9; xor_26 = xor_4 ^ not_3;
xor_27 = xor_0 ^ x[3]; xor_28 = xor_27 ^ xor_5; xor_29 = xor_10 ^ x[4];
xor_30 = xor_10 ^ xor_5; xor_31 = xor_1 ^ xor_6; and_0 = xor_11 & xor_2;
and_1 = xor_12 & not_0; and_2 = xor_13 & xor_14; and_3 = xor_15 & xor_8;
and_4 = xor_18 & xor_7; and_5 = xor_19 & xor_21; xor_32 = and_0 ^ and_4;
xor_33 = and_1 ^ and_4; and_6 = xor_22 & xor_23; and_7 = xor_24 & xor_25;
xor_34 = and_3 ^ and_6; xor_35 = and_2 ^ and_6; and_8 = xor_26 & xor_28;
xor_36 = and_7 ^ and_8; xor_37 = and_5 ^ and_8; xor_38 = xor_35 ^ xor_36;
xor_39 = xor_33 ^ xor_36; xor_40 = xor_36 ^ xor_37; xor_41 = xor_29 ^ xor_38;
xor_42 = xor_5 ^ xor_39; xor_43 = xor_34 ^ xor_40; xor_44 = xor_32 ^ xor_40;
and_9 = xor_12 & xor_41; and_10 = not_0 & xor_41; and_11 = xor_41 & xor_42;
and_12 = xor_14 & xor_42; and_13 = xor_13 & xor_42; xor_45 = xor_41 ^ xor_42;
xor_46 = xor_30 ^ xor_43; xor_47 = xor_31 ^ xor_44; and_14 = xor_11 & xor_46;
and_15 = xor_2 & xor_46; xor_48 = xor_46 ^ xor_41; and_16 = xor_46 & xor_47;
and_17 = xor_15 & xor_47; and_18 = xor_8 & xor_47; xor_49 = xor_47 ^ xor_42;
xor_50 = xor_46 ^ xor_47; and_19 = xor_21 & xor_45; and_20 = xor_19 & xor_45;
and_21 = xor_18 & xor_48; and_22 = xor_7 & xor_48; and_23 = xor_49 & xor_48;
and_24 = xor_23 & xor_49; and_25 = xor_22 & xor_49; xor_51 = xor_50 ^ xor_45;
and_26 = xor_25 & xor_50; and_27 = xor_24 & xor_50; xor_52 = and_14 ^ and_21;
xor_53 = and_9 ^ and_21; xor_54 = and_15 ^ and_22; xor_55 = and_10 ^ and_22;
xor_56 = and_16 ^ and_23; xor_57 = and_11 ^ and_23; xor_58 = and_17 ^ and_24;
xor_59 = and_12 ^ and_24; xor_60 = and_18 ^ and_25; xor_61 = and_13 ^ and_25;
and_28 = xor_28 & xor_51; and_29 = xor_26 & xor_51; xor_62 = xor_56 ^ xor_51;
xor_63 = xor_45 ^ xor_57; xor_64 = and_26 ^ and_28; xor_65 = and_19 ^ and_28;
xor_66 = and_27 ^ and_29; xor_67 = and_20 ^ and_29; xor_68 = xor_62 ^ xor_63;
xor_69 = xor_64 ^ xor_59; xor_70 = xor_64 ^ xor_53; xor_71 = xor_64 ^ xor_65;
xor_72 = xor_61 ^ xor_66; xor_73 = xor_55 ^ xor_66; xor_74 = xor_66 ^ xor_67;
and_30 = xor_69 & xor_62; and_31 = xor_70 & xor_62; xor_75 = xor_71 ^ xor_58;
xor_76 = xor_71 ^ xor_52; and_32 = xor_72 & xor_62; and_33 = xor_73 & xor_62;
xor_77 = xor_60 ^ xor_74; xor_78 = xor_54 ^ xor_74; and_34 = xor_75 & xor_63;
xor_79 = xor_75 ^ xor_69; and_35 = xor_76 & xor_63; xor_80 = xor_76 ^ xor_70;
and_36 = xor_77 & xor_63; xor_81 = xor_77 ^ xor_72; and_37 = xor_78 & xor_63;
xor_82 = xor_78 ^ xor_73; and_38 = xor_68 & xor_79; and_39 = xor_68 & xor_80;
and_40 = xor_68 & xor_81; and_41 = xor_68 & xor_82; xor_83 = and_37 ^ and_33;
xor_84 = and_34 ^ xor_83; xor_85 = and_31 ^ xor_84; xor_86 = and_38 ^ and_35;
xor_87 = and_30 ^ and_40; xor_88 = and_39 ^ xor_85; xor_89 = and_36 ^ and_41;
xor_90 = and_39 ^ and_32; xor_91 = xor_86 ^ xor_90; xor_92 = xor_87 ^ xor_91;
xor_93 = xor_85 ^ xor_86; xor_94 = and_38 ^ xor_88; xor_95 = and_36 ^ xor_84;
xor_96 = xor_95 ^ xor_87; xor_97 = and_30 ^ xor_88; xor_98 = and_35 ^ and_33;
xor_99 = and_39 ^ xor_98; xor_100 = and_40 ^ xor_99;
xor_101 = xor_100 ^ xor_89; xor_102 = and_32 ^ and_37;
xor_103 = xor_102 ^ xor_89; xor_104 = and_31 ^ and_39;
xor_105 = xor_83 ^ xor_104; y[0] = xor_92; y[1] = xor_93; y[2] = xor_94;
y[3] = xor_96; y[4] = xor_97; y[5] = xor_101; y[6] = xor_103; y[7] = xor_105;
```

C.30 8-bit AES S-box (FIPS-197) (8-bit, Category A)

- **Profile**: $\delta = 4$, $L = 32$ (NL = 112), $\deg(S) = 7$, $\mathcal{D}(S) = [7^{255}]$, FP = 0, SAC deviation 0.0625.
- **Cost**: 36 AND, 99 XOR, 4 NOT, AND depth $D = 4$ (unmasked NanGate 45 nm: 198.44 μm^2 / 248.7 GE, delay 1148.6 ps, logic depth 21; pipelined $d = 2$ DOM-indep: 282 `DFF_X1` [144 gadget + 138 sync], 1976.38 μm^2 / 2476.7 GE, stage delay 371.0 ps).
- **Remark**: The exact FIPS-197 lookup table [Nat23]; 33/84/4 in [BDM26b]: our circuit has three more AND and 15 more XOR gates.
- **Inverse (S^{-1})**: $\deg(S^{-1}) = 7$, $\mathcal{D}(S^{-1}) = [7^{255}]$; cost: 36 AND, 95 XOR, 4 NOT, AND depth $D = 4$ (Tier 1: 3-stage affine-folded core reuse; unmasked NanGate 45 nm: 192.05 μm^2 / 240.7 GE, delay 1281.9 ps, logic depth 23; pipelined $d = 2$ DOM-indep: 276 `DFF_X1` [144 gadget + 132 sync], 1936.48 μm^2 / 2426.7 GE, stage delay 462.5 ps).

Lookup table (LUT[0 . . . 255]):

```
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
```

Netlist (36 AND, 99 XOR, 4 NOT, $D = 4$):

```
xor_0 = x[5] ^ x[7]; xor_1 = x[3] ^ x[4]; xor_2 = x[6] ^ xor_0;
xor_3 = x[0] ^ x[2]; xor_4 = xor_0 ^ xor_1; xor_5 = x[5] ^ xor_1;
xor_6 = x[0] ^ x[6]; xor_7 = x[3] ^ xor_2; xor_8 = x[2] ^ xor_2;
xor_9 = x[1] ^ xor_3; xor_10 = x[1] ^ x[2]; xor_11 = x[0] ^ xor_5;
xor_12 = x[6] ^ xor_3; xor_13 = x[4] ^ x[7]; xor_14 = xor_6 ^ xor_13;
xor_15 = x[0] ^ xor_0; xor_16 = xor_7 ^ xor_9; xor_17 = xor_4 ^ xor_9;
xor_18 = xor_3 ^ xor_4; xor_19 = xor_5 ^ xor_10; xor_20 = x[4] ^ x[6];
xor_21 = xor_1 ^ xor_8; xor_22 = x[7] ^ xor_10; xor_23 = xor_1 ^ xor_2;
and_0 = xor_11 & xor_12; and_1 = xor_14 & xor_15; and_2 = xor_7 & xor_8;
and_3 = xor_16 & xor_6; and_4 = xor_17 & xor_18; and_5 = xor_19 & xor_2;
and_6 = xor_20 & xor_21; and_7 = xor_22 & xor_23; and_8 = xor_4 & xor_1;
xor_24 = x[1] ^ and_8; xor_25 = x[2] ^ x[3]; xor_26 = x[7] ^ and_7;
xor_27 = and_5 ^ xor_25; xor_28 = and_0 ^ xor_24; xor_29 = and_4 ^ xor_24;
xor_30 = and_6 ^ xor_27; xor_31 = x[5] ^ and_1; xor_32 = xor_26 ^ xor_31;
xor_33 = and_3 ^ xor_26; xor_34 = xor_20 ^ xor_33; xor_35 = and_2 ^ and_5;
xor_36 = xor_28 ^ xor_35; xor_37 = xor_29 ^ xor_30; xor_38 = and_2 ^ xor_27;
xor_39 = xor_32 ^ xor_38; xor_40 = xor_30 ^ xor_34; xor_41 = xor_25 ^ xor_28;
xor_42 = xor_32 ^ xor_41; xor_43 = xor_29 ^ xor_34; and_9 = xor_36 & xor_37;
and_10 = xor_39 & xor_40; and_11 = xor_42 & xor_43; xor_44 = and_3 ^ xor_20;
xor_45 = and_4 ^ xor_25; xor_46 = and_0 ^ and_11; xor_47 = and_10 ^ xor_31;
xor_48 = xor_44 ^ xor_47; xor_49 = xor_45 ^ xor_46; xor_50 = and_2 ^ and_6;
xor_51 = and_9 ^ xor_50; xor_52 = xor_48 ^ xor_49; xor_53 = xor_49 ^ xor_51;
xor_54 = xor_48 ^ xor_51; and_12 = xor_37 & xor_52; and_13 = xor_36 & xor_52;
and_14 = xor_40 & xor_53; and_15 = xor_39 & xor_53; and_16 = xor_43 & xor_54;
and_17 = xor_42 & xor_54; xor_55 = and_14 ^ and_16; xor_56 = and_12 ^ and_16;
xor_57 = and_15 ^ and_17; xor_58 = and_13 ^ and_17; xor_59 = and_12 ^ and_14;
xor_60 = and_13 ^ and_15; xor_61 = xor_55 ^ xor_57; xor_62 = xor_56 ^ xor_58;
xor_63 = xor_59 ^ xor_60; and_18 = xor_15 & xor_55; and_19 = xor_14 & xor_55;
and_20 = xor_12 & xor_56; and_21 = xor_11 & xor_56; and_22 = xor_6 & xor_57;
and_23 = xor_16 & xor_57; and_24 = xor_18 & xor_58; and_25 = xor_17 & xor_58;
and_26 = xor_8 & xor_59; and_27 = xor_7 & xor_59; and_28 = xor_2 & xor_61;
and_29 = xor_19 & xor_61; and_30 = xor_21 & xor_60; and_31 = xor_20 & xor_60;
and_32 = xor_23 & xor_62; and_33 = xor_22 & xor_62; and_34 = xor_1 & xor_63;
and_35 = xor_4 & xor_63; xor_64 = and_19 ^ and_22; xor_65 = and_34 ^ xor_64;
xor_66 = and_20 ^ and_21; xor_67 = and_29 ^ and_33; xor_68 = and_18 ^ and_30;
```

```
xor_69 = and_28 ^ xor_66; xor_70 = and_24 ^ xor_67; xor_71 = and_32 ^ xor_65;
xor_72 = and_35 ^ xor_69; xor_73 = xor_65 ^ xor_68; xor_74 = and_26 ^ and_30;
xor_75 = and_31 ^ xor_74; xor_76 = and_27 ^ xor_70; xor_77 = xor_71 ^ xor_76;
not_0 = ~xor_77; xor_78 = and_33 ^ xor_72; xor_79 = xor_73 ^ xor_78;
not_1 = ~xor_79; xor_80 = and_20 ^ and_23; xor_81 = xor_70 ^ xor_75;
xor_82 = xor_80 ^ xor_81; xor_83 = and_25 ^ xor_66; xor_84 = xor_67 ^ xor_71;
xor_85 = xor_75 ^ xor_83; xor_86 = xor_84 ^ xor_85; xor_87 = and_23 ^ and_25;
xor_88 = xor_69 ^ xor_73; xor_89 = xor_87 ^ xor_88; xor_90 = and_22 ^ and_26;
xor_91 = xor_68 ^ xor_90; not_2 = ~xor_91; xor_92 = and_26 ^ and_27;
xor_93 = and_29 ^ and_34; xor_94 = xor_72 ^ xor_92; xor_95 = xor_93 ^ xor_94;
not_3 = ~xor_95; xor_96 = and_21 ^ and_25; xor_97 = and_27 ^ and_31;
xor_98 = xor_96 ^ xor_97; y[0] = not_0; y[1] = not_1; y[2] = xor_82;
y[3] = xor_86; y[4] = xor_89; y[5] = not_2; y[6] = not_3; y[7] = xor_98;
```

Inverse lookup table ($\mathrm{LUT}_{S^{-1}}[0 \dots 255]$):

```
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
```

Inverse netlist (36 AND, 95 XOR, 4 NOT, $D = 4$):

```
not_0 = ~x[0]; not_1 = ~x[1]; not_2 = ~x[5]; not_3 = ~x[6];
xor_0 = not_1 ^ not_2; xor_1 = not_3 ^ x[2]; xor_2 = x[2] ^ x[4];
xor_3 = x[3] ^ x[7]; xor_4 = xor_0 ^ not_0; xor_5 = xor_1 ^ not_2;
xor_6 = xor_0 ^ x[7]; xor_7 = not_0 ^ not_1; xor_8 = xor_3 ^ not_2;
xor_9 = not_0 ^ x[4]; xor_10 = xor_3 ^ not_1; xor_11 = not_1 ^ x[2];
xor_12 = xor_2 ^ xor_7; xor_13 = xor_1 ^ xor_6; xor_14 = not_2 ^ x[7];
xor_15 = xor_5 ^ x[3]; xor_16 = xor_1 ^ not_1; xor_17 = not_0 ^ not_2;
xor_18 = xor_0 ^ x[4]; xor_19 = xor_0 ^ xor_2; xor_20 = xor_2 ^ xor_6;
xor_21 = xor_7 ^ x[7]; xor_22 = xor_3 ^ not_3; xor_23 = xor_22 ^ xor_4;
xor_24 = xor_1 ^ xor_4; and_0 = xor_8 & xor_9; and_1 = xor_10 & xor_11;
and_2 = xor_0 & xor_12; xor_25 = and_1 ^ and_2; xor_26 = and_0 ^ and_2;
and_3 = xor_13 & not_1; and_4 = xor_14 & xor_4; and_5 = xor_15 & x[2];
and_6 = xor_16 & xor_17; and_7 = xor_18 & x[3]; xor_27 = and_3 ^ and_6;
xor_28 = and_4 ^ and_6; and_8 = xor_19 & xor_5; xor_29 = and_5 ^ and_8;
xor_30 = and_7 ^ and_8; xor_31 = xor_28 ^ xor_29; xor_32 = xor_26 ^ xor_29;
xor_33 = xor_29 ^ xor_30; xor_34 = xor_20 ^ xor_31; xor_35 = xor_21 ^ xor_32;
xor_36 = xor_27 ^ xor_33; xor_37 = xor_25 ^ xor_33; and_9 = xor_34 & xor_35;
xor_38 = xor_34 ^ xor_35; xor_39 = xor_23 ^ xor_36; xor_40 = xor_24 ^ xor_37;
xor_41 = xor_39 ^ xor_34; and_10 = xor_39 & xor_40; xor_42 = xor_40 ^ xor_35;
xor_43 = xor_39 ^ xor_40; and_11 = xor_42 & xor_41; xor_44 = xor_43 ^ xor_38;
xor_45 = and_10 ^ and_11; xor_46 = and_9 ^ and_11; xor_47 = xor_45 ^ xor_44;
xor_48 = xor_46 ^ xor_38; and_12 = xor_34 & xor_47; and_13 = xor_35 & xor_47;
and_14 = xor_39 & xor_48; xor_49 = xor_47 ^ xor_48; and_15 = xor_40 & xor_48;
and_16 = xor_41 & xor_49; and_17 = xor_42 & xor_49; xor_50 = and_14 ^ and_16;
xor_51 = and_12 ^ and_16; xor_52 = and_17 ^ and_15; xor_53 = and_17 ^ and_13;
and_18 = xor_11 & xor_50; and_19 = xor_10 & xor_50; and_20 = xor_9 & xor_51;
xor_54 = xor_50 ^ xor_51; and_21 = xor_8 & xor_51; and_22 = not_1 & xor_52;
xor_55 = xor_52 ^ xor_50; and_23 = xor_13 & xor_52; and_24 = xor_4 & xor_53;
xor_56 = xor_52 ^ xor_53; xor_57 = xor_53 ^ xor_51; and_25 = xor_14 & xor_53;
and_26 = xor_12 & xor_54; and_27 = xor_0 & xor_54; and_28 = xor_55 & x[2];
and_29 = xor_15 & xor_55; and_30 = xor_17 & xor_56; and_31 = xor_16 & xor_56;
and_32 = xor_18 & xor_57; xor_58 = xor_55 ^ xor_57; and_33 = xor_57 & x[3];
and_34 = xor_19 & xor_58; and_35 = xor_5 & xor_58; xor_59 = and_32 ^ and_33;
xor_60 = and_24 ^ and_25; xor_61 = and_18 ^ and_23; xor_62 = and_29 ^ xor_59;
xor_63 = and_19 ^ and_27; xor_64 = and_30 ^ and_28; xor_65 = xor_63 ^ xor_64;
xor_66 = and_34 ^ and_35; xor_67 = xor_60 ^ xor_66; xor_68 = and_22 ^ xor_62;
xor_69 = and_26 ^ xor_61; xor_70 = xor_65 ^ xor_68; xor_71 = and_31 ^ and_33;
xor_72 = xor_71 ^ xor_65; xor_73 = xor_72 ^ xor_67; xor_74 = and_30 ^ xor_59;
xor_75 = xor_74 ^ xor_67; xor_76 = xor_69 ^ xor_75; xor_77 = and_20 ^ and_19;
xor_78 = and_21 ^ xor_77; xor_79 = and_22 ^ xor_78; xor_80 = xor_60 ^ xor_79;
xor_81 = xor_61 ^ xor_80; xor_82 = and_24 ^ and_23; xor_83 = and_31 ^ xor_82;
xor_84 = and_34 ^ xor_83; xor_85 = xor_84 ^ xor_68; xor_86 = and_28 ^ and_31;
xor_87 = xor_86 ^ xor_62; xor_88 = xor_69 ^ xor_87; xor_89 = and_21 ^ and_27;
xor_90 = and_29 ^ xor_89; xor_91 = and_35 ^ xor_90; xor_92 = and_18 ^ and_20;
```

```
xor_93 = and_32 ^ xor_92; xor_94 = and_34 ^ xor_93; y[0] = xor_70;
y[1] = xor_73; y[2] = xor_76; y[3] = xor_81; y[4] = xor_85; y[5] = xor_88;
y[6] = xor_91; y[7] = xor_94;
```