Construction and Classification of Minimum-Latency 5-Bit APN S-Boxes *

Yaqi Qu[1], Xiutao Feng[1]†, Ruanyiyang Sun[2]

[1]Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China

[2]Duke Kunshan University, Kunshan, China

Abstract

S-boxes serve as essential nonlinear building blocks in symmetric-key cryptography, where their cryptographic properties and hardware implementation costs jointly dictate the security and efficiency of lightweight primitives. This paper studies the construction and classification of minimum-latency 5-bit APN (Almost Perfect Nonlinear) permutations under a prescribed gate-delay model, using exact minimum latencies of Boolean component functions.

Under the fixed gate-delay model of the NanGate 45 nm standard cell library, we partition the search space into input-variable permutation (IP) equivalence classes to establish a comprehensive minimum-latency database covering all 5-input balanced nonlinear Boolean functions. This database delivers exact minimum latencies and gate-level implementations for all equivalence classes up to a latency of 150 ps, while guaranteeing strict lower bounds on latency for classes exceeding that threshold, combining property filtering with circuit synthesis.

We search the five affine equivalence classes of 5-bit APN permutations. For each affine transformation of the input, we use exact component-function latencies to select components with linearly independent masks and choose their polarities, obtaining S-box circuits without repeating gate-level synthesis for each candidate. Exhaustive search with latency-preserving input-variable permutation reduction establishes a minimum latency of 126.745 ps over all 5-bit APN permutations under the prescribed gate-delay model, with explicit constructions attaining this minimum in every class. The classification results give the optimized latency distributions and numbers of optimal IP classes of input-affine transformations within each affine class.

Comparisons with selected 5-bit S-box implementations under the same model show simultaneous reductions in some cases: latency decreases by 10.20%–29.18% and normalized gate area by 5.37%–27.69%, while preserving the original S-box mappings.

Keywords: 5-bit S-boxes; APN permutations; affine equivalence; minimum-latency construction

1 Introduction

S-boxes are essential nonlinear components of symmetric-key algorithms. Their differential uniformity, nonlinearity, and algebraic degree largely determine the S-box's resistance to cryptanalytic attacks. In hardware, an S-box is often a major combinational block on the critical

*This research was supported by the CAS Project for Young Scientists in Basic Research under Grant No. YSBR-035 and National Key Research and Development Program of China (Grant No. 2026ZD1500701)

†Corresponding author: fengxt@amss.ac.cn.

path; its latency therefore affects both the maximum clock frequency and end-to-end throughput. Cryptographic properties and implementation cost must consequently be considered together when designing a new S-box or optimizing an existing one. The construction targets 5-bit APN permutations, while established 5-bit S-box mappings serve as benchmarks for evaluating the latency and area of implementations obtained by exact latency minimization for each coordinate function.

In low-latency block ciphers, both the S-box implementation and the design of the round function influence the critical path. PRINCE targets unrolled hardware and permits S-box selection from eight affine classes satisfying its cryptographic criteria, allowing the area and critical path to be adapted to the implementation technology [2]. MANTIS adopts the involutory four-bit MIDORI S-box, chosen for its small area and low circuit depth, within a reflection-based round structure [3]. QARMA explicitly incorporates critical-path-oriented S-box search, including selection among admissible PRINCE-family S-boxes and evaluation of both forward and inverse implementations [4]. These designs motivate treating the circuit realization of the nonlinear layer as part of S-box construction.

For four-bit S-boxes, Leander and Poschmann's classification identifies the affine classes with optimal differential and linear properties [5], providing a systematic starting point for implementation-oriented selection. Such cryptographic classification does not fix circuit latency: affine-equivalent mappings can have different implementation costs. Rasoolzadeh subsequently studies low-latency Boolean functions and bijective S-boxes under a NAND/NOR/INV depth metric, including the four-bit case [6]. Using the exact minimum latencies of Boolean component functions, we select components with linearly independent masks and choose their polarities to form the output coordinates to construct minimum-latency five-bit APN S-boxes and analyze latency properties across the five known affine equivalence classes under a prescribed, nonuniform gate-delay model.

Existing approaches to S-box construction can be broadly divided into three categories:

1. **Selection from known equivalence classes.** S-boxes satisfying prescribed cryptographic criteria are classified, after which concrete instances are selected from class representatives or equivalent members. Leander and Poschmann, for example, systematically classified optimal 4-bit S-boxes [5].
2. **Algebraic and structured constructions.** S-boxes are constructed from finite-field inversion and affine transformations, or from structures such as Feistel, MISTY, bridge constructions, and cellular automata [7, 8, 9, 10].
3. **Automated constraint-based search.** Cryptographic properties are formulated as SAT/SMT instances or related constraint systems, which are then solved to obtain S-boxes meeting the target criteria [11].

These approaches generate candidate S-boxes, but additional circuit-level optimization is required to assess and reduce their hardware implementation costs. Research in this direction has evolved from structured hand optimization to heuristic and solver-based techniques. Canright derived a compact implementation of the AES S-box using tower-field decomposition to optimize inversion over a finite field [12]. Boyar and Peralta proposed the Boyar–Peralta algorithm for optimizing the linear components of S-box circuits [13]. Stoffelen subsequently employed SAT solvers to optimize circuits for small S-boxes [14], and Jean et al. introduced LIGHTER, a graph-search-based framework for identifying low-cost implementations [15]. To optimize circuit

depth, Boyar et al. developed a depth-constrained method for linear-circuit optimization [16]; Jeon et al. extended the Boyar–Peralta algorithm to optimize low-depth circuits [17]; and Kim et al. proposed SLICE, a segmented search method [1].

In parallel, S-box construction from implementation-efficient Boolean functions has also been explored. Rasoolzadeh investigated the construction of bijective S-boxes from low-latency Boolean functions [6], and Jia et al. subsequently proposed an implementation method for low-latency Boolean functions and S-boxes [18]. Nevertheless, exploring a large candidate space remains computationally expensive when complete technology mapping and gate-level timing analysis must be performed separately for each candidate S-box.

We address this problem by separating S-box search from latency optimization and by evaluating an S-box through precomputed minimum latencies of its coordinate functions. For a 5-bit S-box $S = (f_0, f_1, f_2, f_3, f_4)$, given a prescribed gate set, fixed gate delays, and fixed input-arrival constraints, we define the model latency as

$$D(S) = \max_{0 \le i < 5} D(f_i), \tag{1}$$

where $D(f_i)$ denotes the minimum achievable latency of the coordinate function f_i under the specified model.

Using this formulation, the minimum latencies of scalar Boolean functions are computed offline and stored in a database indexed by input-variable permutation (IP) equivalence classes. Fixed-mapping implementation queries the five coordinate functions, whereas APN affine-class search queries all nonzero component functions and their complements to select the output coordinates. Both use the same latency records without repeating gate-level synthesis.

The resulting database covers all balanced nonlinear Boolean functions in five variables, with latencies evaluated using the fixed gate-delay model derived from the NanGate 45 nm standard-cell library. For each IP equivalence class with a minimum latency of at most 150 ps, the database provides the exact minimum latency and a corresponding gate-level implementation; for every remaining class, it certifies that the minimum latency exceeds 150 ps. This enables cryptographic-property filtering and latency evaluation to be performed jointly during S-box search. Each coordinate function is implemented separately, with only the primary-input stages shared; the gate-delay and area model is specified below.

Contributions. The main contributions are as follows.

1. **Exact scalar minimum-latency database.** Under the prescribed 20-gate model, we partition all 601,080,328 five-input balanced nonlinear Boolean functions into 5,182,316 IP classes. Complete forward construction up to 130 ps and last-gate backward matching determine exact minima for 5,182,282 classes; the remaining 34 have the strict lower bound $D(f) > 150$ ps.
2. **Property-aware queries and circuit recovery.** The database combines latency records with cryptographic properties and supports truth-table queries, ANF recovery, and gate-level circuit reconstruction in the original variable order. Four-input cross-validation gives matching delays for all 12,840 balanced nonlinear targets between direct-forward and forward–backward searches, with the recovered circuits verified on all 16 inputs.
3. **Comparison with selected S-box implementations.** With the original S-box mappings preserved, some comparisons under the same gate-delay and area model show simultaneous reductions of 10.20%–29.18% in latency and 5.37%–27.69% in normalized gate area, together

with a lower area–delay product.

4. **Construction and classification of minimum-latency 5-bit APN S-boxes.** Complete search of the five affine classes establishes 126.745 ps as the minimum latency over all 5-bit APN permutations under the prescribed gate-delay model. We provide an explicit construction attaining this minimum in every class, together with its affine parameters and gate-level circuit. The classification results give the latency distribution and the number of optimal IP classes of input-affine transformations within each affine class.

Figure 1 outlines the construction procedure. Exact Boolean-function latency computation provides latency records and lower bounds. For each input-affine transformation of an APN representative, these records are used to select output components with linearly independent masks and choose their polarities, yielding minimum-latency APN S-boxes and gate-level circuits. The class-dependent latency distributions and optimum counts are reported in Section 5.3.

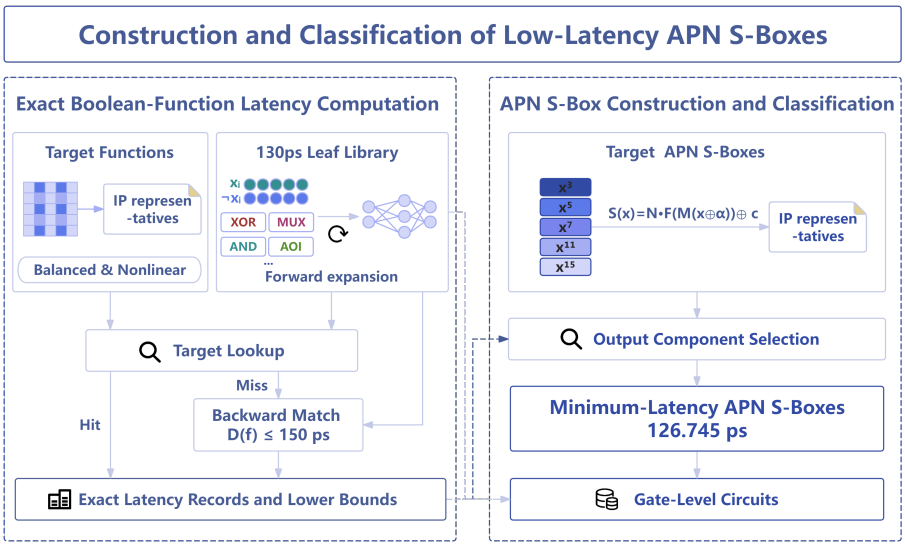

Figure 1: Minimum-latency APN S-box construction and classification: exact Boolean-function latency computation, output component selection, latency statistics, and gate-level circuits

Evaluation scope. All comparisons use the 45 nm gate delays and normalized areas reported by Rasoolzadeh [6], together with the BUF/INV input-arrival convention specified in Section 2.3. Each gate's output arrival time is its maximum input arrival time plus its fixed delay, and circuit latency is the maximum output arrival time. Area is the sum of the reachable gate-instance weights, with NAND2 normalized to one GE and shared gates counted once. Existing sharing in baseline circuits is retained; coordinate circuits obtained by exact latency minimization share only primary-input stages. The reported comparisons evaluate unmasked combinational circuits under this fixed-delay model. Load-dependent timing, fanout-dependent delays, placement, and routing are excluded.

Organization. The remainder of this paper is organized as follows. Section 2 introduces the preliminaries and timing model. Section 3 presents database construction, validation, and fixed-mapping S-box benchmarks. Section 4 presents the construction and classification of minimum-latency APN S-boxes. Section 5 reports the experimental results, and Section 6 concludes the paper. The appendices provide explicit APN constructions, complete benchmark circuits, and comparisons of individual coordinate functions.

2 Preliminaries

2.1 Notation

Let $\mathbb{F}_2 = \{0, 1\}$. For $x = (x_0, \ldots, x_4) \in \mathbb{F}_2^5$, we use the index

$$\mathrm{ind}(x) = \sum_{i=0}^{4} 2^i x_i, \tag{2}$$

so x_0 is the least significant bit and $f(x)$ is bit $\mathrm{ind}(x)$ of the 32-bit truth table.

The same bit order applies to output and translation vectors. For $R \in \mathrm{GL}(5, 2)$, the row masks are listed in output-coordinate order $0, \ldots, 4$, with

$$r_i = \sum_{j=0}^{4} 2^j R_{ij}, \qquad (Rx)_i = \bigoplus_{j=0}^{4} R_{ij} x_j.$$

Row masks and translations are written as two-digit hexadecimal values; the identity matrix is `01 02 04 08 10`. Table 1 summarizes the main notation.

Table 1: Principal notation

Symbol	Meaning
$\mathbb{F}_2^n$	n-dimensional vector space over $\mathbb{F}_2$
$\mathcal{F}(\mathbb{F}_2^n, \mathbb{F}_2^m)$	Set of functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$
$\mathrm{GL}(n, 2)$	General linear group on $\mathbb{F}_2^n$
$\mathrm{AGL}(n, 2)$	General affine group on $\mathbb{F}_2^n$
$\mathrm{wt}(u)$	Hamming weight of u
$\mathrm{ANF}(f)$	Algebraic normal form of f
$\deg(f)$	Algebraic degree of f
$\#\,\mathrm{ANF}(f)$	Number of nonzero terms in $\mathrm{ANF}(f)$
$W_f(u)$	Walsh transform of f
$\mathrm{NL}(f)$	Nonlinearity of f
$\mathrm{CI}(f)$	Correlation-immunity order of f
Δ_f	Absolute indicator of f
$D(f)$	Minimum latency of f under the fixed gate-delay model
$D(S)$	Minimum model latency of an S-box S
$D(C)$	Maximum output arrival time of a concrete circuit C (ps)
AD	Area–delay product: area of circuit C (GE) times its output arrival time $D(C)$ (ps)

For background on correlation immunity, see [20, 21]; avalanche criteria are discussed in [22].

2.2 Cryptographic Properties and Equivalence Relations

Definition 2.1 (Balanced Boolean function [23])**.** A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is balanced if

$$\mathrm{wt}(f) := \#\{x \in \mathbb{F}_2^n \mid f(x) = 1\} = 2^{n-1}.$$

The set of all balanced Boolean functions on $\mathbb{F}_2^n$ is denoted by

$$\mathcal{B}_n = \{f : \mathbb{F}_2^n \to \mathbb{F}_2 \mid \mathrm{wt}(f) = 2^{n-1}\}.$$

Definition 2.2 (General linear and affine groups [19, 23])**.** The general linear group $\mathrm{GL}(n,2)$ consists of all invertible linear mappings on $\mathbb{F}_2^n$ under composition. The general affine group $\mathrm{AGL}(n,2)$ consists of all invertible affine mappings

$$A(x) = Mx \oplus a, \qquad M \in \mathrm{GL}(n,2), \quad a \in \mathbb{F}_2^n,$$

under composition. Their orders are

$$|\,\mathrm{GL}(n,2)| = \prod_{i=0}^{n-1}(2^n - 2^i), \qquad |\,\mathrm{AGL}(n,2)| = 2^n |\,\mathrm{GL}(n,2)|.$$

Definition 2.3 (Affine group action, orbit, and stabilizer [19])**.** Let

$$\mathcal{G}_{m,n} := \mathrm{AGL}(m,2) \times \mathrm{AGL}(n,2).$$

For $F \in \mathcal{F}(\mathbb{F}_2^n, \mathbb{F}_2^m)$, the action of $\mathcal{G}_{m,n}$ is defined by

$$(B,A) \cdot F = B \circ F \circ A^{-1}, \qquad (B,A) \in \mathcal{G}_{m,n}.$$

The orbit and stabilizer of F are, respectively,

$$\mathrm{Orb}_{\mathcal{G}_{m,n}}(F) = \left\{ B \circ F \circ A^{-1} \;\middle|\; (B,A) \in \mathcal{G}_{m,n} \right\}, \tag{3}$$

$$\mathrm{Stab}_{\mathcal{G}_{m,n}}(F) = \left\{ (B,A) \in \mathcal{G}_{m,n} \;\middle|\; B \circ F \circ A^{-1} = F \right\}. \tag{4}$$

Functions in the same orbit are affine-equivalent; see also the textbook treatment of affine equivalence in [23]. By the orbit–stabilizer theorem,

$$\left|\mathrm{Orb}_{\mathcal{G}_{m,n}}(F)\right| = \frac{|\mathcal{G}_{m,n}|}{\left|\mathrm{Stab}_{\mathcal{G}_{m,n}}(F)\right|}.$$

Definition 2.4 (S-box and APN permutation [19, 23])**.** An $n \times m$ S-box is a vectorial Boolean function $S = (f_0, \ldots, f_{m-1}) : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where the f_i are its coordinate functions. If $n = m$ and S is bijective, it is called a permutation. Its differential uniformity is

$$\delta(S) = \max_{\substack{a \in \mathbb{F}_2^n \setminus \{0\} \\ b \in \mathbb{F}_2^m}} \#\{x \in \mathbb{F}_2^n \mid S(x+a) + S(x) = b\}.$$

A function with $\delta(S) = 2$ is almost perfect nonlinear (APN); a bijective APN function is an APN permutation.

For $b \in \mathbb{F}_2^m \setminus \{0\}$, let $b \cdot S = \sum_{i=0}^{m-1} b_i f_i$ denote the corresponding component function. The principal vectorial criteria are

$$\mathrm{NL}(S) = \min_{b \in \mathbb{F}_2^m \setminus \{0\}} \mathrm{NL}(b \cdot S), \tag{5}$$

$$W_{\max}(S) = \max_{\substack{b \in \mathbb{F}_2^m \setminus \{0\} \\ u \in \mathbb{F}_2^n}} |W_{b \cdot S}(u)|, \tag{6}$$

$$\deg(S) = \max_{b \in \mathbb{F}_2^m \setminus \{0\}} \deg(b \cdot S). \tag{7}$$

2.3 Gate-Level Latency Model

We adopt the NanGate 45 nm gate delays and normalized areas reported by Rasoolzadeh [6], consistently with the evaluation scope in the introduction. These parameters are specified for `_X1` cells under typical operating conditions with fan-out one. We use the tabulated delays as fixed weights; the synthesized circuits are not restricted to fan-out one.

Table 2 lists the 20 allowed internal gate types, covering common two-, three-, and four-input operations, AOI/OAI compound gates, XOR/XNOR, a multiplexer, and inversion. Each gate's output arrival time is the maximum input arrival time plus its fixed delay, and circuit latency is the maximum output arrival time. Normalized areas use NAND2 as one GE. The gate set and weights follow the existing evaluation model to keep comparisons consistent with prior work; they are not intended to represent every cell available in the NanGate library.

Table 2: Fixed gate delays and normalized areas for the NanGate 45 nm library adopted from [6]

Gate	Logic function	Delay (ps)	Area (GE)	Gate	Logic function	Delay (ps)	Area (GE)
INV	$\neg a$	22.048	0.67	NOR3	$\neg(a \vee b \vee c)$	61.543	1.33
BUF	a	33.557	1.00	AND3	$a \wedge b \wedge c$	51.869	1.67
NAND2	$\neg(a \wedge b)$	27.886	1.00	OR3	$a \vee b \vee c$	85.840	1.67
NOR2	$\neg(a \vee b)$	40.650	1.00	MUX2	$(\neg a \wedge b) \vee (a \wedge c)$	75.175	2.33
AND2	$a \wedge b$	40.171	1.33	OAI22	$\neg[(a \vee b) \wedge (c \vee d)]$	54.596	1.67
OR2	$a \vee b$	56.414	1.33	AOI22	$\neg[(a \wedge b) \vee (c \wedge d)]$	57.255	1.67
XOR2	$a \oplus b$	73.019	2.00	NAND4	$\neg(a \wedge b \wedge c \wedge d)$	44.487	1.67
XNOR2	$\neg(a \oplus b)$	57.604	2.00	NOR4	$\neg(a \vee b \vee c \vee d)$	91.313	1.67
NAND3	$\neg(a \wedge b \wedge c)$	34.767	1.33	AND4	$a \wedge b \wedge c \wedge d$	65.492	2.00
OAI21	$\neg[(a \vee b) \wedge c]$	32.651	1.33	OR4	$a \vee b \vee c \vee d$	118.592	2.00
AOI21	$\neg[(a \wedge b) \vee c]$	51.619	1.33				

Positive and complemented primary inputs are initialized through BUF and INV, respectively, giving the initial arrival times

$$t(x_i) = 33.557\,\text{ps}, \qquad t(\neg x_i) = 22.048\,\text{ps}, \qquad 0 \le i < 5. \tag{8}$$

BUF is used only for input initialization, whereas INV is also allowed internally; no zero-delay bypass from a primary input is permitted. Because the same initialization and arrival-time constraints apply to every input variable, input-variable permutations preserve model latency. At the output, f and $f + 1$ are treated as distinct synthesis and query targets, so output complementation is not assumed to be free.

Jeon et al. represent combinational S-box circuits as topologically ordered gate lists and optimize their gate count and depth [17]. To accommodate the nonuniform gate delays in our model, we instead represent scalar implementations using the recursive records defined below.

Definition 2.5 (Recursive circuit representation)**.** Let $\mathcal{R}$ be a finite collection of implementation records. Each $r \in \mathcal{R}$ stores a Boolean function F_r, represented by its truth table, and an arrival time t_r. A record is one of the following:

(i) An input record, for which $F_r \in \{x_i, \neg x_i\}$ and t_r is given by (8).

(ii) A gate record, which stores a gate G_r and an ordered tuple of child records $\mathrm{ch}(r) = (r_1, \ldots, r_k)$ such that

$$F_r = G_r(F_{r_1}, \ldots, F_{r_k}), \qquad t_r = d_{G_r} + \max_{1 \le j \le k} t_{r_j}, \tag{9}$$

where d_{G_r} is the fixed delay specified in Table 2.

A root record r_f with $F_{r_f} = f$ represents a gate-level implementation of f, obtained by recursively expanding its child records.

Definition 2.6 (Minimum function latency [17])**.** Let $\mathcal{C}(f)$ denote the finite acyclic circuits over the prescribed gate set that implement f. The minimum latency of f is

$$D(f) = \min_{C \in \mathcal{C}(f)} D(C), \tag{10}$$

where $D(C)$ is the output arrival time of the single-output circuit C. For a circuit with multiple outputs, $D(C)$ denotes the maximum output arrival time. Thus, $D(C)$ evaluates a concrete implementation, whereas $D(f)$ and $D(S)$ denote minima over valid implementations of the corresponding function or mapping.

Equation (1) is exact within the specified model. Indeed, the fan-in cone of each output implements the corresponding coordinate function, giving a lower bound of $\max_i D(f_i)$, while placing minimum-latency implementations of the five coordinate functions in parallel attains this bound.

3 Automated Minimum-Latency Search for 5-Bit Boolean Functions

This section constructs the scalar minimum-latency database using IP reduction, a complete 130 ps leaf library, and last-gate backward matching up to 150 ps. It then presents four-input cross-validation, a query and circuit-recovery example, and comparisons of latency, normalized gate area, and area–delay product for established 5-bit S-box mappings.

3.1 Target Function Set

Every nonzero component of a permutation is balanced. Moreover, for the five APN affine classes considered in this work, all nonzero components of their canonical representatives are nonlinear, and this property is invariant under affine equivalence. Accordingly, we define the target coordinate-function space as the set of all balanced nonlinear Boolean functions in five variables:

$$\mathcal{N}_5 := \{f \in \mathcal{B}_5 \mid \mathrm{NL}(f) > 0\}. \tag{11}$$

This set is obtained by excluding the affine functions from the balanced functions. Its cardinality is

$$|\mathcal{N}_5| = \binom{32}{16} - 62 = 601{,}080{,}328,$$

where $62 = 2(2^5 - 1)$ is the number of balanced affine functions in five variables. Candidate functions are represented by 32-bit truth-table integers. Their balancedness and nonlinearity are

rechecked before database insertion, providing a consistency check for both target generation and subsequent queries.

To reduce the cost of the subsequent exact minimum-latency computation, we partition the target functions into equivalence classes under the natural action of the input-variable permutation group S_5 on $\mathcal{N}_5$. For each orbit $\mathrm{Orb}_{S_5}(f) = \{f \circ \pi \mid \pi \in S_5\}$, the function with the smallest truth-table encoding is selected as the canonical representative. In this way, the 601,080,328 target functions are grouped into 5,182,316 input-permutation equivalence classes. Some functions are invariant under nonidentity permutations of their input positions; equivalently, their stabilizer satisfies $\mathrm{Stab}_{S_5}(f) \neq \{\mathrm{id}\}$; hence an orbit may have size $|\mathrm{Orb}_{S_5}(f)| = |S_5|/|\mathrm{Stab}_{S_5}(f)| < 120$. The database stores the actual orbit size of each class, and completeness of the target space is checked by summing all orbit sizes.

Under the identical input-arrival and polarity constraints of our model, an input-variable permutation merely relabels the circuit inputs and therefore preserves model latency. It therefore suffices to compute the minimum latency of each canonical representative. For any target function $f \in \mathcal{N}_5$, let r be its canonical representative and let π be the corresponding input permutation, so that $f(x) = r(\pi(x))$. Renaming the inputs of a minimum-latency gate-level circuit for r according to π yields a concrete implementation of f with the same latency. Consequently, $D(f) = D(r)$.

3.2 Construction of the Complete 130 ps Leaf Library

Threshold selection. Figure 2 shows how the leaf-library size and target coverage change as the threshold increases from 100 to 130 ps. At lower thresholds (≤ 120 ps), complete bottom-up expansion rapidly obtains all results within the threshold, but direct coverage of $\mathcal{N}_5$ grows slowly and remains below 1%. At 120 ps, for example, the coverage is only 0.268%, which is insufficient for the subsequent backward matching.

Raising the threshold to 130 ps produces a marked increase in coverage to 41.290%, corresponding to 248,188,076 target functions. At the same time, the leaf library contains 1,986,895,776 intermediate functions that can serve as inputs to the last gate, providing a sufficiently large precomputed basis for the subsequent search. Extending the threshold to 140 ps would cause the number of input combinations to exceed the available computational resources. We therefore use 130 ps as the boundary between forward precomputation and backward search.

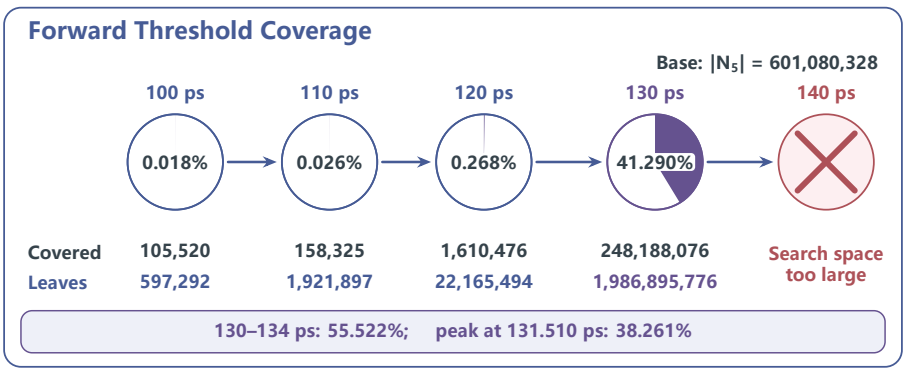

Figure 2: Leaf-library size and target coverage at different forward-expansion thresholds

The final minimum-latency statistics further support this choice: 55.522% of all target functions have minimum latencies in the range 130–134 ps, and the most populated individual

latency bin is 131.510 ps, containing 38.261% of the targets. Stopping complete forward expansion at 130 ps limits the precomputation cost. Enumerating the last output gate and matching its input subfunctions against the leaf library then covers the dominant latency range and extends exact search to 150 ps.

Complete leaf-library construction. For a given threshold τ, define the leaf library as

$$\mathcal{L}_\tau = \{(f, D(f), C_f) : D(f) \le \tau\}, \tag{12}$$

where C_f stores the root gate of an implementation of f and its connections to the input subcircuits, enabling recovery of a concrete gate-level realization. Leaf-library entries are complete subcircuits that serve as inputs to the last gate in backward matching.

Algorithm 1 gives the core procedure used to construct the leaf library. The function table is initialized with the ten primary input signals and their arrival times from (8). The algorithm then expands every legal input combination for each of the 20 logic gates. Let $D_L(f)$ denote the recorded arrival time of f, with $D_L(f) = +\infty$ if $f \notin L$. Whenever a candidate implementation of f has an arrival time smaller than $D_L(f)$, the corresponding latency and implementation record are stored in L. The process terminates when a complete pass produces no new function or latency improvement.

Algorithm 1 Iterative construction of a complete leaf library up to threshold τ

Require: Gate set $\mathcal{G}$ with fixed delays $\{d_g : g \in \mathcal{G}\}$; threshold τ
Ensure: Thresholded leaf library $\mathcal{L}_\tau$

Initialize the function table $L \leftarrow \emptyset$
for $0 \le i < 5$ **do**
    Insert $(x_i, t(x_i), \mathrm{BUF})$ and $(\neg x_i, t(\neg x_i), \mathrm{INV})$ into L
end for
repeat
    $L_{\mathrm{old}} \leftarrow L$, *updated* $\leftarrow$ false
    for all $g \in \mathcal{G}$ **do**
        for all $(f_1, \ldots, f_k) \in L_{\mathrm{old}}^k$ satisfying the input constraints of g **do**
            $d \leftarrow d_g + \max_{1 \le j \le k} D_L(f_j)$
            if $d \le \tau$ **then**
                $h \leftarrow g(f_1, \ldots, f_k)$
                if $h \notin L$ or $d < D_L(h)$ **then**
                    Update $L[h]$ with $(d, g, f_1, \ldots, f_k)$
                    updated $\leftarrow$ true
                end if
            end if
        end for
    end for
until *updated* = false
return $\mathcal{L}_\tau \leftarrow L$

Proposition 3.1 (Minimum-latency guarantee of the leaf library)**.** *Suppose that Algorithm 1 exhaustively enumerates all 20 gates and their input combinations within the threshold and iterates until the function table no longer changes. Then, for every function f satisfying $D(f) \le \tau$, the latency recorded in the leaf library is exactly $D(f)$.*

Proof. Let $\delta = \min_{g \in \mathcal{G}} d_g > 0$. Since initial arrival times are nonnegative, any circuit of latency at most τ has depth at most $\lfloor \tau/\delta \rfloor$. With finitely many initial signals and gate types of bounded fan-in, only finitely many circuit trees of this depth exist, so only finitely many latency values can be recorded. There are also finitely many Boolean functions in five variables. Each update inserts a new function or strictly decreases its recorded latency; hence, only finitely many updates can occur, and the algorithm terminates.

Let L^* be the final table, with $D_{L^*}(f) = +\infty$ for absent functions. Every recorded latency is attained by a valid circuit; hence $D_{L^*}(f) \geq D(f)$.

The final pass makes no updates, so L^* is closed under all legal gate expansions within the threshold: if $f_1, \ldots, f_k \in L^*$ form a legal input tuple for g and

$$d = d_g + \max_j D_{L^*}(f_j) \leq \tau,$$

then $g(f_1, \ldots, f_k) \in L^*$ with recorded latency at most d.

We prove $D_{L^*}(f) \leq D(f)$ for every function with $D(f) \leq \tau$ by strong induction over their finitely many distinct minimum latency values. If a minimum-latency implementation of f is an initial record, initialization gives $D_{L^*}(f) \leq D(f)$. Otherwise, let its last gate be g, with input functions f_j arriving at times a_j. Since

$$D(f_j) \leq a_j \leq D(f) - d_g < D(f),$$

the induction hypothesis gives $D_{L^*}(f_j) = D(f_j)$ for every j. Therefore,

$$d_g + \max_j D_{L^*}(f_j) \leq d_g + \max_j a_j = D(f) \leq \tau.$$

Closure yields $D_{L^*}(f) \leq D(f)$, and consequently $D_{L^*}(f) = D(f)$. □

Completeness-preserving pruning and large-scale parallelization. Suppose that the current function table contains m entries. Direct enumeration for two-, three-, and four-input gates entails $O(m^2)$, $O(m^3)$, and $O(m^4)$ input combinations, respectively, and becomes impractical in both time and memory when m reaches billions. Algorithm 1 specifies the exhaustive updates required for completeness. The implementation uses the following lossless pruning and parallelization techniques to eliminate redundant computation without excluding any minimum-latency realization within the threshold.

(1) **Latency-budget and dominance pruning.**For a gate g with fixed delay d_g, only functions with arrival times at most $\tau - d_g$ are considered as inputs. Any combination containing a later-arriving input necessarily produces an output exceeding τ and can therefore be discarded before generation. A candidate is also discarded if the global table already contains the same function with no greater arrival time. Since a gate evaluation depends only on its input truth tables and arrival times, replacing a slower realization with a faster one preserves all possible extensions and cannot increase their latency.

(2) **Incremental enumeration and input-symmetry reduction.** For each function, the program records the version in which it was last inserted or improved, together with the processed version boundary for each gate. Each iteration considers only combinations containing at least one newly inserted or improved input, thereby avoiding combinations composed entirely of previously processed states. For a two-input gate invariant under

input exchange, only unordered input pairs are enumerated. When both inputs are new or improved, their record indices assign the pair to a unique generation case. The three MUX inputs have distinct roles; if multiple positions are new or improved, the leftmost such position is designated as responsible for generation. These rules reduce duplication across iterations and input orderings without changing the set of combinations covered.

(3) **Truth-table merging for multi-input gates.** For three- and four-input AND, OR, NAND, NOR, and AOI/OAI gates, two inputs are first combined into a subexpression $s = a \wedge b$ or $s = a \vee b$, whose arrival time is the maximum of the two recorded input arrival times. If multiple input pairs produce the same 32-bit truth table s, only the pair with the smallest arrival time is retained. Substituting this pair for a slower pair leaves the gate output unchanged for every remaining input and cannot increase its latency, so the merge preserves minimum-latency results. Three-input gates are generated from one subexpression and one additional input, whereas four-input gates are generated from two subexpressions, avoiding direct enumeration of triples and quadruples. The AND-type and OR-type subexpression tables are shared among their respective gate families and filtered according to each gate's latency budget.

(4) **Batched parallel generation and radix-sort deduplication.** The combination space is partitioned along its outer index and distributed among multiple threads. The 130 ps experiment uses 56 threads and a target batch size of $2^{24} = 16{,}777{,}216$ combinations to limit temporary memory usage. Each thread writes sequentially to a private candidate array, avoiding lock contention on the global table. After each batch, the thread-local arrays are merged, and the 32-bit truth-table values are sorted using two 16-bit radix-sort passes. Only the fastest candidate for each truth table is retained and inserted into the global function table by the main thread. Batching affects only memory usage and task scheduling; because every batch is processed, it does not restrict the candidate space.

The program first forms incremental input combinations from functions inserted or improved in the current iteration. It then handles unary and binary gates, three-input gates, four-input gates, and MUX gates according to their input structures. Finally, latency-budget filtering and truth-table deduplication are applied in sequence. Only new functions or lower-latency realizations are written to the leaf library, and the current updates seed the next incremental expansion.

During construction, the global function table stores leaf records in a contiguous array. Each record contains a 32-bit function truth table, an integer latency, a root-gate identifier, and up to four child-function indices. A separate truth-table-to-array-index hash map supports average $O(1)$ duplicate detection, record access, and minimum-latency updates. The final binary leaf library consists of a fixed header followed by a contiguous record array. Each record references only its immediate subfunctions rather than storing the entire subcircuit; a concrete gate-level implementation is recovered recursively from the target record through the child indices. For $\tau = 130$ ps, construction evaluated approximately 5.184×10^{12} structurally reduced input combinations. After within-batch truth-table deduplication, approximately 4.361×10^{9} candidates were retained in total. The final table contains 1,986,895,776 distinct 5-input Boolean functions, and the resulting binary leaf-library file occupies approximately 59.2 GiB.

Construction results. For $\tau = 130$ ps, the algorithm reaches a fixed point after four expansion rounds, producing 1,986,895,776 distinct truth-table functions. Among them, 2,136,943 input-

permutation classes belong to $\mathcal{N}_5$, representing 248,188,076 individual target functions. Figure 3 illustrates how forward expansion and last-gate backward matching meet at the 130 ps leaf library, together with the number and proportion of target functions in the two parts.

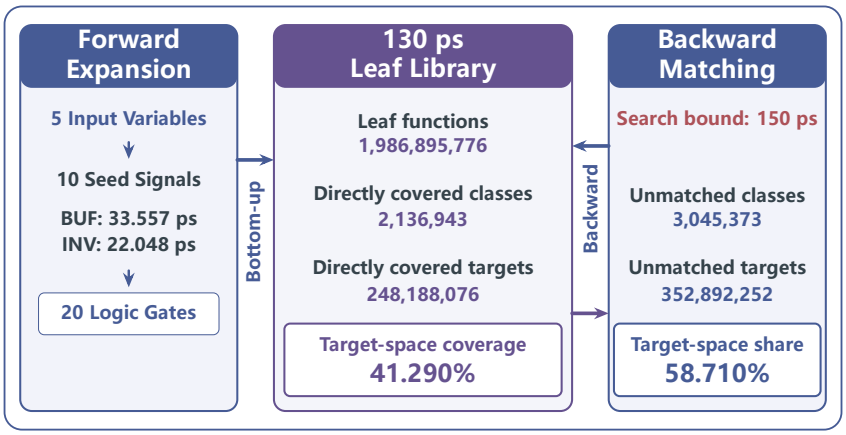

Figure 3: Interface between 130 ps forward leaf-library construction and last-gate-constrained backward matching

3.3 Exact Latency Extension to 150 ps

The 150 ps search bound. If a minimum-latency realization of a target function f ends in gate g with input subfunctions $h_1, \ldots, h_k$, then

$$f = g(h_1, \ldots, h_k), \qquad D(f) = d_g + \max_{1 \le j \le k} D(h_j). \tag{13}$$

The 130 ps leaf library contains the minimum latency of every function within its threshold. The smallest delay among the extension gates in Table 2 is $d_{\min} = 22.048\,\text{ps}$. Therefore, whenever the total-latency threshold satisfies

$$T \le 130\,\text{ps} + d_{\min} = 152.048\,\text{ps}, \tag{14}$$

every input subfunction in (13) must satisfy $D(h_j) \le 130\,\text{ps}$ and can be matched directly in the leaf library.

We set $T_{\max} = 150$ ps, strictly within the complete range established by (14). Of the 5,182,316 target classes, only 34 do not receive an exact minimum latency within this threshold. Thus, 150 ps covers almost the entire target space while avoiding the substantial computation and storage required to enlarge the forward library. For a target within the threshold, exhaustively considering all 20 possible last gates and their legal input decompositions suffices to determine its minimum latency.

Direct leaf-library hits and backward-matching results are written to the same minimum-latency master table, with sources recorded as `TAU130_LEAF_CACHE` and `BACKWARD20_TAU130_BRANCHES`, respectively. The status `EXACT` denotes a proven minimum under the gate-delay model. The remaining 34 classes have the certified lower bound $D(f) > 150$ ps, which suffices for searches below this threshold.

Backward-matching algorithm. For each canonical representative not directly covered by the 130 ps leaf library, Algorithm 2 fixes the last gate in turn and uses its truth-table constraint

to solve for input subfunctions in the leaf library. The latency of every feasible decomposition is evaluated using (13), and the smallest value found so far is retained throughout the search. If all 20 last gates have been exhausted without a decomposition at or below 150 ps, the strict lower bound is recorded.

Algorithm 2 Latency extension to 150 ps by last-gate constraints and backward matching

Require: Complete 130 ps leaf library $\mathcal{L}_{130}$; uncovered targets U; gate set $\mathcal{G}$; $T_{\max} = 150\,\text{ps}$

Ensure: Exact minimum latency or strict lower bound for every $f \in U$, together with its last-gate decomposition when found

Index leaf latencies by truth table and build latency-bounded candidate prefixes for each gate family
for all $f \in U$ in parallel **do**
  $D_{\text{best}} \leftarrow +\infty$, $P_{\text{best}} \leftarrow \varnothing$
  for all $g \in \mathcal{G}$ **do**
    $B \leftarrow \min\{T_{\max}, D_{\text{best}} - 0.001\,\text{ps}\}$
    $\mathcal{P} \leftarrow \textsc{InverseMatch}(f, g, \mathcal{L}_{130}, B - d_g)$
    for all $(h_1, \ldots, h_k) \in \mathcal{P}$ **do**
      $D_c \leftarrow d_g + \max_j D(h_j)$
      if $D_c < D_{\text{best}}$ **then**
        $(D_{\text{best}}, P_{\text{best}}) \leftarrow (D_c, (g, h_1, \ldots, h_k))$
      end if
    end for
  end for
  if $D_{\text{best}} \le T_{\max}$ **then**
    Output `EXACT`, D_{best}, and P_{best}
  else
    Output `GT_BOUND` with certified threshold $T_{\max}$
  end if
end for

Completeness-preserving reductions. The implementation generates feasible input decompositions from each gate's truth-table relation using the following completeness-preserving reductions.

(1) **Gate-family latency prefixes and dynamic bounds.** For a gate of delay d_g, only leaf records satisfying $D(h) \le B - d_g$ are retained. Whenever a faster realization is found, B is immediately tightened to $D_{\text{best}} - 0.001\,\text{ps}$, so subsequent gate families search only inputs that may improve the incumbent result.

(2) **Directly solvable last gates.** For an INV gate, the input truth table is uniquely determined as $\neg f$. For XOR and XNOR gates, once one input h is fixed, the other is uniquely determined as $f \oplus h$ and $\neg(f \oplus h)$, respectively. It is therefore sufficient to scan one latency-sorted input table and query a truth-table index for the other input.

(3) **Coverage reduction for symmetric gates.** For the AND, OR, NAND, and NOR families, the gate constraint fixes each input on either $O_f = \{x : f(x) = 1\}$ or $Z_f = \{x : f(x) = 0\}$, so only the 16 positions in the other set need to be enumerated for a balanced target. The corresponding input truth tables are queried directly in the minimum-latency leaf library. A two-input decomposition is feasible when its masks jointly cover all 16 positions; the three- and four-input cases are solved by a dominance-pruned set-cover search.

(4) **Reusable AND/OR decomposition of compound gates.** During matching, com-

pound gates are expressed through common AND and OR subexpressions. Input pairs producing the same relevant intermediate result are merged, and only the fastest realization is retained. The retained results can then be reused in compatible higher-arity combinations, reducing repeated enumeration without excluding a minimum-latency realization. For example, OAI21 is written as

$$f = \neg\big((a \vee b) \wedge c\big) = \neg(s \wedge c), \qquad s = a \vee b.$$

Candidate pairs producing the same intermediate result s share one fastest representative, after which s is combined with c and checked against the target. The other compound gates are treated analogously according to their AND/OR forms. This decomposition is used only to organize the search and reuse intermediate results; the final implementation still uses the corresponding compound standard cell and its prescribed library delay.

Parallel task partitioning and result integration. The equivalence-class representatives not covered directly by the 130 ps leaf library are divided into independent blocks of 12,800 targets, except for the final smaller block. Because the targets share only the read-only leaf library and do not depend on one another, the blocks can be executed in parallel, with the targets in each block distributed among worker threads. After all blocks have completed, their results are restored to the original target order and combined with the direct 130 ps leaf-library results to form the master table. This partitioning changes only task scheduling and resource usage; it neither restricts the search space nor affects the completeness of Algorithm 2. The resulting 150 ps database is then used to query the minimum latency and cryptographic properties of a target function f, including its algebraic degree, nonlinearity, correlation immunity, autocorrelation, and ANF term count. A concrete query and the corresponding circuit-recovery procedure are presented in Section 3.5.

3.4 Four-Input Cross-Validation

The four-input experiment uses the same 20 internal gate types, integer-femtosecond delay representation, and BUF/INV input-arrival convention as the five-input search. Its target set consists of 12,840 balanced nonlinear Boolean functions, obtained by excluding the 30 balanced affine functions from the 12,870 balanced functions. These targets form 722 IP classes. Intermediate functions are not restricted by balancedness or nonlinearity.

The comparison uses two paths. The direct-forward reference constructs the complete closure up to 150 ps without IP reduction. The forward–backward path constructs a separate 130 ps leaf library, queries the target IP representatives, and applies all 20 last-gate cases to representatives absent from the library. The bound in (14) also applies in four dimensions, so the leaf library covers every possible last-gate input of a target realizable within 150 ps. The forward closure engine is shared between the two paths; the last-gate matcher uses a separate decomposition procedure.

Table 3 shows that the leaf library directly covers 720 target classes, representing 12,832 target functions. Backward matching resolves the remaining two classes, represented by `0x2997` and `0x6897`, at 131.510 and 130.116 ps, respectively; each class has four members. After restoring the input permutations, all 12,840 target latencies agree with the direct-forward reference. A separate scalar verifier evaluates the recovered circuits on all 16 input vectors and recomputes their node arrival times in integer femtoseconds, with no failures. This experiment validates the

Table 3: Four-input cross-validation under the common fixed-delay model

Quantity	Direct forward	Forward–backward
Search bound (ps)	150	150
Forward-library threshold (ps)	150	130
Forward-library functions	65,536	65,509
Balanced nonlinear targets	12,840	12,840
Target IP classes	722	722
Leaf-hit classes	—	720
Backward-matched classes	—	2
Exact target functions	12,840	12,840

forward–backward interface, IP reduction, and circuit recovery on the complete four-input target set.

3.5 Concrete Query Example

The query program first reduces the input truth table to its canonical representative under input-variable permutations and then retrieves the representative's minimum latency, cryptographic properties, and implementation path. After restoring the scalar table, the following command queries $f = \texttt{0x0197be6a}$ without loading the leaf library:

```
$ python3 scripts/artifact.py query 0x0197be6a
truth                       0x0197be6a
canonical_representative    0x0197be6a
anf_expression              x0 ^ x1*x2 ^ x1*x3 ^ x0*x1*x3 ^ x2*x3 ^ x0*x2*x3 ^ x1*x2*x3 ^ x4 ^
    ↪  x0*x4 ^ x0*x1*x4 ^ x0*x2*x4 ^ x0*x1*x2*x4 ^ x0*x3*x4 ^ x0*x1*x3*x4 ^ x0*x2*x3*x4 ^ x1*
    ↪ x2*x3*x4
result_status               EXACT
minimum_delay_ps            139.030
orbit_size                  60
algebraic_degree            4
nonlinearity                8
walsh_max                   16
correlation_immunity        0
absolute_indicator          24
anf_terms                   16
implementation_source       BACKWARD20_TAU130_BRANCHES
implementation_path         OAI21(LEAF[0x222a33bf];LEAF[0xdc405000];LEAF[0xfffdc5d5])
```

The status `EXACT` means that 139.030 ps is the exact minimum latency of this function under the fixed gate-delay model. The source identifier `BACKWARD20_TAU130_BRANCHES` indicates that the stored implementation was obtained during precomputation by backward matching against the 130 ps leaf library. The query retrieves this stored result without repeating the backward search.

Since its minimum latency exceeds 130 ps, the target function itself is absent from the leaf library. The final gate is OAI21, with input functions `0x222a33bf`, `0xdc405000`, and `0xfffdc5d5` supplied by leaf-library records at arrival times 106.379, 100.975, and 100.975 ps, respectively.

Thus, the reconstructed circuit has output arrival time

$$32.651 + \max(106.379, 100.975, 100.975) = 139.030 \text{ ps}.$$

The complete circuit is supplied in `boolean_library/examples/query_0197be6a.txt`. Figure 4 depicts this implementation, including the expanded input subcircuits and the final OAI21 gate.

For functions without a supplied pre-exported netlist, complete circuit reconstruction requires the leaf library. With that library available, the `--netlist` option of the underlying query program expands the referenced leaf records and restores the original input-variable order.

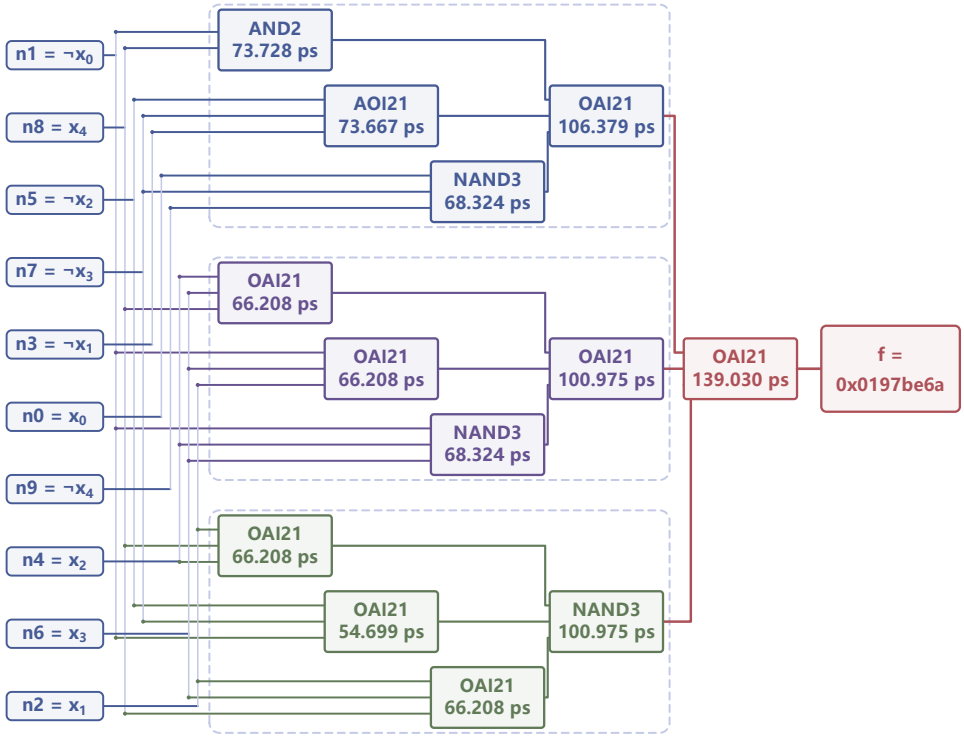

Figure 4: Concrete gate-level implementation of `0x0197be6a` recovered from the query result. Each labeled box represents a standard cell; the value in ps denotes the signal arrival time at its output, including the input-arrival and gate delays.

3.6 Comparison of 5-Bit S-Box Implementations

We evaluate implementations obtained by exact latency minimization for each coordinate function on the eight five-bit S-box mappings studied by Rasoolzadeh [6] and on Keccak and inverse Keccak from Jia et al. [18]. We also include the Ascon v1.2 S-box [24] against the reconstructed SLICE circuit of Kim et al. [1]. Keccak is compared with both source baselines, giving eleven comparisons in total. All cases preserve the original mapping and include all five outputs.

The Rasoolzadeh baselines are locally reproduced depth-oriented NAND2/NOR2 structures, using the fastest mapped realization among the available NAND2-, NOR2-, and INV-top variants for each coordinate. The Jia baselines are obtained from the public optimized RTL and mapped to the prescribed gate set while retaining their decomposition. The SLICE baseline is reconstructed

from the published gate listing. All circuits use the gate delays, normalized areas, and BUF/INV input-arrival convention of Section 2.3; their outputs are verified on all 32 inputs.

Table 4: Latency, area and area–delay product of 5-bit S-box implementations under the common model. The smaller value in each comparison is shown in bold.

	Baseline			This work		
S-box	Delay (ps)	Area (GE)	AD (GE·ps)	Delay (ps)	Area (GE)	AD (GE·ps)
Fides-5 / Primate [6]	175.953	100.95	17762.46	**133.626**	**84.18**	**11248.64**
Shamash [6]	185.685	119.96	22274.77	**131.510**	**91.83**	**12076.56**
SC2000-5 [6]	175.913	118.27	20805.23	**131.510**	**85.52**	**11246.74**
Ascon / ISAP [6]	175.725	94.97	16688.60	**140.697**	**72.60**	**10214.60**
Dry-GASCON128 [6]	168.393	91.62	15428.17	**140.697**	**72.60**	**10214.60**
Keccak [6]	96.210	**26.65**	**2564.00**	**94.870**	33.30	3159.17
ICEPOLE [6]	141.146	63.30	8934.54	**126.745**	**59.90**	**7592.03**
SYCON [6]	169.032	93.96	15882.25	**140.697**	**72.59**	**10213.20**
Keccak [18]	119.047	**23.35**	**2779.75**	**94.870**	33.30	3159.17
Inverse Keccak [18]	183.532	**40.00**	**7341.28**	**130.977**	66.55	8716.52
Ascon [1]	206.369	**34.01**	**7018.61**	**140.697**	71.28	10028.88

Table 4 reports the latency, normalized gate area, and area–delay product (AD) of the evaluated circuits. Seven of the eleven comparisons yield simultaneous improvements in all three metrics: Fides-5 / Primate, Shamash, SC2000-5, Ascon / ISAP, Dry-GASCON128, ICEPOLE, and SYCON, each compared with the corresponding Rasoolzadeh baseline. In these cases, latency reductions range from 10.20% to 29.18%, area reductions from 5.37% to 27.69%, and AD reductions from 15.03% to 45.94%. Each reduction is calculated relative to the corresponding baseline value. Shamash gives the largest latency reduction, whereas SC2000-5 gives the largest area and AD reductions.

Latency decreases in all eleven comparisons. In the remaining four comparisons—the two Keccak baselines, inverse Keccak, and the SLICE baseline—the lower latency is accompanied by higher area and AD. These results show that the latency-oriented construction can improve both delay and area, although an area improvement is not guaranteed for every mapping and baseline.

Gate-set sensitivity on Ascon. We repeat the Ascon search after removing selected gate types from the 20-gate model, with input-arrival times and the remaining delays fixed. Removing XNOR2, MUX2, AOI21, AOI22, or OAI22 individually preserves 140.697 ps. Since these restricted gate sets attain the full-set minimum, this value is also optimal for each such restriction. Removing NAND2 yields a verified implementation at 145.462 ps, while removing OAI21 yields 157.687 ps. Removing both OAI21 and OAI22, or all four AOI/OAI types, yields 160.695 ps.

All resulting circuits are verified on the 32 input vectors. The searches use gate-filtered leaf libraries; the increased latencies are achievable upper bounds, rather than proven restricted-gate minima. These tests show that some gate exclusions retain the Ascon optimum, while others produce slower implementations in the restricted search. The configurations and verification results are included in the artifact.

The two Ascon comparisons use different minimum-latency circuit realizations with different

areas. Complete circuits are given in Appendix B; comparisons of individual coordinate functions are collected in Appendix C.

4 Construction and Classification of Minimum-Latency APN S-Boxes

This section presents the construction and classification of minimum-latency 5-bit APN S-boxes under the prescribed gate-delay model. The APN requirement fixes differential uniformity at $\delta = 2$. The five affine equivalence classes established by Brinkmann and Leander [19] provide a complete search space for determining the minimum latency in each class and obtaining explicit constructions attaining it.

Input-affine transformations are enumerated modulo input-position permutations. For each representative, exact component-function latencies are used to select components with linearly independent masks and choose their polarities. The corresponding gate-level circuits are recovered from the records of Section 3, with input permutations reversed to restore the original variable order. The latency of a resulting S-box is the maximum of its coordinate minimum latencies, as defined in (1). The class-dependent latency distributions and counts of optimal IP classes are reported in Section 5.3.

The constructions are unmasked combinational implementations. Masked and threshold implementations require separate evaluation because their sharing structures, randomness requirements, and glitch-security constraints introduce additional implementation costs.

4.1 The Five APN Permutation Classes and Search Spaces

The following theorem gives representatives of the five affine classes and their inverse-class relations.

Theorem 4.1 (Brinkmann–Leander classification theorem [19])**.** *There is no APN permutation in $\mathcal{F}(\mathbb{F}_2^4, \mathbb{F}_2^4)$. In $\mathcal{F}(\mathbb{F}_2^5, \mathbb{F}_2^5)$, APN permutations form exactly five affine-equivalence classes, represented by the power functions x^{15}, x^{11}, x^7, x^3, and x^5. The class represented by x^{15} contains its inverse, the classes represented by x^{11} and x^3 are inverse to each other, and so are the classes represented by x^7 and x^5.*

The labels x^d identify the affine classes of power permutations over $\mathbb{F}_{2^5}$. In the search and the affine parameter records, F is the corresponding canonical 32-entry permutation in Table 1 of Brinkmann and Leander [19], as reproduced in the artifact. These canonical permutations are affine-equivalent to the indicated power functions.

Applying the group-order formulas above with $n = 5$ gives

$$|\operatorname{GL}(5,2)| = \prod_{i=0}^{4} \left(2^5 - 2^i\right) = 9{,}999{,}360 \approx 2^{23.25},$$
$$|\operatorname{AGL}(5,2)| = 2^5 |\operatorname{GL}(5,2)| = 319{,}979{,}520 \approx 2^{28.25}.$$

Under the two-sided affine action of $\mathcal{G}_{5,5}$, the class size is given by the orbit–stabilizer theorem as $|\mathcal{G}_{5,5}|/|\operatorname{Stab}_{\mathcal{G}_{5,5}}(F)|$, where $|\mathcal{G}_{5,5}| = |\operatorname{AGL}(5,2)|^2$. Brinkmann and Leander report EA

stabilizer orders of 155 for the EA classes represented by x^{15}, x^{11}, and x^7 [19]. Their EA action is $F \mapsto \beta \circ F \circ \alpha + \gamma$, where α is an affine bijection, β is a linear bijection, and γ is affine. The two-sided affine stabilizer embeds into this EA stabilizer: writing $B(y) = B_0(y) + b$, associate (B, A) with $(\alpha, \beta, \gamma) = (A^{-1}, B_0, b)$, where b denotes a constant function. Thus, 155 is an upper bound for the two-sided affine stabilizer orders of these three representatives. Conversely, for $F(x) = x^d$ over $\mathbb{F}_{32}$, the linear bijections

$$A_{a,i}(x) = ax^{2^i}, \qquad B_{a,i}(y) = a^d y^{2^i}, \qquad a \in \mathbb{F}_{32}^*, \quad 0 \le i < 5,$$

satisfy $B_{a,i} \circ F = F \circ A_{a,i}$ and yield $31 \cdot 5 = 155$ distinct stabilizer elements. The upper and lower bounds therefore coincide for $d \in \{15, 11, 7\}$.

For any permutation F, the identity $B \circ F = F \circ A$ is equivalent to $A \circ F^{-1} = F^{-1} \circ B$. Hence, $(B, A) \mapsto (A, B)$ gives a bijection between the two-sided affine stabilizers of F and F^{-1}. Affine-equivalent functions also have conjugate stabilizers under the defined group action. The inverse-class relations in Theorem 4.1 therefore give order 155 for x^3 and x^5 as well, as summarized in Table 5. For comparison, the larger EA action gives stabilizer orders of 4960 for x^3 and x^5 [19].

Table 5: Two-sided affine stabilizer orders of the five APN permutation representatives

Representative F	Representative of the inverse class	$\lvert \mathrm{Stab}_{\mathcal{G}_{5,5}}(F) \rvert$
x^{15}	x^{15}	155
x^{11}	x^3	155
x^7	x^5	155
x^3	x^{11}	155
x^5	x^7	155

Consequently, each affine class contains

$$\lvert \mathrm{Orb}_{\mathcal{G}_{5,5}}(F) \rvert = \frac{\lvert \mathcal{G}_{5,5} \rvert}{155} = 660{,}560{,}601{,}415{,}680 \approx 2^{49.23}$$

distinct permutations. As the five affine classes are disjoint, the total number of 5-bit APN permutations is

$$5 \cdot \lvert \mathrm{Orb}_{\mathcal{G}_{5,5}}(F) \rvert = 3{,}302{,}803{,}007{,}078{,}400 \approx 2^{51.55}.$$

4.2 Exact Latency Optimization over APN Affine Classes

Let F denote the canonical representative of an affine class in Theorem 4.1. Fix an input affine transformation $x \mapsto Mx \oplus a$, where $M \in \mathrm{GL}(5, 2)$ is invertible and $a \in \mathbb{F}_2^5$ is the input translation. Let

$$H(x) = F(Mx \oplus a), \qquad h_b(x) = b \cdot H(x), \quad b \neq 0,$$

where H is the intermediate permutation after the input transformation, b is a component mask, and $\cdot$ is the standard inner product over $\mathbb{F}_2^5$. The rows $b_0, \ldots, b_4$ of an invertible output matrix N form a basis of $\mathbb{F}_2^5$, and the coordinates of $NH(x) \oplus c$ are $h_{b_i}(x) \oplus c_i$. For each of the 31 nonzero masks, the output translation allows either polarity to be selected independently. Define the corresponding latency weight as

$$w(b) = \min\{D(h_b),\, D(h_b \oplus 1)\}.$$

The database supplies a minimum-latency implementation for each polarity, and the selected polarity determines c_i. Each weight is the minimum latency of a component or its complement, including all gate delays and the primary-input arrival times.

For fixed H, feasibility below a latency threshold T is characterized by the following rank condition.

Proposition 4.1 (Threshold-constrained output-basis construction)**.** *Suppose that five component masks $b_0, \ldots, b_4$ simultaneously satisfy*

$$w(b_i) < T \quad (0 \le i < 5), \qquad \operatorname{rank}(b_0, b_1, b_2, b_3, b_4) = 5.$$

Using these vectors as the rows of N and choosing c_i such that $D(h_{b_i} \oplus c_i) = w(b_i)$ produces an invertible output-affine transform $NH \oplus c$ with overall latency below T. Conversely, the rows of any invertible output matrix that produces latency below T satisfy both displayed conditions.

Proof. The inequalities $w(b_i) < T$ enforce the latency requirement: after choosing the corresponding c_i, every coordinate satisfies $D(h_{b_i} \oplus c_i) = w(b_i) < T$. The rank condition independently guarantees that the row matrix N is invertible. Hence $D(NH \oplus c) = \max_i D(h_{b_i} \oplus c_i) < T$. In the reverse direction, invertibility of N gives row rank 5, while $D(NH \oplus c) < T$ implies $w(b_i) \le D(h_{b_i} \oplus c_i) < T$ for every row. □

By Proposition 4.1, a feasible output basis below T exists exactly when the masks satisfying $w(b) < T$ span $\mathbb{F}_2^5$. To obtain an optimal basis for fixed H, process the 31 nonzero masks in nondecreasing order of $w(b)$ and accept a mask whenever it increases the rank of the selected set. The weight of the fifth accepted mask is the smallest maximum coordinate latency at which rank five is attained. Together with the polarity choices defining $w(b)$, this procedure determines the optimal output matrix N and constant c.

For a permutation matrix P acting on the input positions, (M, a) and (MP, a) define IP-equivalent input-affine transformations with identical optimized latency. Thus, one representative per IP class suffices. Since M is invertible, each class contains 5! matrices, leaving $|\operatorname{GL}(5,2)|/5! = 83{,}328$ matrix representatives. Including all 32 translations gives 2,666,496 IP classes per APN affine class.

Affine parameter records. Table A.2 records the five attaining constructions as

$$S(x) = NF\big(M(x \oplus \alpha)\big) \oplus c, \tag{15}$$

where $M, N \in \operatorname{GL}(5,2)$ and $\alpha, c \in \mathbb{F}_2^5$. Here $\alpha = M^{-1}a$, so this form is equivalent to the search representation $NF(Mx \oplus a) \oplus c$. The corresponding IP relation is $(M, \alpha) \sim (MP, P^{-1}\alpha)$. Matrix row masks and translation vectors use the encoding in Section 2.1.

Algorithm 3 enumerates one representative from each IP class of input-affine transformations within an APN affine class. An initial candidate supplies a latency upper bound U. The index Q_U contains exact minimum latencies for all component functions arising in the search with $D(h) \le U$, and returns $+\infty$ for other functions. This threshold-complete index suffices to identify every candidate that improves on U.

Algorithm 3 Exhaustive minimum-latency search within a 5-bit APN affine class

Require: Canonical representative F; verified upper bound U; latency index Q_U complete within U
Ensure: Exact minimum latency and its affine-transformation record

$T_{\text{best}} \leftarrow U$; store an initial candidate attaining U
for each linearly independent 5-tuple $(v_0, \ldots, v_4)$ with $1 \le \text{ind}(v_0) < \cdots < \text{ind}(v_4) \le 31$ **do**
    Construct the input matrix M from the five columns
    for $a \in \mathbb{F}_2^5$ **do**
        $H(x) \leftarrow F(Mx \oplus a)$
        for $b \in \mathbb{F}_2^5 \setminus \{0\}$ **do**
            Form $h_b = b \cdot H$ and query $Q_U(h_b)$ and $Q_U(h_b \oplus 1)$
            Record the weight $w(b)$ and its output constant bit
        end for
        In increasing weight order, select masks that increase the rank
        if a full-rank output basis is found with maximum weight below T_{best} **then**
            Update T_{best} and store the input and output affine transformations
        end if
    end for
end for
return T_{best} and the corresponding affine-transformation record

The complete execution of Algorithm 3 examines 13,332,480 IP classes of input-affine transformations across all five APN affine classes. For each representative, both polarities of the 31 nonzero component functions are queried, giving 826,613,760 component lookups. The class-wise minima and their optimality verification are reported in Section 5.3.

5 Results

5.1 Database of 5-Bit Balanced Nonlinear Boolean Functions

Table 6 summarizes the minimum-latency distribution of the balanced nonlinear Boolean functions in $\mathcal{N}_5$, with counts reported both by input-permutation class and by individual function. Of the 5,182,316 classes, 5,182,282 have exact minimum latencies, while the remaining 34 classes, comprising 797 functions, satisfy $D(f) > 150$ ps. Weighted by orbit size, 98.41% of the 601,080,328 functions have minimum latencies in $(120, 140]$ ps, including 57.39% in $(130, 140]$ ps. Only 0.03% have minimum latencies at or below 110 ps.

Table 6: Minimum-latency interval statistics for balanced nonlinear Boolean functions

Latency interval (ps)	Classes	Functions
$D(f) \le 100$	1,016	105,520
$100 < D(f) \le 110$	571	52,805
$110 < D(f) \le 120$	13,601	1,452,151
$120 < D(f) \le 130$	2,121,755	246,577,600
$130 < D(f) \le 140$	2,971,264	344,954,053
$140 < D(f) \le 150$	74,075	7,937,402
$D(f) > 150$	34	797

Figure 5 shows the class counts at the 104 distinct exact minimum-latency values on a logarithmic scale. The distribution is concentrated around 130 ps and peaks at 131.510 ps. The exact minimum latencies range from 82.585 to 147.887 ps, with relatively few classes near either endpoint.

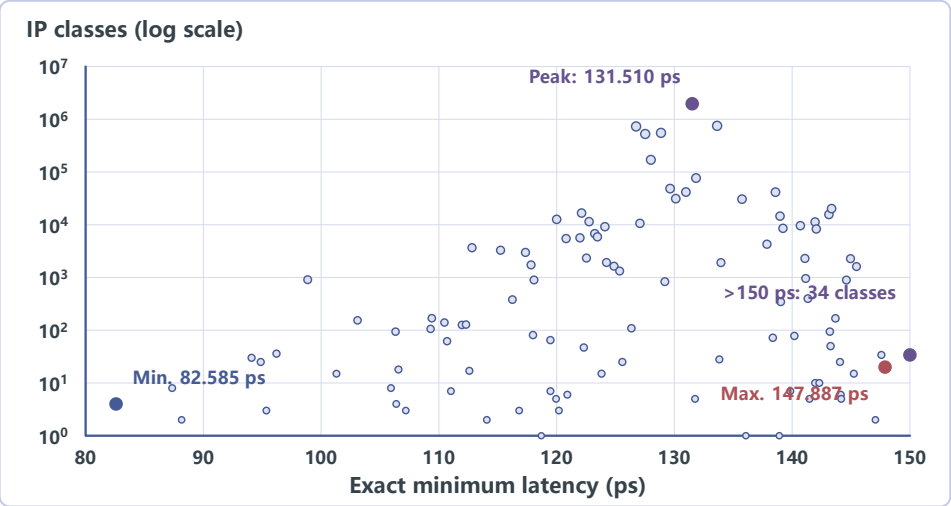

Figure 5: Fine-grained minimum-latency distribution of the input-permutation classes of balanced nonlinear Boolean functions

Figure 2 reports the leaf-library size and target coverage at different forward-expansion thresholds. At 130 ps, the library contains 1,986,895,776 intermediate functions and occupies approximately 59.2 GiB, storing truth tables, minimum latencies, and circuit connections. It directly covers 248,188,076 target functions, corresponding to 41.290% of $\mathcal{N}_5$. The remaining library entries include functions outside the balanced nonlinear target set that can serve as subfunctions in backward matching. Thus, the library size counts all available intermediate functions, whereas Table 6 reports only the target functions and their input-permutation classes.

5.2 Latency Statistics by Algebraic Degree and Nonlinearity

Table 7 groups the balanced nonlinear Boolean functions by algebraic degree and nonlinearity. It reports the numbers of input-permutation classes and individual functions, the range of exact minimum latencies, and the number of classes with $D(f) > 150$ ps. The reported maxima are taken over exact records only.

Degree-four functions account for 96.87% of the target set. The largest group, with degree four and nonlinearity 10, contains 375,753,728 functions (62.51% of the total), whose minimum latencies range from 101.302 to 147.578 ps. All groups with nonlinearity at least 6 have exact records within 150 ps. The 34 classes exceeding this threshold are confined to $(\deg, \mathrm{NL}) = (3, 4)$, $(4, 2)$, and $(4, 4)$, with 4, 24, and 6 classes, respectively.

The minimum achievable latency does not vary monotonically with algebraic degree. Among functions with nonlinearity 12, the smallest latencies are 122.756, 110.695, and 122.117 ps for degrees two, three, and four, respectively. Substantial variation also remains within a fixed property group: the degree-four, nonlinearity-eight group spans 94.094–145.462 ps. These results show that algebraic degree and nonlinearity do not determine implementation latency under the

Table 7: Joint cryptographic-property and latency statistics for the nonlinear-function master table

Degree	NL	Classes	Functions	Exact min. (ps)	Exact max. (ps)	$D > 150$ (classes)
2	8	202	8,680	82.585	140.697	0
2	12	354	27,776	122.756	143.346	0
3	4	988	59,520	87.350	147.887	4
3	8	107,496	11,700,640	82.585	145.462	0
3	12	59,936	6,999,552	110.695	143.122	0
4	2	342	15,872	98.859	147.887	24
4	4	9,432	833,280	87.350	147.887	6
4	6	181,044	19,443,200	87.350	145.462	0
4	8	1,612,224	184,571,520	94.094	145.462	0
4	10	3,195,642	375,753,728	101.302	147.578	0
4	12	14,656	1,666,560	122.117	133.626	0

prescribed model. Database lookup therefore permits latency-based selection among functions satisfying the same cryptographic-property constraints.

5.3 Minimum-Latency Constructions and Classification Results

We apply the exact latency optimization method of Section 4 to all five affine equivalence classes of 5-bit APN permutations. For each input-affine representative, exact component-function latencies determine the selection of components with linearly independent masks and their polarities, producing an implementation without repeating gate-level synthesis. After eliminating latency-equivalent input-variable permutations, the search covers 2,666,496 IP classes of input-affine transformations per APN affine class and 13,332,480 in total.

All five classes attain a minimum latency of 126.745 ps. A complete scan at the immediately preceding threshold, 126.744 ps, finds no feasible output basis in any class. Since delays are represented in integer femtoseconds and the scalar database is complete within this threshold, these results establish 126.745 ps as the global minimum over all 5-bit APN permutations under the prescribed model. Appendix A provides an attaining construction from each class, together with its affine parameters and complete gate-level circuit.

Observations on the common minimum. The output coordinates require five linearly independent component masks. Over the complete search, the maximum rank attainable using components with latency strictly below 126.745 ps is four in the x^3 and x^5 classes and two in the remaining classes. At 126.745 ps, every class admits a full-rank output basis. The common minimum therefore corresponds to the first latency threshold at which five independent output components can be selected in each class. Since S-box latency is the maximum of the five coordinate minimum latencies, an attaining implementation may contain coordinates with lower delays.

The recovered circuits attain this threshold through the following verified critical-path gate

sequences:

$$\text{BUF} \to \text{NAND2} \to \text{OAI21} \to \text{OAI21},$$
$$\text{BUF} \to \text{OAI21} \to \text{NAND2} \to \text{OAI21},$$
$$\text{BUF} \to \text{OAI21} \to \text{OAI21} \to \text{NAND2}.$$

Each sequence contains one BUF, one NAND2, and two OAI21 gates, giving a total delay of $33.557 + 27.886 + 2(32.651) = 126.745$ ps. Thus, the minimum is attained by circuits with different gate orders and topologies. These observations describe how the common minimum is attained. A structural explanation of why all five affine classes first admit a full-rank output basis at the same threshold remains an open problem.

Latency distributions. Table 8 summarizes the exact optimized latencies of $N = 2{,}666{,}496$ IP classes of input-affine transformations within each APN affine class. Each IP class is counted once, with its latency minimized over the output basis and constant. P5 and P95 denote the 5th and 95th percentiles, and Q1 and Q3 the 25th and 75th percentiles. Percentiles use linear interpolation at position $(N-1)p$ in the sorted latencies, indexed from zero. *Optimal IP classes* gives the number of IP classes attaining 126.745 ps.

Table 8: Optimized latency distributions over IP classes within each APN affine class (ps)

Class	Min.	P5	Q1	Median	Q3	P95	Max.	Optimal IP classes
x^3	126.745	126.745	126.745	127.521	127.521	128.000	128.861	691,269
x^5	126.745	126.745	126.745	127.521	127.521	128.000	130.977	713,837
x^7	126.745	126.745	127.521	127.521	127.521	128.861	131.510	263,872
x^{11}	126.745	126.745	127.521	127.521	127.521	128.861	131.510	255,781
x^{15}	126.745	126.745	127.521	127.521	128.000	128.861	131.510	169,632

All five affine classes have a median latency of 127.521 ps. P95 is 128.000 ps for x^3 and x^5 and 128.861 ps for the other three classes. For x^7 and x^{11}, Q1 and Q3 both equal the median, showing a concentration at 127.521 ps.

The proportions of optimal IP classes are 25.92%, 26.77%, 9.90%, 9.59%, and 6.36% for x^3, x^5, x^7, x^{11}, and x^{15}, respectively. The two quadratic affine classes therefore contain higher proportions of optimal IP classes. The maximum latency is 128.861 ps for x^3, 130.977 ps for x^5, and 131.510 ps for the other three classes.

Comparison with canonical representatives. Table 9 compares the class-wise minima with the latencies of the canonical representatives. The constructions reduce latency by 3.62%–5.15% while preserving the affine-invariant cryptographic properties listed in the table. In particular, the x^{15} class attains the same minimum as the other four classes despite its higher algebraic degree and lower nonlinearity.

6 Conclusion

We constructed a minimum-latency database for five-input balanced nonlinear Boolean functions, covering 5,182,316 IP classes. Exact minimum latencies are established for 5,182,282 classes; the

Table 9: Class-wise minimum latencies and reductions relative to the canonical APN representatives

Class	Degree	NL(S)	$W_{\max}(S)$	$\delta(S)$	Canonical (ps)	Minimum (ps)	Reduction (%)
x^3	2	12	8	2	133.626	126.745	5.15%
x^5	2	12	8	2	131.510	126.745	3.62%
x^7	3	12	8	2	131.510	126.745	3.62%
x^{11}	3	12	8	2	133.626	126.745	5.15%
x^{15}	4	10	12	2	133.626	126.745	5.15%

remaining 34 satisfy $D(f) > 150$ ps. For selected 5-bit S-box mappings, some comparisons show simultaneous reductions of 10.20%–29.18% in latency and 5.37%–27.69% in normalized gate area while preserving the original mappings under the same gate-delay and area model. Complete search of the five APN affine classes establishes 126.745 ps as the global minimum latency for 5-bit APN permutations under the prescribed model, with explicit attaining constructions in every class. The classification results give the optimized latency distributions and numbers of optimal IP classes within each affine class, revealing differences despite the shared minimum.

The optimality results concern the prescribed fixed gate-delay model. A structural explanation of the common minimum across the five APN affine classes remains an open problem. Future work also includes evaluation with load-dependent timing and physical implementation flows, and adaptation to other cell libraries. Extending the search to six-input functions will require stronger equivalence reduction, pruning, and lower bounds to control the larger search space.

Artifact Availability

The artifact includes source code, the complete scalar latency table, APN search tools and results, affine parameters, and standalone gate-level circuits for the constructions and comparisons reported in this paper. Instructions and checksum manifests accompany the files.

Latency queries, complete APN affine-class searches, statistical analysis, and verification of the supplied circuits do not require the intermediate leaf library.

The artifact repository is maintained at `https://github.com/Syndeere/low_latency_of_5bit_boolean`.

Acknowledgments

We thank the team participating in the 2026 National College Cryptographic Mathematics Challenge for making their search code publicly available at `https://github.com/xieh-Nox/2026-Challenge-Problem-2`. Parts of their leaf-library construction and branch-matching implementation provided a useful reference for our implementation.

Disclosure of AI assistance. Generative AI tools, including OpenAI Codex, were used to assist with code development and manuscript revision. The research methodology, experimental design, execution of the experiments, and interpretation of the results were carried out by the authors. The authors retain full responsibility for the code, the reported results, and the final

manuscript.

References

[1] Kim G, Baek S, Jeon Y, Hadžić V, Kim J. Bitsliced Segment-Based Search Technique for Low-Depth and Hardware-Efficient S-Box Circuits[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2026(3): 1152–1176. doi:10.46586/tches.v2026.i3.1152-1176.

[2] Borghoff J, Canteaut A, Güneysu T, et al. PRINCE—A Low-Latency Block Cipher for Pervasive Computing Applications[C]// Advances in Cryptology—ASIACRYPT 2012, Lecture Notes in Computer Science, 2012, 7658: 208–225. doi:10.1007/978-3-642-34961-4_14.

[3] Beierle C, Jean J, Kölbl S, et al. The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS[C]// Advances in Cryptology—CRYPTO 2016, Lecture Notes in Computer Science, 2016, 9815: 123–153. doi:10.1007/978-3-662-53008-5_5.

[4] Avanzi R. The QARMA Block Cipher Family[J]. IACR Transactions on Symmetric Cryptology, 2017(1): 4–44. doi:10.13154/tosc.v2017.i1.4-44.

[5] Leander G, Poschmann A. On the Classification of 4 Bit S-Boxes[C]// WAIFI 2007, Lecture Notes in Computer Science, 2007, 4547: 159–176. doi:10.1007/978-3-540-73074-3_13.

[6] Rasoolzadeh S. Low-Latency Boolean Functions and Bijective S-boxes[J]. IACR Transactions on Symmetric Cryptology, 2022(3): 403–447. doi:10.46586/tosc.v2022.i3.403-447.

[7] Daemen J, Rijmen V. The Design of Rijndael: AES–The Advanced Encryption Standard[M]. Berlin: Springer, 2002. doi:10.1007/978-3-662-04722-4.

[8] Canteaut A, Duval S, Leurent G. Construction of Lightweight S-Boxes using Feistel and MISTY Structures[R]. IACR Cryptology ePrint Archive, Report 2015/711, 2015. `https://eprint.iacr.org/2015/711`.

[9] Tian S, Liu Y, Zeng X. A further study on bridge structures and constructing bijective S-boxes for low-latency masking[J]. Designs, Codes and Cryptography, 2023, 91(11): 3709–3739. doi:10.1007/s10623-023-01266-w.

[10] Mariot L, Picek S, Leporati A, Jakobovic D. Cellular automata based S-boxes[J]. Cryptography and Communications, 2019, 11(1): 41–62. doi:10.1007/s12095-018-0311-8.

[11] Lu Z, Mesnager S, Cui T, Fan Y, Wang M. An STP-based model toward designing S-boxes with good cryptographic properties[R]. IACR Cryptology ePrint Archive, Report 2023/1023, 2023. `https://eprint.iacr.org/2023/1023`.

[12] Canright D. A Very Compact S-Box for AES[C]// Cryptographic Hardware and Embedded Systems—CHES 2005, Lecture Notes in Computer Science, 2005, 3659: 441–455. doi:10.1007/11545262_32.

[13] Boyar J, Peralta R. A New Combinational Logic Minimization Technique with Applications to Cryptology[C]// Experimental Algorithms, Lecture Notes in Computer Science, 2010, 6049: 178–189. doi:10.1007/978-3-642-13193-6_16.

[14] Stoffelen K. Optimizing S-Box Implementations for Several Criteria Using SAT Solvers[C]// Fast Software Encryption—FSE 2016, Lecture Notes in Computer Science, 2016, 9783: 140–160. doi:10.1007/978-3-662-52993-5_8.

[15] Jean J, Peyrin T, Sim S M, Tourteaux J. Optimizing Implementations of Lightweight Building Blocks[J]. IACR Transactions on Symmetric Cryptology, 2017(4): 130–168. doi:10.13154/tosc.v2017.i4.130-168.

[16] Boyar J, Find M G, Peralta R. Small Low-Depth Circuits for Cryptographic Applications[J]. Cryptography and Communications, 2019, 11(1): 109–127. doi:10.1007/s12095-018-0296-3.

[17] Jeon Y, Baek S, Kim G, Kim J. A Framework for Generating S-Box Circuits with Boyar–Peralta Algorithm-Based Heuristics, and Its Applications to AES, SNOW3G, and Saturnin[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2025(1): 586–631. doi:10.46586/tches.v2025.i1.586-631.

[18] Jia C, Cui T, Fan Y, Wei P, Wang M. A New Optimized Implementation Method for Boolean Functions Towards Low-Latency: Application to S-Boxes[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2026(3): 147–170. doi:10.46586/tches.v2026.i3.147-170.

[19] Brinkmann M, Leander G. On the classification of APN functions up to dimension five[J]. Designs, Codes and Cryptography, 2008, 49(1–3): 273–288. doi:10.1007/s10623-008-9194-6.

[20] Siegenthaler T. Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications[J]. IEEE Transactions on Information Theory, 1984, 30(5): 776–780. doi:10.1109/TIT.1984.1056949.

[21] Xiao G Z, Massey J L. A Spectral Characterization of Correlation-Immune Combining Functions[J]. IEEE Transactions on Information Theory, 1988, 34(3): 569–571. doi:10.1109/18.6037.

[22] Zhang X M, Zheng Y. GAC—The Criterion for Global Avalanche Characteristics of Cryptographic Functions[J]. Journal of Universal Computer Science, 1995, 1(5): 320–337. doi:10.3217/jucs-001-05-0320.

[23] Carlet C. Boolean Functions for Cryptography and Coding Theory[M]. Cambridge: Cambridge University Press, 2021. doi:10.1017/9781108606806.

[24] Dobraunig C, Eichlseder M, Mendel F, Schläffer M. Ascon v1.2: Lightweight Authenticated Encryption and Hashing[J]. Journal of Cryptology, 2021, 34(3): 33. doi:10.1007/s00145-021-09398-9.

A Optimal APN Permutations and Their Implementations

A.1 Permutation Representations

Table A.1 gives the 32-entry permutation representations of the five candidates. Each output is encoded as a two-digit hexadecimal value.

Table A.1: 32-entry representations of the five low-latency candidates

Affine class	Output sequence
x^3	02 08 1b 15 1e 16 0f 03 01 12 0a 1d 05 14 06 13
	11 1c 0d 04 18 17 0c 07 0e 1a 00 10 1f 09 19 0b
x^5	01 14 08 17 19 00 15 06 1f 03 12 04 1b 0b 13 09
	10 1d 0a 0d 0e 0f 11 1a 1c 18 02 0c 1e 16 05 07
x^7	11 15 10 01 14 0f 1b 09 05 0d 07 12 06 1e 17 0e
	00 0c 0a 19 02 03 1f 1d 08 16 1a 13 04 18 1c 0b
x^{11}	18 1d 0b 09 02 11 03 16 12 1a 04 19 1b 1f 07 17
	15 13 10 08 01 0f 14 05 00 1e 0e 1c 0a 06 0c 0d
x^{15}	08 18 06 10 05 02 0a 00 0c 16 1f 0d 13 12 19 11
	0e 1b 0f 09 07 03 1a 14 01 04 15 1c 0b 17 1d 1e

A.2 Affine Transformation Parameters

Table A.2 gives the affine parameters for reproducing the permutations in Table A.1 using (15) and the encoding specified in Section 4.2.

Table A.2: Affine parameters of the five low-latency APN candidates relative to their canonical representatives

Affine class	Input row masks M	α	Output row masks N	c
x^3	0e 17 14 04 16	0d	0a 0d 08 1c 1a	14
x^5	17 1c 1b 16 0e	11	19 13 1f 0d 02	1d
x^7	01 02 04 08 10	00	02 08 09 18 1c	11
x^{11}	0e 03 12 05 16	11	19 0e 09 0f 1c	13
x^{15}	19 1d 1e 08 13	10	18 11 12 0f 01	0e

A.3 Gate-Level Implementations

The listings below implement the fixed optimal permutations in Table A.1, with affine parameters given in Table A.2. Gate definitions and pin order follow Table 2.

A.3.1 Class x^3

This circuit obtained by exact latency minimization for each coordinate function realizes the x^3-class permutation in Table A.1. Input and output indices run from the least significant bit. The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps.

Listing A.1. Complete optimal APN circuit in the class x^3

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(p2, p3, p4);
t1 = OAI21(q1, p4, p0);
t2 = NAND2(t0, t1);
t3 = NAND2(p3, p1);
t4 = OAI21(q2, q4, q0);
t5 = NAND3(q3, q4, q1);
t6 = OAI21(t3, t4, t5);
t7 = NAND2(q0, q3);
t8 = NAND3(p4, t7, p2);
f0 = OAI21(t2, t6, t8);
// Coordinate f1
t9 = NAND2(p2, p0);
t10 = NAND2(q1, p3);
t11 = OAI21(q2, q0, q3);
t12 = OAI21(t9, t10, t11);
t13 = OAI21(p1, q3, q2);
t14 = OAI21(p1, q4, q0);
t15 = NAND2(t13, t14);
t16 = AND2(q0, q4);
t17 = NAND2(t10, t16);
f1 = OAI21(t12, t15, t17);
// Coordinate f2
t18 = OAI21(p2, q0, p1);
t19 = NAND3(q2, q1, q4);
t20 = NAND2(t18, t19);
t21 = NAND2(p4, p3);
t22 = NAND3(q0, q1, q3);
t23 = OAI21(q0, t21, t22);
t24 = OAI21(q2, p0, q4);
t25 = OAI21(q1, p3, p4);
t26 = NAND2(t24, t25);
f2 = OAI21(t20, t23, t26);
// Coordinate f3
t27 = OAI21(p4, q1, p3);
t28 = OAI21(p3, p0, p2);
t29 = NAND2(t27, t28);
t30 = NAND2(p1, p0);
t31 = NAND3(q2, q0, q1);
t32 = OAI21(p3, t30, t31);
t33 = NAND2(q2, p1);
t34 = NOR2(q3, q4);
t35 = NAND2(t33, t34);
f3 = OAI21(t29, t32, t35);
// Coordinate f4
t36 = NAND2(p3, p0);
t37 = NAND2(p2, p4);
t38 = NAND2(q3, p1);
t39 = OAI21(t36, t37, t38);
t40 = OAI21(q2, q4, q0);
t41 = NAND3(q3, q4, q2);
t42 = OAI21(q3, t40, t41);
t43 = NAND4(q3, q2, p1, q4);
f4 = OAI21(t39, t42, t43);
```

A.3.2 Class x^5

This circuit obtained by exact latency minimization for each coordinate function realizes the x^5-class permutation in Table A.1. Input and output indices run from the least significant bit. The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps.

Listing A.2. Complete optimal APN circuit in the class x^5

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p2, p1);
t1 = NAND2(q3, p0);
t2 = OAI21(q2, q1, q0);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(p3, q0, q2);
t5 = OAI21(p3, q4, q1);
t6 = NAND2(t4, t5);
t7 = AND2(q1, q4);
t8 = NAND2(t1, t7);
f0 = OAI21(t3, t6, t8);
// Coordinate f1
t9 = OAI21(p2, p0, p4);
t10 = OAI21(p4, q3, q1);
t11 = NAND2(t9, t10);
t12 = NAND2(p3, p1);
t13 = NAND3(q4, q0, q3);
t14 = OAI21(q0, t12, t13);
t15 = NAND2(q0, p1);
t16 = NOR2(q2, q4);
t17 = NAND2(t15, t16);
f1 = OAI21(t11, t14, t17);
// Coordinate f2
t18 = NAND2(q4, q1);
t19 = NAND2(p4, p1);
t20 = OAI21(q2, t18, t19);
t21 = OAI21(p0, p1, p3);
t22 = NAND3(q3, q0, q2);
t23 = NAND2(t21, t22);
t24 = OAI21(q4, q3, p2);
t25 = OAI21(q0, q1, q2);
t26 = NAND2(t24, t25);
f2 = OAI21(t20, t23, t26);
// Coordinate f3
t27 = NAND2(p0, p4);
t28 = NAND2(p2, p3);
t29 = NAND2(q0, p1);
t30 = OAI21(t27, t28, t29);
t31 = OAI21(q2, q3, q4);
t32 = NAND3(q0, q3, q2);
t33 = OAI21(q0, t31, t32);
t34 = NAND4(q0, q2, p1, q3);
f3 = OAI21(t30, t33, t34);
// Coordinate f4
t35 = OAI21(q4, q1, p2);
t36 = OAI21(p2, p0, p3);
t37 = NAND2(t35, t36);
t38 = NAND2(p4, p1);
t39 = NAND3(q4, q0, q3);
t40 = OAI21(p2, t38, t39);
t41 = OAI21(q2, p0, q4);
t42 = OAI21(q3, p1, p4);
t43 = NAND2(t41, t42);
f4 = OAI21(t37, t40, t43);
```

A.3.3 Class x^7

This circuit obtained by exact latency minimization for each coordinate function realizes the x^7-class permutation in Table A.1. Input and output indices run from the least significant bit. The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps.

Listing A.3. Complete optimal APN circuit in the class x^7

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p1, q3);
t1 = OAI21(q1, q0, p4);
t2 = NAND2(t0, t1);
t3 = NAND2(p1, p0);
t4 = NAND2(q4, p3);
t5 = NAND2(q1, p2);
t6 = OAI21(t3, t4, t5);
t7 = OAI21(q1, q2, q0);
t8 = NAND2(q1, q2);
t9 = NAND3(q3, t7, t8);
f0 = OAI21(t2, t6, t9);
// Coordinate f1
t10 = OAI21(q1, q2, q3);
t11 = OAI21(q0, q2, q4);
t12 = NAND2(p1, p0);
t13 = OAI21(t10, t11, t12);
t14 = NAND2(p2, p4);
t15 = NAND2(q1, q2);
t16 = OAI21(q3, t14, t15);
t17 = NOR2(q0, q3);
t18 = OAI21(p2, q4, q1);
t19 = NAND2(t17, t18);
f1 = OAI21(t13, t16, t19);
// Coordinate f2
t20 = NAND2(q2, p4);
t21 = OAI21(q1, q4, q3);
t22 = NAND2(p3, p0);
t23 = NAND3(t20, t21, t22);
t24 = AND2(q1, q4);
t25 = OAI21(q3, p4, p2);
t26 = OAI21(p1, q0, q2);
t27 = OAI21(t24, t25, t26);
f2 = NAND2(t23, t27);
// Coordinate f3
t28 = NAND2(q2, q4);
t29 = OAI21(q4, p0, p3);
t30 = NAND2(t28, t29);
t31 = NAND2(p2, p4);
t32 = OAI21(q0, q3, q1);
t33 = NAND3(q1, q3, q0);
t34 = OAI21(t31, t32, t33);
t35 = NOR2(q0, q3);
t36 = OAI21(p4, p1, q2);
t37 = NAND2(t35, t36);
f3 = OAI21(t30, t34, t37);
// Coordinate f4
t38 = NAND2(p3, p1);
t39 = NAND2(p2, p0);
t40 = OAI21(q3, q0, q1);
t41 = OAI21(t38, t39, t40);
t42 = OAI21(q4, q3, q2);
t43 = OAI21(q1, q0, q2);
t44 = OAI21(q4, q1, q3);
t45 = OAI21(t42, t43, t44);
t46 = AND2(q4, q3);
t47 = OAI21(p1, p2, p0);
t48 = NAND2(t46, t47);
f4 = OAI21(t41, t45, t48);
```

A.3.4 Class x^{11}

This circuit obtained by exact latency minimization for each coordinate function realizes the x^{11}-class permutation in Table A.1. Input and output indices run from the least significant bit. The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps.

Listing A.4. Complete optimal APN circuit in the class x^{11}

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p2, p0);
t1 = NAND2(p1, q4);
t2 = NAND2(q1, p3);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(q2, q0, p4);
t5 = OAI21(q1, p3, q0);
t6 = NAND2(t4, t5);
t7 = OAI21(q2, p4, p3);
t8 = OAI21(p1, q4, q3);
t9 = NAND2(t7, t8);
f0 = OAI21(t3, t6, t9);
// Coordinate f1
t10 = OAI21(p3, p0, q4);
t11 = OAI21(p2, p0, p1);
t12 = NAND2(t10, t11);
t13 = NAND2(q3, q0);
t14 = NAND3(q2, q0, q1);
t15 = OAI21(q4, t13, t14);
t16 = OAI21(q3, p4, q1);
t17 = OAI21(q2, p4, p1);
t18 = NAND2(t16, t17);
f1 = OAI21(t12, t15, t18);
// Coordinate f2
t19 = OAI21(p2, p3, q1);
t20 = OAI21(p3, p4, p0);
t21 = NAND2(t19, t20);
t22 = OAI21(q2, q0, q3);
t23 = NAND3(q4, q0, q1);
t24 = OAI21(q1, t22, t23);
t25 = NAND2(q3, q4);
t26 = NAND2(q1, q0);
t27 = OAI21(q3, q4, q2);
t28 = NAND3(t25, t26, t27);
f2 = OAI21(t21, t24, t28);
// Coordinate f3
t29 = OAI21(q4, q2, q1);
t30 = NAND2(p1, p2);
t31 = OAI21(p3, t29, t30);
t32 = NAND2(p3, p4);
t33 = NAND2(p0, p2);
t34 = OAI21(q3, q2, q0);
t35 = OAI21(t32, t33, t34);
t36 = OAI21(q4, q1, p3);
t37 = OAI21(p4, p2, q3);
t38 = NAND2(t36, t37);
f3 = OAI21(t31, t35, t38);
// Coordinate f4
t39 = AND2(q1, q4);
t40 = OAI21(q0, p2, p3);
t41 = OAI21(p1, p2, q3);
t42 = OAI21(t39, t40, t41);
t43 = NOR2(q1, q4);
t44 = NOR2(q0, q2);
t45 = OAI21(p0, p3, p4);
t46 = OAI21(t43, t44, t45);
f4 = NAND2(t42, t46);
```

A.3.5 Class x^{15}

This circuit obtained by exact latency minimization for each coordinate function realizes the x^{15}-class permutation in Table A.1. Input and output indices run from the least significant bit. The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps.

Listing A.5. Complete optimal APN circuit in the class x^{15}

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(q3, q4, q2);
t1 = OAI21(q4, q1, q0);
t2 = NAND2(p2, p1);
t3 = OAI21(t0, t1, t2);
t4 = NAND2(q2, p0);
t5 = NAND2(q4, p0);
t6 = OAI21(q3, t4, t5);
t7 = NAND2(p4, p0);
t8 = NOR2(q3, q1);
t9 = NAND2(t7, t8);
f0 = OAI21(t3, t6, t9);
// Coordinate f1
t10 = OAI21(q2, q0, q4);
t11 = OAI21(q3, q0, q1);
t12 = NAND2(p0, p1);
t13 = OAI21(t10, t11, t12);
t14 = AOI21(q2, q4, q3);
t15 = NOR2(q2, q3);
t16 = OAI21(q4, q0, p1);
t17 = NAND2(t15, t16);
f1 = OAI21(t13, t14, t17);
// Coordinate f2
t18 = OAI21(p3, p0, p4);
t19 = OAI21(p3, p1, p2);
t20 = NAND2(t18, t19);
t21 = NAND2(q4, q3);
t22 = NAND2(q1, q2);
t23 = NAND2(q3, p0);
t24 = OAI21(t21, t22, t23);
t25 = OAI21(q1, q2, q3);
t26 = OAI21(q3, q1, q0);
t27 = NAND3(p4, t25, t26);
f2 = OAI21(t20, t24, t27);
// Coordinate f3
t28 = NAND2(q1, p0);
t29 = OAI21(q4, p2, q3);
t30 = NAND2(t28, t29);
t31 = NAND2(p4, p3);
t32 = OAI21(q1, q0, q2);
t33 = NAND2(q4, p2);
t34 = OAI21(t31, t32, t33);
t35 = OAI21(q2, p0, p1);
t36 = OAI21(p3, p2, q1);
t37 = NAND2(t35, t36);
f3 = OAI21(t30, t34, t37);
// Coordinate f4
t38 = NAND2(p3, p1);
t39 = NAND2(q4, p0);
t40 = NAND3(q2, q1, q0);
t41 = OAI21(t38, t39, t40);
t42 = OAI21(p3, p1, p4);
t43 = OAI21(p2, q0, q3);
t44 = NAND2(t42, t43);
t45 = OAI21(q3, q2, q4);
t46 = OAI21(q2, q0, q1);
t47 = OAI21(q2, q1, q3);
t48 = NAND3(t45, t46, t47);
f4 = OAI21(t41, t44, t48);
```

B Complete Gate-Level Circuits for the Comparison Benchmarks

The circuits follow the benchmark order in Table 4; the same Keccak circuit is used for both source baselines. Input and output indices run from the least significant bit unless otherwise stated. Gate definitions and pin order follow Table 2. Coordinate-function comparisons are collected in Appendix C.

B.1 Fides-5 / Primate [6]

The coordinate delays, in order from f_0 to f_4, are 130.977, 131.510, 128.861, 133.626, and 128.861 ps; the overall latency is 133.626 ps. The coordinate comparison is given in Table C.1 in Appendix C.

Listing B.1. Minimum-latency coordinate implementation: Fides-5 / Primate circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p1, p4);
t1 = NAND3(q3, q2, p0);
t2 = OAI21(p2, q0, p3);
t3 = NAND3(t0, t1, t2);
t4 = NOR2(q1, q4);
t5 = OAI21(p2, q0, q3);
t6 = NAND3(p3, q2, p0);
t7 = NAND3(t4, t5, t6);
f0 = NAND2(t3, t7);
// Coordinate f1
t8 = OAI21(p0, p2, p4);
t9 = OAI21(q0, q2, q3);
t10 = NAND3(q4, q0, q2);
t11 = OAI21(t8, t9, t10);
t12 = AND2(q1, q4);
t13 = OAI21(p3, p1, p0);
t14 = OAI21(q3, q2, q4);
t15 = OAI21(t12, t13, t14);
t16 = OAI21(p3, p4, p2);
t17 = OAI21(q3, q4, p1);
t18 = NOR2(q1, q0);
t19 = OAI21(t16, t17, t18);
f1 = OAI21(t11, t15, t19);
// Coordinate f2
t20 = OAI21(q3, q2, q4);
t21 = OAI21(q2, q1, q0);
t22 = AND2(t20, t21);
t23 = OAI21(q0, q2, q3);
t24 = OAI21(q2, q1, q4);
t25 = NAND2(p0, p1);
t26 = OAI21(t23, t24, t25);
t27 = OAI21(q3, q0, q4);
t28 = OAI21(q2, q4, q1);
t29 = NAND2(p4, p1);
t30 = NAND3(t27, t28, t29);
f2 = OAI21(t22, t26, t30);
// Coordinate f3
t31 = OAI21(p3, p2, q4);
t32 = OAI21(q3, p1, q0);
t33 = NAND3(q1, q2, q4);
t34 = OAI21(t31, t32, t33);
t35 = AND2(q4, q0);
t36 = OAI21(q1, q2, p3);
t37 = OAI21(q2, p0, p4);
t38 = OAI21(t35, t36, t37);
t39 = OAI21(p1, p2, q3);
t40 = OAI21(q2, p0, q4);
t41 = OAI21(p1, q0, p2);
t42 = NAND3(t39, t40, t41);
f3 = OAI21(t34, t38, t42);
// Coordinate f4
t43 = OAI21(q1, q2, p4);
t44 = NAND3(q1, q2, p0);
t45 = NAND2(t43, t44);
t46 = NAND2(q0, p3);
t47 = AND2(q1, q2);
t48 = NAND3(q1, q2, q3);
t49 = OAI21(t46, t47, t48);
t50 = AND2(q1, q0);
t51 = AND2(q2, q0);
t52 = OAI21(q3, q4, q0);
t53 = OAI21(t50, t51, t52);
f4 = OAI21(t45, t49, t53);
```

B.2 Shamash [6]

The coordinate delays, in order from f_0 to f_4, are 131.510, 131.510, 131.510, 131.510, and 131.510 ps; the overall latency is 131.510 ps. The coordinate comparison is given in Table C.2 in Appendix C.

Listing B.2. Minimum-latency coordinate implementation: Shamash circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(q1, p0, q4);
t1 = OAI21(q3, p0, q2);
t2 = OAI21(p1, p2, p4);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(p4, p2, p3);
t5 = OAI21(q4, q2, q0);
t6 = OAI21(p3, p0, p1);
t7 = OAI21(t4, t5, t6);
t8 = NOR2(q4, q3);
t9 = NOR2(q4, q0);
t10 = OAI21(p0, q2, q1);
t11 = OAI21(t8, t9, t10);
f0 = OAI21(t3, t7, t11);
// Coordinate f1
t12 = OAI21(q2, p1, q0);
t13 = OAI21(q4, p1, q3);
t14 = OAI21(p2, p3, p0);
t15 = OAI21(t12, t13, t14);
t16 = OAI21(p0, p3, p4);
t17 = OAI21(q0, q3, q1);
t18 = OAI21(p4, p1, p2);
t19 = OAI21(t16, t17, t18);
t20 = NOR2(q0, q4);
t21 = NOR2(q0, q1);
t22 = OAI21(p1, q3, q2);
t23 = OAI21(t20, t21, t22);
f1 = OAI21(t15, t19, t23);
// Coordinate f2
t24 = OAI21(q3, p2, q1);
t25 = OAI21(q0, p2, q4);
t26 = OAI21(p3, p4, p1);
t27 = OAI21(t24, t25, t26);
t28 = OAI21(p1, p4, p0);
t29 = OAI21(q1, q4, q2);
t30 = OAI21(p0, p2, p3);
t31 = OAI21(t28, t29, t30);
t32 = NOR2(q1, q0);
t33 = NOR2(q1, q2);
t34 = OAI21(p2, q4, q3);
t35 = OAI21(t32, t33, t34);
f2 = OAI21(t27, t31, t35);
// Coordinate f3
t36 = OAI21(q4, p3, q2);
t37 = OAI21(q1, p3, q0);
t38 = OAI21(p4, p0, p2);
t39 = OAI21(t36, t37, t38);
t40 = OAI21(p2, p0, p1);
t41 = OAI21(q2, q0, q3);
t42 = OAI21(p1, p3, p4);
t43 = OAI21(t40, t41, t42);
t44 = NOR2(q2, q1);
t45 = NOR2(q2, q3);
t46 = OAI21(p3, q0, q4);
t47 = OAI21(t44, t45, t46);
f3 = OAI21(t39, t43, t47);
// Coordinate f4
t48 = OAI21(p1, p2, q4);
t49 = OAI21(q1, q2, q3);
t50 = OAI21(p1, p3, p4);
t51 = OAI21(t48, t49, t50);
t52 = OAI21(q1, p2, p3);
t53 = OAI21(p1, q2, q0);
t54 = OAI21(p2, q3, p0);
t55 = OAI21(t52, t53, t54);
t56 = NAND2(p0, p3);
t57 = OAI21(q2, q0, q4);
t58 = OAI21(q1, q3, q0);
t59 = NAND3(t56, t57, t58);
f4 = OAI21(t51, t55, t59);
```

B.3 SC2000-5 [6]

The coordinate delays, in order from f_0 to f_4, are 131.510, 131.510, 131.510, 128.861, and 126.745 ps; the overall latency is 131.510 ps. The coordinate comparison is given in Table C.3 in Appendix C.

Listing B.3. Minimum-latency coordinate implementation: SC2000-5 circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(p1, p2, p3);
t1 = OAI21(p4, q1, q0);
t2 = NAND3(q4, q1, q2);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(p4, p1, p2);
t5 = OAI21(p4, q0, q3);
t6 = OAI21(p3, q1, p0);
t7 = OAI21(t4, t5, t6);
t8 = NAND2(q3, p2);
t9 = OAI21(q1, q2, q0);
t10 = NAND2(q4, p0);
t11 = NAND3(t8, t9, t10);
f0 = OAI21(t3, t7, t11);
// Coordinate f1
t12 = OAI21(q1, p4, q3);
t13 = OAI21(p1, q4, q2);
t14 = NAND2(p1, q1);
t15 = OAI21(t12, t13, t14);
t16 = AND2(q1, q0);
t17 = OAI21(q1, q0, p3);
t18 = OAI21(p0, p3, p2);
t19 = OAI21(t16, t17, t18);
t20 = AND2(q4, q2);
t21 = OAI21(q1, p4, p0);
t22 = OAI21(p3, q2, q0);
t23 = OAI21(t20, t21, t22);
f1 = OAI21(t15, t19, t23);
// Coordinate f2
t24 = NAND2(p2, q0);
t25 = OAI21(p1, q4, q3);
t26 = OAI21(p4, p0, q2);
t27 = OAI21(t24, t25, t26);
t28 = OAI21(p1, p4, p3);
t29 = OAI21(q1, q4, p2);
t30 = OAI21(p4, p3, p0);
t31 = OAI21(t28, t29, t30);
t32 = AND2(q4, q2);
t33 = OAI21(q0, p3, p1);
t34 = OAI21(p2, q3, q1);
t35 = OAI21(t32, t33, t34);
f2 = OAI21(t27, t31, t35);
// Coordinate f3
t36 = OAI21(p2, p3, q1);
t37 = NAND3(q4, p3, p2);
t38 = NAND2(t36, t37);
t39 = NAND2(p1, p4);
t40 = NOR2(q2, q0);
t41 = NAND3(q2, q4, q0);
t42 = OAI21(t39, t40, t41);
t43 = OAI21(q3, p0, p1);
t44 = OAI21(p4, q0, q1);
t45 = NAND2(t43, t44);
f3 = OAI21(t38, t42, t45);
// Coordinate f4
t46 = OAI21(p0, p4, p2);
t47 = OAI21(p2, p1, p3);
t48 = NAND2(t46, t47);
t49 = NAND2(q2, p0);
t50 = NAND2(p4, q1);
t51 = NAND2(q0, p1);
t52 = OAI21(t49, t50, t51);
t53 = OAI21(q0, q3, q4);
t54 = OAI21(q1, q3, q0);
t55 = NAND3(p2, t53, t54);
f4 = OAI21(t48, t52, t55);
```

B.4 Ascon / ISAP [6]

The coordinate delays, in order from f_0 to f_4, are 112.811, 138.581, 126.745, 140.697, and 128.861 ps; the overall latency is 140.697 ps. The coordinate comparison is given in Table C.4 in Appendix C.

Listing B.4. Minimum-latency coordinate implementation: Ascon / ISAP circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND3(p4, p3, p1);
t1 = NAND3(q3, q1, p0);
t2 = NAND3(q4, p3, q1);
t3 = NAND3(q3, p1, q0);
f0 = NAND4(t0, t1, t2, t3);
// Coordinate f1
t4 = AND2(q0, q1);
t5 = OAI21(q0, q1, q4);
t6 = NAND3(q2, q3, q4);
t7 = OAI21(t4, t5, t6);
t8 = AND2(q2, q3);
t9 = OAI21(q2, q3, p4);
t10 = NAND3(p2, p3, q4);
t11 = OAI21(t8, t9, t10);
t12 = NAND2(p0, p1);
t13 = NAND3(q4, q0, q1);
t14 = NAND2(q2, p3);
t15 = NAND2(p2, q3);
t16 = NAND4(t12, t13, t14, t15);
f1 = OAI21(t7, t11, t16);
// Coordinate f2
t17 = OAI21(q3, q0, q2);
t18 = OAI21(q2, q3, p1);
t19 = NAND2(t17, t18);
t20 = NAND2(p2, p3);
t21 = NAND2(q1, p0);
t22 = OAI21(q2, q3, q0);
t23 = OAI21(t20, t21, t22);
t24 = AND2(q2, q3);
t25 = NAND2(t21, t24);
f2 = OAI21(t19, t23, t25);
// Coordinate f3
t26 = NAND2(p0, q4);
t27 = NAND3(p1, p2, p3);
t28 = NAND2(q0, p4);
t29 = NAND3(q1, q2, q3);
t30 = NAND4(t26, t27, t28, t29);
t31 = NAND2(p0, p4);
t32 = OAI21(q2, q3, p1);
t33 = NAND2(q0, q4);
t34 = OAI21(p2, p3, q1);
t35 = NAND4(t31, t32, t33, t34);
f3 = NAND2(t30, t35);
// Coordinate f4
t36 = NAND2(p4, q3);
t37 = AND2(q1, q2);
t38 = NAND2(p1, p2);
t39 = OAI21(t36, t37, t38);
t40 = OAI21(p1, p0, p3);
t41 = NAND4(q1, q2, q4, q3);
t42 = NAND2(t40, t41);
t43 = NAND2(q0, p3);
t44 = OAI21(q2, q4, q3);
t45 = NAND3(t43, t44, p1);
f4 = OAI21(t39, t42, t45);
```

B.5 Dry-GASCON128 [6]

The coordinate delays, in order from f_0 to f_4, are 128.861, 140.697, 126.745, 138.581, and 112.811 ps; the overall latency is 140.697 ps. The coordinate comparison is given in Table C.5 in Appendix C.

Listing B.5. Minimum-latency coordinate implementation: Dry-GASCON128 circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p2, q1);
t1 = AND2(q3, q0);
t2 = NAND2(p3, p0);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(p3, p4, p1);
t5 = NAND4(q3, q0, q2, q1);
t6 = NAND2(t4, t5);
t7 = NAND2(q4, p1);
t8 = OAI21(q0, q2, q1);
t9 = NAND3(t7, t8, p3);
f0 = OAI21(t3, t6, t9);
// Coordinate f1
t10 = NAND2(p0, q4);
t11 = NAND3(p1, p2, p3);
t12 = NAND2(q0, p4);
t13 = NAND3(q1, q2, q3);
t14 = NAND4(t10, t11, t12, t13);
t15 = NAND2(p0, p4);
t16 = OAI21(q2, q3, p1);
t17 = NAND2(q0, q4);
t18 = OAI21(p2, p3, q1);
t19 = NAND4(t15, t16, t17, t18);
f1 = NAND2(t14, t19);
// Coordinate f2
t20 = OAI21(q2, q4, q1);
t21 = OAI21(q1, q2, p3);
t22 = NAND2(t20, t21);
t23 = NAND2(p1, p2);
t24 = NAND2(q3, p4);
t25 = OAI21(q1, q2, q4);
t26 = OAI21(t23, t24, t25);
t27 = AND2(q1, q2);
t28 = NAND2(t24, t27);
f2 = OAI21(t22, t26, t28);
// Coordinate f3
t29 = AND2(q3, q4);
t30 = OAI21(q3, q4, q0);
t31 = NAND3(q1, q2, q0);
t32 = OAI21(t29, t30, t31);
t33 = AND2(q1, q2);
t34 = OAI21(q1, q2, p0);
t35 = NAND3(p1, p2, q0);
t36 = OAI21(t33, t34, t35);
t37 = NAND2(p3, p4);
t38 = NAND3(q0, q3, q4);
t39 = NAND2(q1, p2);
t40 = NAND2(p1, q2);
t41 = NAND4(t37, t38, t39, t40);
f3 = OAI21(t32, t36, t41);
// Coordinate f4
t42 = NAND3(p0, p1, p3);
t43 = NAND3(q1, q3, p4);
t44 = NAND3(q0, p1, q3);
t45 = NAND3(q1, p3, q4);
f4 = NAND4(t42, t43, t44, t45);
```

B.6 Keccak [6, 18]

The coordinate delays, in order from f_0 to f_4, are 94.870, 94.870, 94.870, 94.870, and 94.870 ps; the overall latency is 94.870 ps. The coordinate comparison is given in Table C.6 in Appendix C.

Listing B.6. Minimum-latency coordinate implementation: Keccak circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = AND2(q0, q3);
t1 = OAI21(q0, q3, q4);
t2 = NAND2(p0, p4);
f0 = OAI21(t0, t1, t2);
// Coordinate f1
t3 = AND2(q1, q4);
t4 = OAI21(q1, q4, q0);
t5 = NAND2(p1, p0);
f1 = OAI21(t3, t4, t5);
// Coordinate f2
t6 = AND2(q2, q0);
t7 = OAI21(q2, q0, q1);
t8 = NAND2(p2, p1);
f2 = OAI21(t6, t7, t8);
// Coordinate f3
t9 = AND2(q3, q1);
t10 = OAI21(q3, q1, q2);
t11 = NAND2(p3, p2);
f3 = OAI21(t9, t10, t11);
// Coordinate f4
t12 = AND2(q4, q2);
t13 = OAI21(q4, q2, q3);
t14 = NAND2(p4, p3);
f4 = OAI21(t12, t13, t14);
```

B.7 ICEPOLE [6]

The coordinate delays, in order from f_0 to f_4, are 126.745, 126.745, 126.745, 126.745, and 126.745 ps; the overall latency is 126.745 ps. The coordinate comparison is given in Table C.7 in Appendix C.

Listing B.7. Minimum-latency coordinate implementation: ICEPOLE circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = AND2(q2, q3);
t1 = AND2(q0, q4);
t2 = NAND3(q1, t0, t1);
t3 = OAI21(q4, p1, p0);
t4 = OAI21(q4, p2, p3);
t5 = OAI21(p4, q3, q0);
t6 = OAI21(t3, t4, t5);
f0 = NAND2(t2, t6);
// Coordinate f1
t7 = AND2(q3, q4);
t8 = AND2(q1, q0);
t9 = NAND3(q2, t7, t8);
t10 = OAI21(q0, p2, p1);
t11 = OAI21(q0, p3, p4);
t12 = OAI21(p0, q4, q1);
t13 = OAI21(t10, t11, t12);
f1 = NAND2(t9, t13);
// Coordinate f2
t14 = AND2(q4, q0);
t15 = AND2(q2, q1);
t16 = NAND3(q3, t14, t15);
t17 = OAI21(q1, p3, p2);
t18 = OAI21(q1, p4, p0);
t19 = OAI21(p1, q0, q2);
t20 = OAI21(t17, t18, t19);
f2 = NAND2(t16, t20);
// Coordinate f3
t21 = AND2(q4, q1);
t22 = AND2(q3, q2);
t23 = NAND3(q0, t21, t22);
t24 = OAI21(q2, p0, p3);
t25 = OAI21(q2, p4, p1);
t26 = OAI21(p2, q1, q3);
t27 = OAI21(t24, t25, t26);
f3 = NAND2(t23, t27);
// Coordinate f4
t28 = AND2(q1, q2);
t29 = AND2(q4, q3);
t30 = NAND3(q0, t28, t29);
t31 = OAI21(q3, p0, p4);
t32 = OAI21(q3, p1, p2);
t33 = OAI21(p3, q2, q4);
t34 = OAI21(t31, t32, t33);
f4 = NAND2(t30, t34);
```

B.8 SYCON [6]

The coordinate delays, in order from f_0 to f_4, are 128.000, 140.697, 138.581, 112.811, and 127.521 ps; the overall latency is 140.697 ps. The coordinate comparison is given in Table C.8 in Appendix C.

Listing B.8. Minimum-latency coordinate implementation: SYCON circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(q0, q4, q3);
t1 = OAI21(q1, q2, q0);
t2 = AND2(t0, t1);
t3 = NAND3(q1, q2, q0);
t4 = NAND3(q3, q4, q0);
t5 = OAI21(q3, t3, t4);
t6 = AND2(q1, q2);
t7 = NOR2(q1, q2);
t8 = NOR2(q3, q0);
t9 = OAI21(t6, t7, t8);
f0 = OAI21(t2, t5, t9);
// Coordinate f1
t10 = NAND2(p2, q4);
t11 = NAND3(p0, p1, p3);
t12 = NAND2(q2, p4);
t13 = NAND3(q0, q1, q3);
t14 = NAND4(t10, t11, t12, t13);
t15 = NAND2(p2, p4);
t16 = OAI21(q1, q3, p0);
t17 = NAND2(q2, q4);
t18 = OAI21(p1, p3, q0);
t19 = NAND4(t15, t16, t17, t18);
f1 = NAND2(t14, t19);
// Coordinate f2
t20 = OAI21(q2, p0, q3);
t21 = OAI21(p2, q1, q3);
t22 = NAND3(p2, p4, q1);
t23 = OAI21(t20, t21, t22);
t24 = AND2(q2, q3);
t25 = OAI21(q2, q4, p1);
t26 = NAND3(p2, q0, p3);
t27 = OAI21(t24, t25, t26);
t28 = NAND2(p4, p1);
t29 = OAI21(q2, q0, q3);
t30 = NAND2(p0, p3);
t31 = OAI21(q2, q1, q4);
t32 = NAND4(t28, t29, t30, t31);
f2 = OAI21(t23, t27, t32);
// Coordinate f3
t33 = NAND3(p2, q3, p0);
t34 = NAND3(q2, q3, q0);
t35 = NAND3(p3, q0, p4);
t36 = NAND3(p3, p0, q4);
f3 = NAND4(t33, t34, t35, t36);
// Coordinate f4
t37 = NAND2(p0, q4);
t38 = AND2(q1, q3);
t39 = NAND2(p1, p3);
t40 = OAI21(t37, t38, t39);
t41 = NAND2(q1, q3);
t42 = NAND3(q0, q3, q1);
t43 = OAI21(q4, t41, t42);
t44 = NAND4(q4, p1, p0, p3);
f4 = OAI21(t40, t43, t44);
```

B.9 Inverse Keccak [18]

The coordinate delays, in order from f_0 to f_4, are 130.977, 130.977, 130.977, 130.977, and 130.977 ps; the overall latency is 130.977 ps. The coordinate comparison is given in Table C.9 in Appendix C.

Listing B.9. Minimum-latency coordinate implementation: Inverse Keccak circuit

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = OAI21(p4, p2, p1);
t1 = OAI21(q0, q1, p3);
t2 = OAI21(p0, q1, q3);
t3 = NAND3(t0, t1, t2);
t4 = NAND3(p0, q4, p3);
t5 = NAND3(q4, p1, q2);
t6 = OAI21(p4, q3, q0);
t7 = NAND3(t4, t5, t6);
f0 = NAND2(t3, t7);
// Coordinate f1
t8 = OAI21(p0, p3, p2);
t9 = OAI21(q1, q2, p4);
t10 = OAI21(p1, q2, q4);
t11 = NAND3(t8, t9, t10);
t12 = NAND3(p1, q0, p4);
t13 = NAND3(q0, p2, q3);
t14 = OAI21(p0, q4, q1);
t15 = NAND3(t12, t13, t14);
f1 = NAND2(t11, t15);
// Coordinate f2
t16 = OAI21(p1, p4, p3);
t17 = OAI21(q2, q3, p0);
t18 = OAI21(p2, q3, q0);
t19 = NAND3(t16, t17, t18);
t20 = NAND3(p2, q1, p0);
t21 = NAND3(q1, p3, q4);
t22 = OAI21(p1, q0, q2);
t23 = NAND3(t20, t21, t22);
f2 = NAND2(t19, t23);
// Coordinate f3
t24 = OAI21(p2, p0, p4);
t25 = OAI21(q3, q4, p1);
t26 = OAI21(p3, q4, q1);
t27 = NAND3(t24, t25, t26);
t28 = NAND3(p3, q2, p1);
t29 = NAND3(q2, p4, q0);
t30 = OAI21(p2, q1, q3);
t31 = NAND3(t28, t29, t30);
f3 = NAND2(t27, t31);
// Coordinate f4
t32 = OAI21(p3, p1, p0);
t33 = OAI21(q4, q0, p2);
t34 = OAI21(p4, q0, q2);
t35 = NAND3(t32, t33, t34);
t36 = NAND3(p4, q3, p2);
t37 = NAND3(q3, p0, q1);
t38 = OAI21(p3, q2, q4);
t39 = NAND3(t36, t37, t38);
f4 = NAND2(t35, t39);
```

B.10 Ascon (SLICE baseline) [1]

This Ascon implementation obtained by exact latency minimization for each coordinate function [24] is used in the SLICE baseline comparison in Table 4. Input and output signal indices follow the most-significant-bit-first Ascon source convention. The coordinate comparison is given in Table C.10. The coordinate delays, in order from f_0 to f_4, are 128.861, 140.697, 126.745, 138.581, and 112.811 ps; the overall latency is 140.697 ps.

Listing B.10. Minimum-latency coordinate implementation: Ascon circuit for the SLICE comparison

```
p0 = BUF(x0); q0 = INV(x0);
p1 = BUF(x1); q1 = INV(x1);
p2 = BUF(x2); q2 = INV(x2);
p3 = BUF(x3); q3 = INV(x3);
p4 = BUF(x4); q4 = INV(x4);
// Coordinate f0
t0 = NAND2(p2, q1);
t1 = AND2(q3, q0);
t2 = NAND2(p3, p0);
t3 = OAI21(t0, t1, t2);
t4 = OAI21(p3, p4, p1);
t5 = NAND4(q3, q0, q2, q1);
t6 = NAND2(t4, t5);
t7 = NAND2(q4, p1);
t8 = OAI21(q0, q2, q1);
t9 = NAND3(t7, t8, p3);
f0 = OAI21(t3, t6, t9);
// Coordinate f1
t10 = NAND2(q0, p4);
t11 = NAND3(p1, p2, p3);
t12 = NAND2(p0, q4);
t13 = NAND3(q1, q2, q3);
t14 = NAND4(t10, t11, t12, t13);
t15 = NAND2(p0, p4);
t16 = OAI21(p1, p3, q2);
t17 = NAND2(q0, q4);
t18 = OAI21(q1, q3, p2);
t19 = NAND4(t15, t16, t17, t18);
f1 = NAND2(t14, t19);
// Coordinate f2
t20 = AOI22(q3, p4, p1, p2);
t21 = NAND2(p2, p4);
t22 = NAND2(p1, q3);
t23 = NAND2(q1, q2);
t24 = OAI21(t21, t22, t23);
t25 = NAND2(q3, p4);
t26 = AND2(q1, q2);
t27 = NAND2(t25, t26);
f2 = OAI21(t20, t24, t27);
// Coordinate f3
t28 = AND2(q3, q4);
t29 = OAI21(q3, q4, q0);
t30 = NAND3(q1, q2, q0);
t31 = OAI21(t28, t29, t30);
t32 = AND2(q1, q0);
t33 = OAI21(q1, q0, p2);
t34 = NAND3(p1, q2, p0);
t35 = OAI21(t32, t33, t34);
t36 = NAND2(p3, p4);
t37 = NAND2(q1, p2);
t38 = NAND2(p1, q2);
t39 = NAND3(q0, q3, q4);
t40 = NAND4(t36, t37, t38, t39);
f3 = OAI21(t31, t35, t40);
// Coordinate f4
t41 = NAND3(p0, p1, p3);
t42 = NAND3(q1, q3, p4);
t43 = NAND3(q0, p1, q3);
t44 = NAND3(q1, p3, q4);
f4 = NAND4(t41, t42, t43, t44);
```

C Coordinate-Function Comparisons

Each subsection includes a coordinate-function comparison under the common gate-delay and input-arrival model of Section 3.6. The baseline is the locally reproduced depth-oriented structure for Rasoolzadeh [6], the mapped author RTL for Jia et al. [18], or the published SLICE gate listing for Kim et al. [1]. All baselines are evaluated under the prescribed model. The *This work* column gives the exact minimum latency of the same coordinate function. Reduction is $100(D_{\text{baseline}} - D_{\text{ours}})/D_{\text{baseline}}$. Truth tables use the encoding of Section 2.1. Keccak has one table containing both source baselines, since the fixed mapping and circuit obtained by exact latency minimization for each coordinate function are shared.

C.1 Fides-5 / Primate

Table C.1: Coordinate-function latency comparison for Fides-5 / Primate

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xc6390af5`	161.512	130.977	18.91
f_1	`0xdd2d7888`	173.348	131.510	24.14
f_2	`0x35359f60`	163.628	128.861	21.25
f_3	`0x6f5390ac`	175.953	133.626	24.06
f_4	`0x95c0a9fc`	133.626	128.861	3.57

C.2 Shamash

Table C.2: Coordinate-function latency comparison for Shamash

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xde8b2174`	170.509	131.510	22.87
f_1	`0xa6a9959a`	185.685	131.510	29.18
f_2	`0xc939c9c6`	169.221	131.510	22.29
f_3	`0xf0c35a96`	166.277	131.510	20.91
f_4	`0x44bb1ee1`	181.351	131.510	27.48

C.3 SC2000-5

Table C.3: Coordinate-function latency comparison for SC2000-5

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xeb0f449c`	175.913	131.510	25.24
f_1	`0x09538bde`	166.277	131.510	20.91
f_2	`0x43981fad`	168.650	131.510	22.02
f_3	`0xc4836eea`	175.470	128.861	26.56
f_4	`0xe1a9a31b`	172.491	126.745	26.52

C.4 Ascon / ISAP

Table C.4: Coordinate-function latency comparison for Ascon / ISAP

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xcc663366`	140.697	112.811	19.82
f_1	`0xf00f6996`	166.855	138.581	16.95
f_2	`0xd22dd22d`	128.000	126.745	0.98
f_3	`0x6aa99556`	166.467	140.697	15.48
f_4	`0x99c3993c`	175.725	128.861	26.67

C.5 Dry-GASCON128

Table C.5: Coordinate-function latency comparison for Dry-GASCON128

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xed1221de`	167.105	128.861	22.89
f_1	`0x6aa99556`	166.467	140.697	15.48
f_2	`0xc33cc3c3`	128.861	126.745	1.64
f_3	`0x96c3c396`	168.393	138.581	17.70
f_4	`0x8877bb44`	140.697	112.811	19.82

C.6 Keccak

Table C.6: Coordinate-function latency comparison for Keccak

Coordinate	Truth table	Source	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xaaaa55aa`	[18]	119.047	94.870	20.31
f_1	`0x9999cccc`	[18]	119.047	94.870	20.31
f_2	`0xd2d2d2d2`	[18]	119.047	94.870	20.31
f_3	`0xf30cf30c`	[18]	119.047	94.870	20.31
f_4	`0xff0f00f0`	[18]	119.047	94.870	20.31
f_0	`0xaaaa55aa`	[6]	96.210	94.870	1.39
f_1	`0x9999cccc`	[6]	96.210	94.870	1.39
f_2	`0xd2d2d2d2`	[6]	96.210	94.870	1.39
f_3	`0xf30cf30c`	[6]	96.210	94.870	1.39
f_4	`0xff0f00f0`	[6]	96.210	94.870	1.39

C.7 ICEPOLE

Table C.7: Coordinate-function latency comparison for ICEPOLE

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0x2aaa55ab`	141.146	126.745	10.20
f_1	`0x1999cccd`	141.146	126.745	10.20
f_2	`0x52d2d2d3`	141.146	126.745	10.20
f_3	`0x730cf30d`	141.146	126.745	10.20
f_4	`0x7f0f00f1`	141.146	126.745	10.20

C.8 SYCON

Table C.8: Coordinate-function latency comparison for SYCON

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0x965596aa`	169.032	128.000	24.27
f_1	`0x78e1871e`	168.393	140.697	16.45
f_2	`0x936c639c`	166.277	138.581	16.66
f_3	`0x55a5aaa5`	140.697	112.811	19.82
f_4	`0x33cc9966`	128.861	127.521	1.04

C.9 Inverse Keccak

Table C.9: Coordinate-function latency comparison for Inverse Keccak

Coordinate	Truth table	Baseline (ps)	This work (ps)	Reduction (%)
f_0	`0xaaaa59a6`	183.532	130.977	28.64
f_1	`0x99c9cc9c`	183.532	130.977	28.64
f_2	`0xd2d2e1d2`	183.532	130.977	28.64
f_3	`0xf609f30c`	183.532	130.977	28.64
f_4	`0xff2d00d2`	183.532	130.977	28.64

C.10 Ascon (SLICE baseline)

The coordinate order follows the most-significant-bit-first Ascon convention used in Section B.10. Coordinate areas refer to individual output cones; overall area counts their union with shared gates included once. The circuit used here and the Ascon / ISAP circuit above use different realizations with the same minimum latency.

Table C.10: Coordinate and overall comparison with the reconstructed SLICE Ascon circuit under the common model. Bold values indicate the smaller result for each metric.

	Delay (ps)		Area (GE)		AD (GE·ps)	
Coordinate	SLICE [1]	This work	SLICE	This work	SLICE	This work
f_0	206.369	**128.861**	**14.67**	22.00	3027.43	**2834.94**
f_1	176.651	**140.697**	**14.00**	22.01	**2473.11**	3096.74
f_2	148.765	**126.745**	**8.67**	15.67	**1289.79**	1986.09
f_3	176.651	**138.581**	**11.67**	26.32	**2061.52**	3647.45
f_4	176.651	**112.811**	**11.34**	13.67	2003.22	**1542.13**
Overall	206.369	**140.697**	**34.01**	71.28	**7018.61**	10028.88