BASTION: Lock-Step, Staggered and Sequential Share Schedules for a First-Order Masked ML-KEM NTT on FPGA

Marcos Lázaro Díaz

Universidad Politécnica de Madrid, Spain

Abstract. The Number Theoretic Transform (NTT) is the arithmetic core of ML-KEM (FIPS 203) and a prominent side-channel target in hardware. Because the transform is linear, first-order arithmetic masking applies share-wise without fresh randomness, which leaves one design decision open: when each share is processed. Published designs process the shares in sequence on one datapath, or on one datapath per share started a configurable delay apart and in random order; running both datapaths in lock-step is the third option. We implement the three schedules on one first-order masked NTT core for ML-KEM on Artix-7 and measure their cost and pre-silicon leakage under one synthesis flow. An iso-architectural unmasked control separates the cost of masking from that of parallelisation: the delivered lock-step core costs 1.56× the LUTs, 2.33× the slices and twice the DSPs of the control at an unchanged cycle count and 196.1 instead of 227.3 MHz, while the sequential core needs 9 LUTs more than the control at twice the cycles. We state when a set of samples combines both shares of a secret, measure the criterion on the RTL and use it as a cycle-accurate oracle. On a whole-design toggle model that sums both share domains, calibrated with injected flaws detected within 750 traces, no schedule leaks at first order in five pairs of runs of 10^5 traces. At second order the schedules form a scale: the lock-step core leaks in single samples, the sequential core only at the pair of samples that its public offset relates, and the staggered core in both ways at every delay we tested short of one transform. Every detected sample lies in the support view of the oracle, which flags up to 80% of the samples of a window for the univariate test, and some lie outside its form view, so lazy reduction matters. A check on the routed netlist finds no cell shared by the domains of the lock-step and staggered cores, and finds the cells that synthesis shares once the preservation attributes are removed. Silicon validation is left to future work.

Keywords: ML-KEM · Number Theoretic Transform · Masking · Share scheduling · Side-channel analysis · TVLA · FPGA

1 Introduction

Shor's algorithm [Sho94] breaks the integer factorisation and discrete logarithm problems underpinning RSA, DSA and (elliptic-curve) Diffie–Hellman, and in August 2024 NIST responded with FIPS 203 [Nat24], which specifies the *Module-Lattice-Based Key-Encapsulation Mechanism* (ML-KEM), derived from CRYSTALS-Kyber [ABD+21]. The computational core of ML-KEM is the *Number Theoretic Transform* (NTT) over the ring $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q = 3329$. Polynomial multiplication, which dominates key generation, encapsulation and decapsulation, is carried out by pointwise multiplication in the NTT domain, reducing its cost from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ [Nat24, XL21]. In all three algorithms the forward transform is applied to secret polynomials, the key-generation secrets

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-11-04.

and the encryption randomness, so every coefficient routed through the adders, multipliers and pipeline registers of an NTT datapath depends on secret material. This makes the NTT a prominent side-channel target. Attacks on lattice-based implementations have been demonstrated repeatedly, including single-trace attacks on the NTT [PPM17, PP19] and EM attacks that exploit generic vulnerabilities of lattice-based PKE and KEMs [RBSC20]. On FPGA, deep-learning profiling has classified pairs of secret coefficients with over 95% accuracy from the first stage of an unprotected Kyber NTT accelerator [CWZ+25]. An ML-KEM hardware accelerator therefore has to treat side-channel resistance as a design goal from the start.

State of the art. Unprotected FPGA NTT designs have progressed rapidly. Xing and Li [XL21] report a compact full-protocol Kyber implementation on Artix-7 with a unified butterfly and careful scheduling of NTT and hashing. Mert et al. [MÖS20] and Bisheh-Niasar et al. [BNAMK21] establish multi-butterfly datapaths with conflict-free memory banking as the standard technique for throughput-oriented NTT accelerators, memory parallelisation for iterative NTTs is still being refined [IKR+26], and unified designs that support both ML-KEM and ML-DSA [KBMSH24, TAM25] push performance further. None of these reports a side-channel countermeasure.

On the protection side, Reparaz et al. [RSVV15] masked Ring-LWE decryption in hardware: the secret polynomial is split into two arithmetic shares, the multiplication by the ciphertext and the inverse NTT, both linear in the secret, are applied to each share in turn on a single coprocessor, and a dedicated masked decoder handles the non-linear final step. Oder et al. [OSPG18] later extended first-order masking to a CCA2-secure Ring-LWE scheme in software. For Kyber on hardware, Jati et al. [JGCS24] proposed a hiding-protected implementation with random delays, address shuffling and instruction randomisation; they verify leakage with TVLA at 2×10^5 traces but do not implement arithmetic masking, so their design has no asymptotic guarantee against averaging attacks. Masked Kyber has been implemented as a pure-hardware design that runs its NTT core once per share [KNAH22] and as masked accelerators driven through RISC-V instruction-set extensions [FVB+22]. Carrera et al. [CVBB24] implement on FPGA the local-masking approach of Ravi et al. [RPBC20], which randomises the twiddle factors rather than sharing the data, and show that some configurations remain vulnerable; the same authors later showed, on simulated traces, that soft-analytical attacks on local masking with few masks per layer can outperform those on the unmasked NTT [CBVB25]. Shuffling has been studied as a standalone hiding mechanism for NTT hardware [CMJ23]. Land et al. [LMRBG24] apply a gadget-based masking framework to Streamlined NTRU Prime on Artix-7, with formal verification and TVLA of its building blocks, and discuss but do not implement its extension to Kyber. For ML-KEM itself, the design closest to this work is HOPE-MLKEM [CRNTC26], an open-source, configurable hardware implementation of the whole KEM, evaluated in first- and third-order masked configurations. Its linear polynomial arithmetic, the NTT included, is masked share-wise: at first order each share has its own butterfly unit and memories, the two units start a configurable delay of at least one cycle apart in a random order, and a first-order TVLA of ML-KEM-768 measured on a SAKURA-X board stays within the ± 4.5 threshold at 10^6 traces. Our analysis uses the t-probing model of Ishai, Sahai and Wagner [ISW03], its glitch- and transition-robust refinement [FGM+18] and Domain-Oriented Masking (DOM) [GMK16]; threshold implementations [NRR06] are the other established route to glitch-resistant hardware masking.

Research gap. Multi-butterfly NTT datapaths with conflict-free banking are mature prior art, as are dummy operations and shuffled execution as hiding countermeasures. Masking the NTT share-wise is not new either: Reparaz et al. [RSVV15] and Kamucheka

et al. [KNAH22] process the two shares in sequence on one unit, and the first-order configuration of HOPE-MLKEM [CRNTC26] gives each share its own butterfly unit and starts the units a configurable delay apart. These share schedules differ in area and latency, and in whether the two shares of an intermediate are ever handled in the same clock cycle, so that one power sample combines them. We found no measurement of this trade-off for NTT hardware under one synthesis flow and one leakage model that sums both share domains, at first and second order and with deliberately injected flaws as positive controls. This is the gap we address.

Contributions. We present BASTION, a first-order masked NTT core for ML-KEM on FPGA, in three share schedules. Our contributions are:

- **Three share schedules under one flow.** Lock-step, staggered with a configurable delay and random order, and sequential, implemented on one first-order masked NTT core for ML-KEM-512 and synthesised under one flow, with an iso-architectural unmasked control that separates the cost of masking from that of parallelisation (Sections 3.4 and 3.5). At matched synthesis regime, masking in lock-step costs 1.76× the LUTs and 1.86× the slices of the control at an unchanged cycle count; under the preservation attributes the sequential core needs 9 LUTs more than the control at twice its cycles, and the staggered core 1.38× the LUTs of the lock-step core.
- **A criterion for when the shares meet, measured on the RTL.** Every value a share domain handles is a function of linear forms in that domain's share vector, so a set of samples combines both shares of the secret exactly when the forms of the two domains are linearly dependent (Observation 1). We measure the forms and the supports of every register on the RTL itself and turn the criterion into a cycle-accurate oracle (Section 6.4). It shows that a delay of one iteration does not separate the domains, because the results of a butterfly in one domain meet its operands in the other, and that two iterations do in the form view but not under lazy reduction. It also showed that the first version of our cores let the datapath run on before the pipeline had drained, at the end of the transform and between the passes of the sequential core; a three-cycle flush and a read-back that waits two cycles after done remove both.
- **A calibrated leakage evaluation.** A pre-silicon TVLA on a whole-design toggle model that sums both share domains, with first-order, univariate second-order and bivariate tests, calibrated with three injected flaws detected within 150 to 750 traces (Section 6). In five pairs of runs of 10^5 traces no schedule leaks at first order. At second order the schedules form a scale, from the lock-step core, which leaks in single samples, to the sequential core, which leaks only at the pair of samples its public offset relates; the staggered core leaks in both ways at every delay we tested short of one transform. Every detected sample lies in the support view of the oracle, and some outside its form view, so lazy reduction matters.
- **A structural check at cell level.** Queries on the routed netlist for timing paths between the domains, from the domains into the shared control, and for every primitive whose input cone contains registers of both domains (Section 4.5). The lock-step and staggered cores pass, with the read-back multiplexer as the only, documented exception; the lock-step core synthesised without the preservation attributes fails all three, because synthesis merges a register across the boundary and the physical optimiser moves share data into the controller.

All configurations target an Artix-7 XC7A200T-FBG676-2 (AC701 platform) under Vivado 2022.1, the family of Xing and Li [XL21]. They reproduce the reference NTT on six

ML-KEM-512 test vectors, and every one of the 8.08×10^6 executions of the leakage campaigns, about half of them on one fixed input and the rest on random ones, is checked against an independent model of the FIPS 203 transform; the RTL, testbenches, leakage harness, oracle and synthesis scripts are released as an artifact[1].

Scope and non-contributions. We claim no novelty in the components we reuse: the share-wise linearity that removes the randomness cost [ISW03, GMK16], the sequential and staggered schedules [RSVV15, CRNTC26], the four-bank organisation of our unmasked baseline [MÖS20, BNAMK21, XL21], the even/odd dual-port layout [WSKB22, CRNTC26] and the hiding layers L_1 and L_2 [HOM06, CMJ23, RPBC20]. What we add is a controlled comparison: one datapath, one flow, one leakage model and one set of positive controls. Our evidence is pre-silicon, and single-trace attacks are outside our threat model (Section 6.6).

2 Preliminaries

This section recalls the ring and NTT structure that FIPS 203 standardises for ML-KEM [Nat24], the Montgomery reduction used in our butterfly and the probing models underpinning the masked design, and states when a leakage sample can combine both shares of a secret.

2.1 NTT structure in FIPS 203

Because $q \equiv 1 \pmod{n}$ but $q \not\equiv 1 \pmod{2n}$, $\mathbb{Z}_q^*$ contains a primitive n-th root of unity but no $2n$-th root, so $X^{256}+1$ factors not into 256 linear terms but into 128 quadratic ones, $\prod_{i=0}^{127}(X^2 - \zeta^{2\,\mathrm{br}(i)+1})$, with $\zeta = 17$ the primitive 256-th root fixed by FIPS 203 and $\mathrm{br}(i)$ the 7-bit reversal of i. This is the *incomplete* NTT of Kyber: the forward transform runs Cooley–Tukey butterflies over 7 stages rather than 8, the layer of degree-1 splits being absent, each stage performing 128 butterflies; the inverse uses Gentleman–Sande butterflies symmetrically. A CT butterfly on (a, b) with twiddle ω computes

$$(a', b') \;=\; \big(a + \omega b,\; a - \omega b\big) \bmod q, \tag{1}$$

with twiddles $\omega = \zeta^k$ precomputed in bit-reversed order. The modulus $q = 3329$ is prime, so $\mathbb{Z}_q$ is a field and every stage is an invertible linear map on $\mathbb{Z}_q^{256}$.

2.2 Montgomery reduction for NTT-friendly primes

Modular multiplication dominates the arithmetic cost of every butterfly. Montgomery reduction [Mon85] replaces the trial division by a precomputed radix R coprime to q, computing $\mathrm{MontMul}(a, b) = a \cdot b \cdot R^{-1} \bmod q$ with multiplications, additions and shifts alone. For the Kyber prime $q = 3329$ we take $R = 2^{16}$ and the signed 16-bit constant $q' = q^{-1} \bmod 2^{16} = -3327$. A single reduction then computes, from a 32-bit product $t = a \cdot b$,

$$m = \big(t \bmod 2^{16}\big) \cdot q' \bmod 2^{16}, \qquad \mathrm{MontRed}(t) = \big(t - m \cdot q\big) \gg 16, \tag{2}$$

the word-level reduction of the reference ML-KEM implementation, with two 16×16 multiplications and a subtraction that map onto DSP48E1 primitives. Reductions specialised to primes of Proth form $q = q_h 2^{\omega} + 1$, which include $3329 = 13 \cdot 2^8 + 1$, trade part of these multiplications for shifts and additions [TKKA24]; we keep the standard reduction

[1] https://github.com/m-lazaro/bastion-ntt, released under the Apache 2.0 licence; the version used in this paper, v4.1, is archived at https://doi.org/10.5281/zenodo.23956329.

because it is common to every configuration we compare. It is pipelined across the five butterfly stages and, the mapping being share-wise linear, replicated in each share domain without cross-domain interaction.

2.3 Probing models

Our countermeasures rest on the t-probing model [ISW03]: a circuit is t-private if the joint distribution of any t probed wires can be simulated without the secret. Additively sharing every intermediate into $d+1$ shares gives a d-private scheme under $t = d$ probes; BASTION targets $d = t = 1$. In hardware a probe on a wire can observe more than its stable value. Glitches can make a combinational output depend transiently on every register in its fan-in [NRR06], and a register that is overwritten leaks the transition between its consecutive values. The robust probing model of Faust et al. [FGM+18] captures both effects by extending each probe: a glitch-extended probe observes every registered value in the combinational fan-in of the probed wire, and a transition-extended probe observes the probed value in two consecutive cycles. We target first-order security in this model, which requires a register barrier wherever a combinational cone could otherwise draw on both shares.

2.4 Domain-Oriented Masking

Domain-Oriented Masking (DOM) [GMK16] is the hardware masking framework we adopt: every share of every intermediate variable is assigned to a *share domain*, with $x = x^{(0)} + x^{(1)} \bmod q$ in the arithmetic case. Its central object is the masked multiplication. For $z = x \cdot y$ the inner-domain products $x^{(i)}y^{(i)}$ are computed locally, while the cross-domain products $x^{(0)}y^{(1)}$ and $x^{(1)}y^{(0)}$ must be re-shared with fresh randomness, $d(d+1)/2$ elements per protected multiplication at order d, and DOM mandates a register as the final step of that resharing, so that glitches in the cross-domain paths cannot reach the output domain. We carry the discipline from Boolean masking in $\mathrm{GF}(2^n)$ to arithmetic masking in $\mathbb{Z}_q$: independent datapaths per share and registers at cross-domain boundaries are unchanged, and only the algebraic operations are reinterpreted. Arithmetic sharing modulo q is how masked lattice implementations protect their linear polynomial arithmetic, in hardware [RSVV15, CRNTC26] and in software [OSPG18]. The linearity of the NTT butterfly lets BASTION avoid the fresh randomness the DOM multiplier requires; we develop this in Section 4.5.

2.5 When does a sample combine both shares?

Let the secret be a vector $f \in \mathbb{Z}_q^n$, for the NTT its input polynomial, shared as $f = F^{(0)} + F^{(1)}$ with $F^{(0)}$ uniform and $F^{(1)} = f - F^{(0)}$. In a share-wise computation built from the butterfly (1) and multiplications by public constants, every value that domain s handles is a deterministic function of residues $a \cdot F^{(s)}$ for public vectors $a \in \mathbb{Z}_q^n$. If every stored value is reduced to $[0, q)$, every value, the wide products of the Montgomery pipeline included, is a function of a single residue. Our butterfly, like the PQClean code it transcribes, reduces lazily, so the integer value of a coefficient after stage s is a function of the 2^s input coefficients that reach it through earlier butterflies, its *support*, and its vectors a are their unit vectors. Collect the residues of domain 0 on which a leakage sample, or a set of samples, depends as $AF^{(0)}$ with $A \in \mathbb{Z}_q^{k\times n}$, and those of domain 1 as $BF^{(1)}$ with $B \in \mathbb{Z}_q^{m\times n}$.

Observation 1. *The joint distribution of $(AF^{(0)}, BF^{(1)})$ is independent of f if and only if the row spaces of A and B intersect only in $\{0\}$.*

Proof. If a nonzero w lies in both row spaces, then $w = \alpha A = \beta B$ for some α and β, and $\alpha\, AF^{(0)} + \beta\, BF^{(1)} = wF^{(0)} + wf - wF^{(0)} = w \cdot f$, a non-constant function of f computable from the pair; the distribution therefore depends on f. Conversely, if the row spaces intersect only in $\{0\}$, the map $F \mapsto (AF, BF)$ has rank $\operatorname{rank} A + \operatorname{rank} B$, so its image is $\operatorname{Im} A \times \operatorname{Im} B$, and since $F^{(0)}$ is uniform, $AF^{(0)}$ and $BF^{(0)}$ are independent and uniform on their images. Then $BF^{(1)} = Bf - BF^{(0)}$ is uniform on $\operatorname{Im} B$ and independent of $AF^{(0)}$, whatever f is. □

Three consequences guide the rest of the paper. First, no function of the values of a single domain depends on f. Second, a sample that adds independent contributions of the two domains, as the power drawn by two disjoint circuits does, can leak only in its higher moments: its mean is the sum of two single-domain means. First-order leakage therefore needs a cell, or a register transition, whose value depends jointly on both domains, which is what our first two positive controls inject (Section 6.3). Third, a sample or a pair of samples that draws on both domains can leak, at any order, only if the forms of the two domains in it are dependent: with full reduction, in practice, when both handle the same intermediate, up to a public factor, or values that one butterfly combines; with lazy reduction, when the supports of their values intersect. Section 6.4 evaluates both conditions cycle by cycle for each schedule.

3 Architecture

This section presents the NTT configurations we compare, all sharing the same five-stage Montgomery butterfly (Section 2.2) and twiddle ROM. Baseline A (Section 3.1) is a single-butterfly transcription of the PQClean reference. Baseline B (Section 3.2) is a four-butterfly datapath with conflict-free banking and L_1/L_2 hiding. The BASTION masked core (Section 3.3) adds first-order masking with one datapath per share, run in lock-step. Section 3.4 derives the staggered and the sequential schedules from the same datapath, and Baseline C, the masked core with one share domain removed, is the unmasked control of Section 3.5. Table 1 summarises the structural differences.

3.1 Baseline A: single-butterfly baseline

Baseline A is a direct hardware transcription of the PQClean reference NTT [PQC]: a sequential finite-state machine (FSM) that iterates over the seven Cooley–Tukey stages of Section 2.1, invoking one butterfly per iteration. It comprises a single butterfly unit, a 256×16-bit true dual-port BRAM, the 128×16-bit twiddle ROM and a lightweight FSM controller. Each butterfly costs $1 + 6 + 1 + 1 = 9$ cycles: one to present the coefficient addresses and twiddle index, six holding control stable while the BRAM latency and the five pipeline stages drain, one to store the two outputs in place, and one to advance the indices per the CT recursion. The seven stages of 128 butterflies give $896 \times 9 = 8{,}064$ cycles, and 8,066 with one cycle to leave the idle state and one to register done. On the

Table 1: Structural summary of the configurations. δ is the delay between the two domains of the staggered core.

Config	BFUs	Memory banks	Banking key	Masking	Share schedule
Baseline A	1	1×256	monolithic	none	n/a
Baseline B	4	4×64	$addr[1{:}0]$	none	n/a
Baseline C	2	2×128	$addr[0]$	none	n/a
Lock-step (BASTION)	4	$2 \times 2 \times 128$	$(share, addr[0])$	$d = 1$	same cycles
Staggered	4	$2 \times 2 \times 128$	$(share, addr[0])$	$d = 1$	δ cycles apart
Sequential	2	2×256	$addr[0]$, share as MSB	$d = 1$	in sequence

target Artix-7 XC7A200T-FBG676-2, Baseline A synthesises to 150 LUTs, 145 FFs, 3 DSP48E1 blocks (all inside the butterfly) and one BRAM tile (one RAMB18 each for the coefficient memory and the twiddle ROM). It closes timing at 217.4 MHz, the critical path running from the coefficient memory to the DSP48E1 input of the butterfly.

3.2 Baseline B: four-butterfly datapath with hiding

Baseline B replaces the monolithic memory and single butterfly of Baseline A with a four-butterfly datapath, integrates the two hiding countermeasures L_1 and L_2 (Section 4) and reduces the cycle count to 2,306, a 3.50× reduction. Multi-butterfly datapaths with conflict-free banking are established prior art [MÖS20, BNAMK21, XL21]; we describe our instance only because its cost is one of our reference points. It keeps the non-overlapping nine-cycle iteration of Baseline A and so does not match the throughput of the streaming designs of Table 4; Section 5.1 states the gap and why our comparison is internal and flow-matched.

Conflict-free banking and the $len = 2$ fallback. The 256-coefficient polynomial is split into four banks of 64×16 bits by the two low address bits. For any stride $len \equiv 0 \pmod 4$ the four operand pairs hit all four banks once, so four butterflies run without a crossbar; the final stage ($len = 2$) breaks this, and a `len2_mode` flag runs stage 7 at two-butterfly parallelism across bank pairs.

Hiding layers. L_1 substitutes LFSR-derived pseudo-random operands for the butterfly inputs during non-computation cycles, and L_2 randomises the execution order of the butterfly groups within each block through an XOR bijection on the group counter. Both are inherited unchanged by every masked configuration and are detailed in Sections 4.3 and 4.4.

Cycle count and implementation results. Stages 1–6 process 128 butterflies in groups of four ($32 \times 9 = 288$ cycles each, 1,728 in total) and stage 7 in groups of two (576 cycles); with the same two cycles the total is 2,306. On the same target, Baseline B synthesises to 624 LUTs (592 logic, 32 packed into SRL16E primitives), 479 FFs, 12 DSP48E1 blocks (three per butterfly) and 2.5 BRAM tiles: each of the four 64×16 banks and the twiddle ROM infers one RAMB18. It closes timing at 222.2 MHz, so the 3.50× cycle reduction becomes a 3.58× latency reduction; the critical path lies in the controller's address generation.

3.3 BASTION masked core

The BASTION masked core adds first-order arithmetic masking over $\mathbb{Z}_q$ on top of the hiding layers of Baseline B. The polynomial is represented by two shares $f^{(0)}, f^{(1)} \in R_q$ with $f^{(0)} + f^{(1)} \equiv f \pmod q$. Share 0 is stored in banks `s0_even` and `s0_odd` and processed by BFU0 and BFU1, share 1 in `s1_even` and `s1_odd` by BFU2 and BFU3, so no bank, butterfly or pipeline register stores values of more than one share. The twiddle ROM is shared, since the twiddle factors are public, and the core injects no fresh randomness beyond the initial share split (Section 4.5). This construction is compliant with the DOM discipline (Section 2.4) adapted to arithmetic masking over $\mathbb{Z}_q$. Figure 1 shows the block diagram.

Stride-invariant even/odd banking. Baseline B distributes a polynomial across four $addr[1{:}0]$-indexed banks and binds each butterfly unit to one bank. This binding is conflict-free only while the stage stride satisfies $len \equiv 0 \pmod 4$: at the final stage ($len = 2$) the

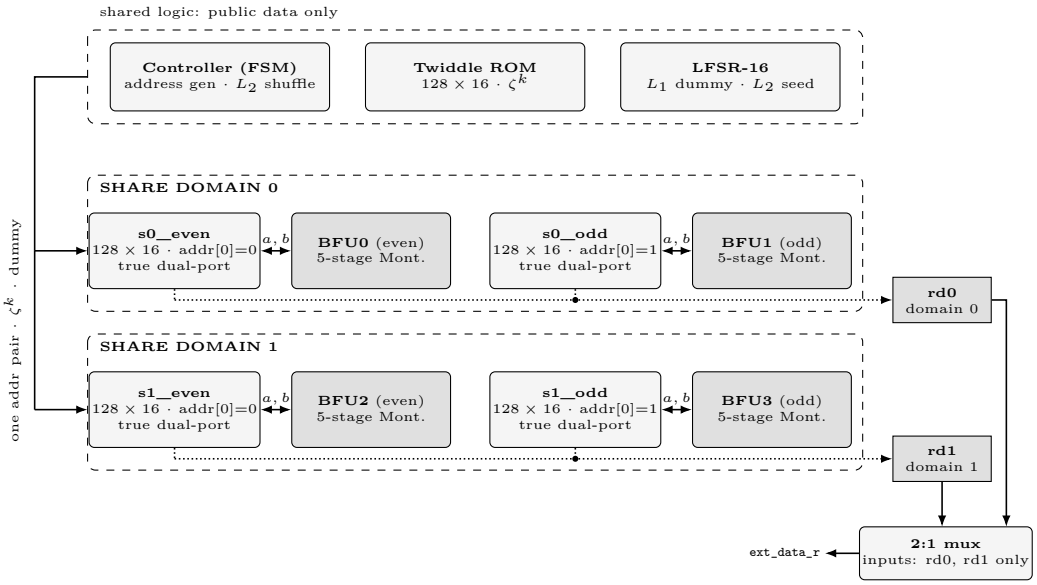


Figure 1: BASTION masked core in its lock-step schedule. The two share domains (dashed boxes) are structurally identical and disjoint: every bank, butterfly and pipeline register belongs to exactly one of them. Both operands of a butterfly share the parity of their index and are served through the two ports of a single dual-port bank at every stride, so one address pair drives all four banks and no crossbar is needed. The controller, twiddle ROM and LFSR are shared and carry only public data. The read-back path (dotted) taps both banks of each domain into that domain's own register, loaded only after the transform has finished and held at zero otherwise, so the 2:1 multiplexer never receives share data from both domains.

operands required by BFU0 straddle `bank0` and `bank2`, so a crossbar and the `len2_mode` fallback are needed. Because each share of the masked core is processed by only two butterflies, a share needs only two banks, and we split them by the least significant address bit, storing coefficient i in bank $i \bmod 2$ at physical address $\lfloor i/2 \rfloor$. Every NTT stage has even *len*, and the controller advances the butterfly index in steps of two, so both operands (j, j+*len*) of a butterfly carry the same parity and reside in the same bank, where they are retrieved concurrently through the two ports of that bank's true dual-port BRAM at the distinct physical addresses $\lfloor j/2 \rfloor$ and $\lfloor (j+len)/2 \rfloor$. The even-indexed butterfly (j, j+*len*) therefore executes entirely inside the even bank, and the odd-indexed butterfly (j+1, j+1+*len*) entirely inside the odd bank. Two consequences follow. Conflict-freedom is provided by the dual-port memory rather than by the bank partition, so it holds at every stage, including $len = 2$, without a fallback mode or a crossbar. And because the odd butterfly's physical addresses coincide with those of the even butterfly, a single address pair emitted by the controller drives all four banks and no bank-index logic exists. This layout is not new (Section 3.5); what matters here is its cost under our flow.

Read-back path. After the transform, an external reader retrieves the two shares through one output port, so some element must select between the domains. In the revised RTL it sees only two registers: `rd0` loads from the banks of domain 0 and `rd1` from those of domain 1, each only when the transform has finished, a read is requested and its own domain is selected, with enable and parity delayed by one cycle so that it captures the addressed word; otherwise it holds zero. Both are therefore zero during the NTT, and

the reader drops the request for two cycles between the shares so that both clear before the other domain is selected. The controller raises done while the last butterfly is still in stages four and five of the pipeline, so the interface also asks the reader to wait two clock edges after done before the first read: the pipeline is then drained before any share word reaches a latch or a read register. In simulation of the six test vectors, the two registers are never non-zero in the same cycle or before the transform ends. The natural alternative, a combinational 4:1 multiplexer on the four bank outputs, is unsafe: since one address pair drives all four banks, it would receive both shares of the same coefficient in every read cycle of the NTT, and a glitch-extended probe on it would observe the unmasked value.

Cycle count. All seven stages run with the same per-share parallelism: two butterflies, one even-indexed and one odd-indexed, per iteration and per share. Each stage processes 128 butterflies per share in 64 iterations of 9 cycles, i.e. 576 cycles per stage and $7 \times 576 = 4{,}032$ cycles of butterfly computation. With one cycle to leave IDLE and one to register done, the total is 4,034 cycles, 1.75× that of Baseline B: the cost of moving from four butterflies on one polynomial (Baseline B, stages 1–6) to two butterflies per share.

Implementation results. On the same target BASTION synthesises to 514 LUTs, 1,037 FFs, 245 slices, 12 DSP48E1 blocks and 2.5 BRAM tiles[2], closing timing at 196.1 MHz with the critical path inside the Montgomery reduction of a butterfly. The `DONT_TOUCH` discipline of Section 4.5 forbids the retiming that Baseline B's synthesis may apply, and we did not pursue frequency optimisation, as determinism across the share boundary takes precedence.

The LUT count is lower than Baseline B's 624 although BASTION carries twice the coefficient memory. This follows from the stride-invariant banking: the masked controller emits one address pair for all four banks, computes no bank index and contains neither the stride-2 crossbar nor the `len2_mode` path. Section 3.5 localises the difference to the controller. The flip-flop count, in contrast, rises to 1,037, against 496 for the masked core synthesised without the preservation attributes: with retiming permitted, synthesis absorbs pipeline registers into DSP48E1 internal registers and packs delay chains into SRL16E primitives, so the 896 architectural registers of the four butterflies report 259 flip-flops and 32 SRL16E LUTs, whereas the attributes materialise all 896 in the fabric, a measured trace of the discipline surviving synthesis.

3.4 Three share schedules

The lock-step core processes both shares of every intermediate in the same cycles. We derive two further schedules from the same datapath, so that only the schedule changes between the three.

Lock-step. One controller drives both domains, so BFU0 and BFU2, and likewise BFU1 and BFU3, execute the butterfly of the same index in the same cycle on the two shares of the same coefficients. The latency is 4,034 cycles.

Staggered. Following HOPE-MLKEM [CRNTC26], the second domain runs the same program δ cycles after the first, and a fresh random bit per execution decides which domain leads. We implement this with a second controller and a second LFSR, loaded with the state of the first at the start and released δ cycles later, so that the lagging domain repeats the address, twiddle, dummy and shuffle sequence of the leading one; the twiddle ROM

[2] Five RAMB18 primitives, one per bank and one for the twiddle ROM, as in Baseline B. A bank holds 2,048 bits of the 18,432 a RAMB18 offers, so doubling the coefficient storage is absorbed within primitives already allocated; at a footprint that filled them, masking would carry a genuine BRAM cost.

serves the second domain through its second port. Both δ and the order bit are inputs sampled at the start, so one netlist covers every delay and the area does not depend on δ. The latency is $4{,}034 + \delta$ cycles, and in simulation the lagging domain repeats the public control of the leading one exactly δ cycles later, for every δ and both orders. We sweep $\delta \in \{1, 4, 9, 18, 576, 4{,}034\}$: inside one nine-cycle iteration, one and two iterations, one stage, and one whole transform, the last being a sequential schedule on two datapaths. The second controller, the second LFSR and the per-domain selection of the public control add 196 LUTs and 123 flip-flops to the lock-step core, for 710 LUTs, 1,160 FFs and 310 slices at 188.7 MHz.

Sequential. Following Reparaz et al. [RSVV15], one datapath transforms share 0 and then share 1. We implement it on the datapath of Baseline C, with each bank deepened to 256×16 bits, the share index as the most significant address bit, and the preservation attributes of the lock-step core, so that the three schedules share one synthesis regime. A register going directly from a value of one share to one of the other would combine both in one transition, with supports that intersect under lazy reduction (Section 2.5), and so would two registers of the pipeline that hold the last butterfly of one pass and the first operands of the next at the same edge. Three flush cycles between the passes, with dummy operands and cleared memory output latches, let the last butterfly of pass 0 leave the five-stage pipeline before the first operands of pass 1 enter it, and the latches are held at zero outside the passes and the read-back; in simulation, no memory latch or read register goes directly from a word of one share to a word of the other. The LFSR state captured at the start is restored at the edge that enters the third flush cycle, so the second pass replays the addresses, twiddles, dummies and shuffle of the first 4,035 cycles later, for a latency of 8,069 cycles. The deeper banks still fit in the same RAMB18 primitives, and the core takes 338 LUTs, 595 FFs and 163 slices at 200.0 MHz. Our second positive control injects the direct transition on purpose (Section 6.3). An earlier version of this core had a single flush cycle; the oracle of Section 6.4 exposed the overlap it left.

3.5 Comparative summary and cost decomposition

Table 2 reports all configurations. Every one is swept in steps of 0.1 ns to the shortest period T closing with non-negative slack under the same scripted flow, with interface paths excluded identically, so $f_{\max}$ is core-only and comparable across rows; the next shorter period fails for every row.

The iso-architectural control. Attributing a cost to masking alone requires a configuration that changes only masking. Baseline B and BASTION differ in two respects at once: BASTION adds share separation, but it also drops from four butterflies on one polynomial to two per share. Baseline C removes that confound. It is the masked core with one share domain deleted, namely the banks `s1_*`, the butterflies BFU2 and BFU3, the share-select port and the second read-back register, and nothing else changed: the same controller, twiddle ROM, LFSR, hiding layers and even/odd banking. Because the two domains of the lock-step core compute in parallel, deleting one leaves the schedule untouched, and Baseline C completes in the same 4,034 cycles. We also report the lock-step core synthesised without the preservation attributes of Section 4.5, which quantifies the cost of the synthesis discipline itself. That configuration is a measurement point only: without the attributes, synthesis may merge or retime logic across the share boundary, which is what the discipline exists to prevent, and we make no security claim for it.

Three cost contributions. The progression separates three effects, each changing one variable. From Baseline A to Baseline B, parallelisation and L_1/L_2 hiding give a 3.50×

Table 2: Post-implementation results on Artix-7 XC7A200T-FBG676-2, Vivado 2022.1. Every configuration is swept to the shortest clock period closing with WNS ≥ 0 under one scripted flow, and $f_{\max} = 1/T_{\text{clk}}$: Vivado places and routes again for every target, so the slack of that implementation does not carry over to a shorter period, and the next shorter period of the sweep fails for every row. Cycles run from the cycle in which start is high to the edge that raises done. LUTs are split into logic and memory (SRL16E), the latter holding pipeline registers rather than combinational logic. ATP is the area-latency product in LUTs and in slices. †Resource measurement control only, not claimed secure; it fails the cell-level checks (Section 4.5). ‡From $\delta = 1$ to $\delta = 4{,}034$.

	T_{clk}	WNS	$f_{\max}$	Cycles	Lat.	LUTs		FFs	Slices	DSP	BRAM	ATP	
Config	(ns)	(ns)	(MHz)		(μs)	log.	mem					LUT·μs	slice·μs
Baseline A	4.600	+0.000	217.4	8,066	37.10	142	8	145	58	3	1.0	5,566	2,152
Baseline B	4.500	+0.010	222.2	2,306	10.38	592	32	479	198	12	2.5	6,475	2,055
Baseline C	4.400	+0.021	227.3	4,034	17.75	313	16	319	105	6	1.5	5,840	1,864
Lock-step, no discipline†	4.400	+0.031	227.3	4,034	17.75	546	32	496	195	12	2.5	10,259	3,461
Lock-step (BASTION)	5.100	+0.017	196.1	4,034	20.57	514	0	1,037	245	12	2.5	10,575	5,040
Staggered	5.300	+0.262	188.7	4,034+δ	21.4–42.8‡	710	0	1,160	310	12	2.5	15.2k–30.4k‡	6.6k–13.3k‡
Sequential	5.000	+0.057	200.0	8,069	40.34	338	0	595	163	6	1.5	13,637	6,576

cycle reduction and a 3.58× latency reduction for 4.16× the LUTs and 4× the DSPs. From Baseline B to Baseline C, the reorganisation gives 0.53× the LUTs and the slices, half the DSPs and three RAMB18 primitives instead of five, at 1.75× the cycles. From Baseline C to the lock-step core, masking proper doubles the DSPs, takes five RAMB18 primitives against three, keeps the cycle count and adds no fresh randomness. Its LUT and frequency cost depends on the synthesis regime, so we report it at matched regime, against the lock-step core synthesised without the attributes: 1.76× the LUTs (578 vs. 329) and 1.86× the slices (195 vs. 105) at an unchanged clock (227.3 MHz for both). Against the delivered core the same step is 1.56× the LUTs (514 vs. 329) and 2.33× the slices (245 vs. 105) at 196.1 MHz; the difference is the price of the synthesis discipline, quantified below, not of masking.

The cost of the schedules. The three schedules share the synthesis regime, so they compare directly. The sequential core needs only 9 LUTs more than Baseline C (338 vs. 329), half the DSPs and three RAMB18 primitives of the lock-step core, and 1.55× the slices of Baseline C, the materialised pipeline registers of the discipline, at 2.27× its latency. The lock-step core buys the unchanged cycle count with the second datapath: 514 LUTs and 245 slices. The staggered core adds to the lock-step core a second controller, a second LFSR and the per-domain selection of the public control, 1.38× its LUTs and 1.27× its slices, and δ cycles. By area-latency product in slices the lock-step core is the cheapest of the three (5,040), the sequential core follows (6,576), and the staggered core lies between 6.6k and 13.3k depending on δ.

Where the LUT difference lives. Between Baseline B and the lock-step core without the discipline, both with four butterfly units, the butterflies cost the same 256 LUTs and the controller falls from 348 to 279: the crossbar, the `len2_mode` fallback and the bank-index logic, partly spent again on the read-back path and the write multiplexers of twice as many banks, for a net 46 LUTs. This follows from the banking, not from masking.

On the synthesis discipline. Enforcing the separation costs 13.7% of the clock frequency (227.3 to 196.1 MHz), 2.09× the flip-flops and 1.26× the slices. The accompanying 64-LUT reduction is not a saving: `DONT_TOUCH` forbids packing pipeline registers into SRL16E primitives and DSP48E1 internal registers, so they are materialised as fabric flip-flops. Slices, which LUTs and flip-flops share, are the fairer metric, and they remove

the apparent advantage over Baseline B: 195 slices against 198 at matched regime, 245 with the discipline. What the discipline buys is shown by the cell-level checks of Section 4.5.

Baseline C as a design point. Baseline C is not dominated by Baseline B: about half the LUTs and slices and half the DSPs at 1.71× the latency give it a lower area-time product, in LUTs and in slices. Neither its even/odd banking nor the stride invariance that comes with it is new. Storing two coefficients per memory word and serving two butterflies through the two ports of a dual-port memory is an established layout [WSKB22], and because every stage of the ML-KEM NTT pairs coefficients of equal parity, the transform splits into independent even and odd halves, which HOPE-MLKEM [CRNTC26] exploits by packing both halves into one memory word. What this comparison adds is the cost of that choice under one flow: against the four-bank organisation of Baseline B it removes the crossbar and the `len2_mode` fallback, at half the butterfly parallelism per polynomial.

Other parameter sets. All ML-KEM parameter sets share the same NTT [Nat24]; only the number of transforms per operation changes with the module rank, so Table 2 holds at every security level.

4 Threat Model and Countermeasures

This section states the threat model, details the four countermeasure layers L_0 to L_3 integrated into BASTION, and develops the security argument for the masking layer L_3.

4.1 Threat model and countermeasure taxonomy

We target first-order security in the glitch- and transition-robust probing model [FGM+18] (Section 2.3), for a passive adversary who observes power or EM emanations while the NTT runs. Whenever a schedule handles, in one cycle, values of the two domains whose forms are dependent, such as both shares of an intermediate, a univariate second-order adversary can combine them in one sample; otherwise a bivariate adversary must combine two samples (Observation 1). A first-order core is not expected to resist either, but the schedule decides which of the two applies and at what cost, so we measure both rather than exclude them (Section 6). Single-trace and few-trace attacks combine the leakage of many intermediates of one execution through belief propagation: they recover the secret from a single trace [PPM17, PP19] and, with chosen ciphertexts that make the input of the inverse NTT sparse, the long-term key of Kyber from a few traces [HHP+21]. First-order masking does not prevent them, since it counters averaging over many traces, and the attacks of [PPM17] and [HHP+21] target masked implementations. Our shuffling layer only raises their cost, and we make no claim against them. Fault attacks on Kyber [PP21, Del22], for which fault-tolerant FPGA designs have been proposed [CMBM26], and decapsulation-oracle attacks are out of scope.

Table 3 summarises the four layers; L_1 and L_2 are integrated into Baseline B and inherited by every masked configuration. Leakage assessment is not a layer of the design; Section 6 reports it.

4.2 L_0: constant-time execution

L_0 requires the sequence of control states and the cycle count to be independent of the secret input. In BASTION this is a direct property of the FSM controller (Section 3.3): the iterations per stage and the number of stages are fixed by the polynomial degree $n = 256$ and the *len* schedule, neither depending on the coefficients, and the write-back path has no conditional at all: the signed Montgomery reduction (2) returns a value in

Table 3: BASTION countermeasure taxonomy.

Layer	Name	Type	Primary threat mitigated
L_0	Constant-time execution	Discipline	Timing leakage
L_1	Dummy-operation insertion	Hiding	SPA, SEMA
L_2	Scheduling shuffle	Hiding	Single-trace SCA (partial)
L_3	First-order arithmetic masking	Masking	First-order DPA/DEMA

$(-q, q)$ without a final correction, and the butterfly reduces lazily (Section 2.5). Every run of the lock-step core completes in exactly 4,034 cycles, which we confirmed on the six test vectors and on each of the 10^6 executions of its leakage campaigns, about half of them on one fixed input and the rest on random ones.

4.3 L_1: dummy-operation insertion

L_1 is a hiding countermeasure [HOM06, MOP07] that prevents the butterfly operands from collapsing to a fixed constant during the cycles in which no live butterfly is issued. Without it, the inputs are driven from the banks only during WAIT and forced to zero otherwise, so the profile alternates between operand-dependent activity and a fixed zero pattern that an SPA adversary can use to delimit computation cycles within each FSM iteration. L_1 replaces that constant with pseudo-random data. A single 16-bit LFSR [Alf96] produces one word per cycle, from which the streams `dummy_a`, `dummy_b` and `dummy_w` are derived as bit permutations (identity, byte swap, nibble rotation); an input multiplexer selects, per butterfly, between the live operand and the dummy operand under the `bf_active` signal. The butterfly pipeline has no clock enable, so the substitution acts on the data presented to the datapath, not on register activation.

Residual profile. The three dummy streams share the Hamming weight of their common word, which independent LFSR taps would avoid.

4.4 L_2: scheduling shuffle

L_2 randomises the order in which butterflies execute within each NTT block, following the shuffling countermeasures proposed for the NTT [RPBC20, CMJ23]: a single-trace attacker then no longer knows which butterfly produced each leakage sample. A block at stage length *len* contains $num_groups = len/p$ groups of p butterflies sharing a twiddle factor ($p = 4$ for Baseline B, $p = 2$ for the masked cores). The controller maintains a group counter gc and executes the permuted group

$$gs \;=\; gc \;\oplus\; \big(mask \wedge (num_groups - 1)\big), \tag{3}$$

with *mask* drawn afresh from the LFSR at each block start. Since num_groups is a power of two, the XOR in (3) is a bijection for any *mask*: every group runs exactly once, preserving the result.

Injected entropy. Over a forward transform the XOR masks span a permutation space of 120 bits, the final stage, with a single group per block, contributing none. All masks come from one free-running 16-bit LFSR without reseed, however, so the schedule is fixed by the LFSR state at invocation and carries at most about 16 bits. Since masking does not counter single-trace attacks (Section 4.1), the shuffle is the only layer against them, and with 16 bits it is weak.

4.5 L_3: first-order arithmetic masking

We implement a $d = 1$ arithmetic masking over $\mathbb{Z}_q$, adapting the Boolean DOM construction of Groß et al. [GMK16] to arithmetic shares, and develop three structural properties of the masked NTT: spatial separation of the two share domains, linear butterfly operations that eliminate cross-domain products, and public twiddle factors that permit ROM sharing.

Sharing scheme. Every secret $f \in R_q$ is encoded as $(f^{(0)}, f^{(1)})$ with $f \equiv f^{(0)} + f^{(1)} \pmod q$: a uniform $f^{(0)}$ is drawn externally from a TRNG and $f^{(1)} = f - f^{(0)} \bmod q$. The core receives all coefficients of share 0 and then all of share 1, so that the input bus never carries the two shares of one coefficient in consecutive cycles, and stores them in disjoint banks. The input bus reaches the write multiplexers of both domains, so the loader returns it to zero after the last word; otherwise it would hold a share-1 coefficient at an input of domain-0 cells throughout the transform. TRNG design and share refresh are orthogonal; we assume uniform initial sharings, following [OSPG18, LMRBG24].

Spatial domain separation. The datapath is physically partitioned into two domains, $\mathcal{D}^{(0)}$ (banks `s0_*`, BFU0, BFU1 and read register `rd0`) and $\mathcal{D}^{(1)}$ (banks `s1_*`, BFU2, BFU3 and `rd1`). Every bank, butterfly, pipeline register and read register belongs to exactly one domain, and no butterfly reads an operand from the other domain's banks. The only element that draws on both domains is the 2:1 read-back multiplexer, whose two inputs are the read registers; both are constant zero during the NTT, and at most one of them is non-zero at any time (Section 3.3). Under the robust probing model at first order, a glitch-extended probe on any wire of the datapath therefore observes values of a single domain, and a probe on the multiplexer observes registers that carry no share data while secret-dependent intermediates are present. After the transform the multiplexer passes the shares one at a time, as the interface requires.

Enforcing and verifying the separation. The separation is backed by Vivado attributes: `KEEP_HIERARCHY` on `ntt_top`, `memory_bank` and `butterfly` prevents boundary flattening, and `KEEP`/`DONT_TOUCH` on the share-boundary butterfly and bank outputs, on the read registers and on the pipeline registers forbids register merging, resource sharing and retiming across the boundary (full attribute list in the artifact). Attributes express intent, not outcome, so we discharge the claim on the post-implementation netlist directly, with three queries. Let $\mathcal{C}^{(0)}$ and $\mathcal{C}^{(1)}$ be the cell sets of the two domains. First, a static timing query for all paths from $\mathcal{C}^{(0)}$ to $\mathcal{C}^{(1)}$, and the symmetric one, must return none; static timing enumerates register-to-register paths irrespective of logic depth, so this excludes any combinational path between the domains. Second, a query from both sets to every register outside them must return none, which excludes a path through a shared register over several cycles. Third, static timing does not cover cells outside both sets, such as the read-back multiplexer, so we list every primitive whose combinational input cone contains registers of both domains; the list must contain only that multiplexer and the output buffers it drives, with `rd0` and `rd1` as their only domain inputs. A design-wide check additionally must report no combinational loops, no latch loops, no unconstrained internal endpoints and no register pins without a clock, excluding an unclocked path that escapes the queries.

On the routed netlist of the lock-step core, each domain holds 472 sequential cells (flip-flops, the RAMB18 of its banks and its DSP48E1 blocks) and 110 lie outside both, in the controller, the LFSR, the twiddle ROM and the read enables. The first two queries return no path, the third lists exactly the 16 output LUTs of the read-back multiplexer and the 16 output buffers they drive, with `rd0` and `rd1` as their only domain inputs, and the design-wide check reports none of the four conditions. The staggered core passes the

same checks, with 472 sequential cells per domain and 233 outside, its second controller and LFSR included.

The checks can fail, and do on the lock-step core synthesised without the attributes (Section 3.5), in two ways. Synthesis packs part of each butterfly pipeline into SRL16E primitives and adds a short chain of registers that tracks the reset for them; these chains hold the same value in every butterfly, and the optimiser merges them, so that BFU2 and BFU3 use the one of BFU1. The first query then reports 32 timing paths from that register of domain 0 into the pipeline of domain 1, and the third 64 primitives of domain 1, the gating LUTs and the registers after them, whose cone contains it. The register carries the reset, not share data, but it is a cell shared across the boundary. And the physical optimiser moves the product register of BFU0 and of BFU3 out of their DSP48E1 blocks into fabric flip-flops that it places in the hierarchy of the controller, so the second query reports 64 paths from the two domains into cells that are meant to carry public data only. Neither is visible in the RTL; both are what the discipline exists to prevent and what the checks exist to find.

Linearity and the absence of fresh randomness. The simplification BASTION exploits, relative to a generic DOM multiplier, is that every operation in the butterfly is linear over $\mathbb{Z}_q$: the CT butterfly $(a', b') = (a + \omega b,\ a - \omega b) \bmod q$ combines a multiplication by a public constant with modular addition and subtraction. Applied share-wise, a linear map L gives $L(x^{(0)}) + L(x^{(1)}) = L(x)$ at no randomness cost [ISW03, GMK16]: every value inside a domain is a function of that domain's share alone, and by Observation 1 such values are independent of the secret. This contrasts with the bilinear gadget $(x \cdot y)^{(i)}$, whose cross-domain products $x^{(0)}y^{(1)}$ and $x^{(1)}y^{(0)}$ require fresh randomness (Section 2.4). The masked butterfly therefore decomposes into

$$(a')^{(s)} = a^{(s)} + \omega b^{(s)}, \quad (b')^{(s)} = a^{(s)} - \omega b^{(s)}, \quad s \in \{0, 1\}, \tag{4}$$

computed in disjoint butterflies on disjoint banks, with no fresh randomness injected anywhere in the pipeline. Reparaz et al. [RSVV15] exploit the same linearity in masked Ring-LWE decryption but process the two shares in sequence on a single coprocessor, noting that two coprocessors could run them in parallel when lower latency matters more than compactness. The lock-step core takes that parallel option, and the staggered and sequential cores of Section 3.4 cover the other two.

Public twiddle sharing. Twiddle factors $\omega = \zeta^k$ ($\zeta = 17$, $k \in \{0, \ldots, 127\}$) are fixed by FIPS 203 [Nat24] and stored in ROM; a probe on its output sees a function of the cycle and butterfly index, both public, so sharing the ROM between domains introduces no side channel.

Glitches, transitions and scope. DOM requires a register barrier so that glitches cannot carry share information across domains [GMK16, NRR06]. In BASTION no combinational cone of the datapath draws on both domains, so glitches and register transitions combine only values of one domain, which Observation 1 shows to be independent of the secret, and the two-cycle gap of the read protocol prevents a share-0 to share-1 transition at the output. The argument covers first-order probes in the robust probing model, not couplings outside it, such as the supply network, routing or the inside of the DSP48E1 primitives, nor the higher-order leakage of a sample that integrates both domains, which the schedules change and Section 6 measures.

Table 4: Comparison against unmasked NTT accelerators for ML-KEM. NTT op. lists the operations a unit supports; cycles are per transform. Latency is cycles/$f_{\max}$; ATP is LUTs×latency [LUT·μs]. †Hiding only ($d = 0$); NTT/INTT module (230 LUTs, one RAMB18) and shared multiply-and-reduce unit (262 LUTs, 2 DSPs) of the ePrint version (2021/1189), at the 250 MHz of the full design. ‡Server-class; latency is amortised throughput after fill-up.

Work	FPGA	NTT op.	Cycles	$f_{\max}$ (MHz)	LUTs	DSPs	BRAM	ATP (LUT·μs)
Xing–Li [XL21]	Artix-7	NTT+INTT+PWM	512	161	1,737	2	3	5,524
Bisheh-Niasar et al. [BNAMK21]	Artix-7	NTT+INTT	324	222	801	4	2	1,169
Kundi et al. [KBMSH24]	UltraScale+	NTT+INTT	258‡	322	3,821	20	5	n/a
Sonbul et al. [SRM+26]	Artix-7	NTT (FNTT)	1,029	200	503	1	2	2,588
Bertels–Verbauwhede [BV25]	Alveo U55c‡	NTT	16‡	400	67,210	0	0	n/a
Jati et al. [JGCS24]†	Artix-7	NTT+INTT	570	250	492	2	0.5	1,122
Baseline A	Artix-7	NTT	8,066	217.4	150	3	1.0	5,566
Baseline B	Artix-7	NTT	2,306	222.2	624	12	2.5	6,475
Baseline C	Artix-7	NTT	4,034	227.3	329	6	1.5	5,840

5 Implementation Results and Comparison

This section compares the configurations of Table 2 against the unmasked and the side-channel protected state of the art in NTT hardware for ML-KEM and related lattice schemes. All comparisons use the figures reported by the original authors; we do not re-synthesise prior work. The ATP columns report the area-time product LUTs × latency [LUT·μs], with latency computed as cycles/$f_{\max}$.

5.1 Comparison with unmasked designs

Table 4 compares Baselines A, B and C against representative unmasked NTT accelerators for ML-KEM and related schemes from 2021 to 2026, on Artix-7 and on server-class devices; the FPGA family is stated explicitly, as LUT comparisons across families require caution. Jati et al. [JGCS24] appears here rather than in the protected table because, despite its hiding countermeasures, it implements no arithmetic masking ($d = 0$).

Footprints span two orders of magnitude, so LUT figures must be read against each operating point. No entry carries a countermeasure beyond hiding, and the TVLA pass of Jati et al. [JGCS24] at 2×10^5 traces does not rule out attacks that average over more traces. Our baselines do not match the best unmasked figures: the controller holds the addresses while the BRAM latency and the pipeline drain and does not overlap iterations, so Baseline B issues about 0.39 butterflies per cycle, whereas the core of Bisheh-Niasar et al. [BNAMK21] completes the 896 butterflies of a transform in 324 cycles, for an ATP 5.5 times below Baseline B's and 9.0 times below the lock-step core's; run once per share it would take about 648 cycles. The comparison of schedules does not depend on closing that gap, since all our configurations share the iteration, the butterfly and the synthesis script; a streaming datapath would shift them together.

5.2 Comparison with side-channel protected implementations

Table 5 compares BASTION against side-channel protected designs, separating NTT-aligned entries, which are directly comparable, from full-system cores, which are context. Land et al. [LMRBG24] target Streamlined NTRU Prime rather than ML-KEM.

NTT-aligned comparison. Among NTT-only designs, the closest is the local-masking core of [CVBB24], which differs from BASTION in three respects. *Mechanism:* it protects the twiddle and leaves the data polynomial in the clear [RPBC20], whereas BASTION

Table 5: Comparison against side-channel protected hardware. All entries are first-order ($d = 1$); ATP is LUTs×latency [LUT·μs], reported for NTT-aligned designs only. §NTRU Prime (`sntrup761`), not ML-KEM. ¶Full KEM decapsulation, not NTT-only. ⋆Leakage detected at u=1 in every stage, attributed to non-surjective masking, and at u=128 only for an input that creates a zero intermediate [CVBB24]. °TVLA of individual modules, not of the whole decapsulation. ◇Kyber768 (NIST level III); LUTs of the whole RISC-V platform with its masked accelerators; cycles include randomness generation. ‡XC7A200T-3; area of the whole configurable first-order KEM, masked Keccak included, randomness generator excluded; cycles of ML-KEM-512 decapsulation; TVLA measured on a SAKURA-X board for ML-KEM-768, 10^6 traces per set. †Pre-silicon: first-order TVLA on the whole-design toggle model of Section 6, five independent pairs of runs of 10^5 traces, not a hardware measurement. ♯$\delta = 18$, the smallest delay of our sweep at which the form view of the oracle separates the domains (Section 6.4).

Work	FPGA	Scope	Cycles	$f_{\max}$ (MHz)	LUTs	ATP (LUT·μs)	TVLA	Traces
NTT-aligned: directly comparable to BASTION								
LocalMasking [CVBB24] (u=1)	Artix-7	NTT+INTT	467	167.4	3,651	10,187	✗⋆	10^6
LocalMasking [CVBB24] (u=128)	Artix-7	NTT+INTT	1,613	167.4	3,651	35,186	✗⋆	3.4×10^5
BASTION lock-step	Artix-7	NTT	4,034	196.1	514	10,575	✓†	10^5
BASTION staggered♯	Artix-7	NTT	4,052	188.7	710	15,248	✓†	10^5
BASTION sequential	Artix-7	NTT	8,069	200.0	338	13,637	✓†	10^5
Full-system: masked Kyber or lattice KEM cores (context)								
Kamucheka et al. [KNAH22]	Virtex-7	Full decap¶	137,738	100	152,860	n/a	✓°	4×10^4
Fritzmann et al. [FVB+22]◇	Artix-7	Full decap¶	1.40×10^6	58	29,889	n/a	✓°	10^5
Land et al. [LMRBG24]§	Artix-7	Full decap¶	1.87×10^6	200	19,923	n/a	✓°	10^7
HOPE-MLKEM [CRNTC26]	Artix-7‡	Full decap¶	9.0×10^3	175	43,473	n/a	✓	10^6

shares the coefficients themselves, as in [RSVV15, OSPG18], and processes each share on its own datapath. *Security:* its TVLA detects leakage in every stage at u=1, attributed to the non-surjectivity of multiplicative masking over $\mathbb{Z}_q$, and at u=128 for inputs that create a zero intermediate, which no multiplicative mask hides; the same authors later attacked local masking with soft-analytical methods [CBVB25]. BASTION's evaluation is pre-silicon (Section 6). *Resources:* the lock-step core uses 514 LUTs against 3,651 (7.1× fewer) for an NTT-only scope; with 8.6× the cycles, its ATP is 4% higher than that of local masking at u=1 and 3.3× lower than at u=128.

Full-system comparison. HOPE-MLKEM [CRNTC26] is the closest design in topology: at first order it gives each share its own butterfly unit and memories and starts the two units a configurable delay apart in a random order, the schedule our staggered core reproduces; its first-order evidence is a TVLA on a SAKURA-X board at 10^6 traces, whereas ours is pre-silicon. Its published figures cover the whole first-order KEM, 43,473 LUTs at 175 MHz on Artix-7 with a threshold-implemented Keccak, and one butterfly unit of its unprotected design, 1,513 LUTs and 517 cycles per NTT at 209 MHz; neither isolates the cost of masking the NTT, so Table 5 lists it as context. Our lock-step core uses 514 LUTs for both shares but needs 4,034 cycles per transform, because its iterations do not overlap (Section 5.1). The two designs answer different questions: HOPE-MLKEM shows that a fully masked KEM is practical in hardware, while BASTION measures what each share schedule of a masked NTT costs and exposes.

6 Security Evaluation

This section evaluates the three schedules with one pre-silicon harness. We describe the tests (Section 6.1) and the leakage model (Section 6.2), calibrate the harness with injected

flaws (Section 6.3), predict with an oracle where each schedule can combine both shares (Section 6.4), and report the results (Section 6.5) before stating the limitations of the evidence (Section 6.6).

6.1 Tests

Non-specific TVLA [GJJR11, SM16] partitions traces into a fixed group, whose input is one fixed polynomial, and a random group, whose input is uniform and fresh in every trace, and computes at each sample Welch's two-sample statistic

$$t = \frac{\bar{\mu}_F - \bar{\mu}_R}{\sqrt{\sigma_F^2/n_F + \sigma_R^2/n_R}}, \tag{5}$$

with $\bar{\mu}, \sigma^2, n$ the per-group mean, variance and trace count. We apply three tests to the same traces. The *first-order* test uses the samples as they are. The *univariate second-order* test centres each sample on the mean of its group and squares it before applying (5) [SM16]; it detects a dependency on the secret in the variance of a single sample, which is how a sample that adds the activity of both domains leaks (Section 2.5). The *bivariate second-order* test centres the samples at cycles c and $c + \Delta$ on their group means and applies (5) to their product, with Δ the offset between the domains: δ for the staggered core and the 4,035 cycles between the two passes of the sequential core. We use the conventional threshold $|t| > 4.5$ [SM16]. To keep false positives over a long window low, a campaign is a pair of runs on independent traces, and leakage is declared at N traces only when both runs, each on its first N traces, exceed the threshold at the same sample. We evaluate N on the grid $\{500, 1{,}000, 2{,}000, 3{,}000, 5{,}000, 7{,}500, 10{,}000\}$ followed by steps of 10,000 up to 100,000, and repeat every campaign as five independent pairs of runs of 10^5 traces. For each test we report in how many of the five pairs leakage is detected, the median and range of the N at which a pair first detects it, a pair without detection counting as more than 10^5, and the median and range of the largest $|t|$ of the ten runs at 10^5 traces. A pass at N traces only shows that nothing is detected within the power of N traces and the chosen model; 10^5 traces per run are over a hundred times the largest count at which a positive control is first detected at first order, 750 traces, and 50 times the largest at univariate second order, 2,000 traces (Section 6.3).

6.2 Leakage model, window and simulator

Whole-design toggle model. Each trace has one sample per clock cycle, equal to the number of bits that toggle at that clock edge in every register and every BRAM output latch of the design, in both share domains and in the shared control: 1,187 bits in 78 registers for the lock-step core, 1,331 in 99 for the staggered core, which adds a controller and an LFSR, and 677 in 51 for the sequential core. The contents of the memory arrays are not part of the model; what a read exposes is captured by the output latches. This Hamming distance between consecutive register states is the standard first-order proxy for the dynamic power of CMOS logic [MOP07], and summing it over the whole design is what a power measurement does: the two domains are not observed separately, so a sample in which both domains handle the same intermediate combines both shares. The model is evaluated at register transfer level, so it has no glitches; Section 6.6 discusses this.

Window. The window opens with the first cycle in which share 0 is written into the core and closes with the last cycle of the read-back of share 1, so it covers loading and read-back as well as the transform, the phases in which the shares pass through the interface: 5,070 samples for the lock-step core, $5{,}070 + \delta$ for the staggered core and 9,105 for the sequential core. The reader waits two edges after done before the first read (Section 3.3), and the

read-back is pipelined, one word per cycle. The core is reset and its LFSR reseeded to the same value before every trace (Section 6.6, limitation (ii)); the coefficient memories are not reset, so each trace starts from what the previous one left.

Groups and shares. Each trace joins the fixed or the random group by a fresh coin flip, so that what earlier executions left in memory is independent of the group. In both groups, share 0 is drawn uniformly from $\mathbb{Z}_q$ and afresh for every trace and share 1 completes the sharing, so the two groups differ only in the unshared input. In the staggered core the order bit is also drawn afresh for every trace. Every execution draws its inputs from its own generator, seeded by the run and the index of the execution. A campaign is simulated in jobs, and a job first replays, without recording it, the execution before its first one: an execution starts from the memory contents the previous one left, which depend only on the previous inputs, since loading and the transform overwrite every address. The traces of a run are therefore the same however it is split into jobs.

Simulator. The model is defined by a VPI module for Icarus Verilog 12 that, after every rising edge, reads every register of the design and adds up the bits that changed. Campaigns run on Verilator 5, on which the testbench computes the same count over the same register list, taken from the VPI module; on 10 executions of every configuration the two simulators give identical traces and identical read-backs, the only exception being the loading phase of the first execution of C1, where Icarus propagates the unknown contents of the uninitialised memories through the adder that C1 injects and Verilator, a two-state simulator, sees zeros. All stimulus changes on the falling edge, so the traces do not depend on the event order of the simulator. Every execution is checked: its two read-back shares must recombine to the transform of its input computed by an independent model of FIPS 203, and its window must have the same number of samples as every other execution of its schedule, which also confirms the constant cycle count of L_0 on every input. The toggle counts are integers, so the raw moments that the three tests need, up to the fourth order and, for the bivariate test, the mixed moments of the sample pairs, are accumulated exactly in integer arithmetic over the executions in order. At every point of the grid the centred statistics are formed from them exactly, centring on the group means of the first N traces [SM16]; this gives the same values as two passes over the traces, in one.

6.3 Calibration with injected flaws

A pass is informative only if the harness can fail, and failing on an unmasked design is a weak check, since every coefficient is then in the clear. We therefore derive three variants of the lock-step core, each with a single deliberate flaw of a kind a masked design can actually contain, and run the same campaign on each, five independent pairs of runs of 2,000 traces (Table 6). C1 adds a 16-bit register to the 1,187 bits of the model and C2 a 16-bit register and its 1-bit phase, so their leakage has to emerge from the switching noise of the whole core, the shares included. In C2 only the transition of the register combines the shares, and in C3 each domain on its own carries information on the secret. Every pair detects every control at first order, the hardest, C1, within 750 traces, and at univariate second order within 2,000. Baseline B, which processes the coefficients in the clear, is a basic check of the toolchain: every pair detects it at the first point of the grid, 25 traces, with $|t|$ near 100. Every execution of the control campaigns recombines correctly, so the flaws change the leakage and nothing else.

Table 6: Positive controls: single injected flaws in the lock-step core. For the first-order and the univariate second-order test, the median and range over five independent pairs of runs of the smallest trace count of the grid $\{25, 50, 100, 150, 200, 300, 400, 500, 750, 1{,}000, 1{,}500, 2{,}000\}$ at which both runs of a pair exceed $|t| = 4.5$ at a common sample. Every pair detects every control in both tests.

Control	Injected flaw	What detection shows	1st	2nd
C1	A register loads, every cycle, the sum of the port-A outputs of the share-0 and share-1 even banks	Two shares meet in one cell	400 (300–750)	1,500 (1,500–2,000)
C2	A register loads the port-A output of the share-0 and of the share-1 even bank in alternate cycles	Transitions are captured	300 (200–300)	1,500 (1,000–2,000)
C3	Each coefficient of share 0 is drawn from $\{0, \ldots, 255\}$ instead of $\mathbb{Z}_q$	Weak randomness is detected	100 (100–150)	750 (750–750)

6.4 When do the shares meet?

The addresses, the shuffle and the dummy operands are fixed by the public schedule and the LFSR state, so for every register and BRAM output latch the residues on which its value depends at each cycle are fixed too (Section 2.5). We measure them on the RTL itself, not on a model of it. A base execution is followed by 512 executions that each add 1 modulo q to one input coefficient of one share, every one of them preceded by a base execution so that all start from the same memory contents. For a register whose value is linear in the shares modulo q, which is every register of the datapath except the Montgomery intermediate m, loaded at the same edge as t from the same product, the difference to the base execution modulo q is the coefficient of its form for that input coefficient; for every register, a non-zero difference puts the coefficient in its support. This gives two views: the *form view* keeps each value's linear form, as a design that reduces every value to $[0, q)$ would expose, and the *support view* the unit vectors of its support, as our lazy butterfly exposes. The sample of an edge depends on the old and new values of the registers that toggle at it in some execution; for these, the oracle checks whether the residues of the sample determine a non-zero combination $w \cdot f$, by a rank computation over $\mathbb{Z}_q$ in the form view (a register depending on both shares, as in C1, is handled by the same computation) and a set intersection in the support view, and for the bivariate test it takes the union of the samples c and $c + \Delta$. By Observation 1, a sample that the support view does not flag cannot leak at any order in this model. The form view flags a subset: the samples at which a design that reduced every value to $[0, q)$ could leak, so a leaking sample outside it is evidence that lazy reduction matters.

What the oracle found, and the fix. On the first version of the cores the oracle showed that the datapath went on before the pipeline had drained, in two places. The controller raises done while the last butterfly is still in stages four and five, so a read-back started at done met it: in the lock-step core the flagged range reached into the read-back, and in the staggered core at $\delta = 18$ and $\delta = 4{,}034$ the form view flagged the first edge of the read-back, where one share is read while the last butterfly of the other leaves the pipeline. And the sequential core had a single flush cycle, so at the two edges after it the last butterfly of pass 0 was still in the pipeline when the first operands of pass 1 arrived; the support view flagged those two samples and the first edge of the read-back. We therefore

lengthened the flush to three cycles (Section 3.4) and made the reader wait two edges after done (Section 3.3). Every result below is for the revised cores.

Flags of the revised cores. Table 7 lists the samples each view flags for the univariate and the bivariate test. For the lock-step core both views flag the 4,034 samples, 80% of the window, from the edge at which the first live operands enter the butterflies to the second edge after done, at which the last butterfly leaves them: whenever a butterfly is live, both domains hold the same intermediates. Nothing is flagged during loading or read-back, which confirms that the read-back registers keep the shares apart. In the sequential core neither view flags any sample; for the bivariate test at the 4,035-cycle offset the form view flags 4,035 pairs, the whole transform, and the support view 4,293.

One might expect the staggered core to separate the domains once δ exceeds the time a butterfly spends in the datapath, about seven cycles; the oracle shows otherwise. At $\delta = 1$ and $\delta = 4$, inside one iteration, the domains still handle the same butterfly in overlapping cycles, and both views flag 4,033 and 4,030 samples. At $\delta = 9$, one iteration, the form view still flags 2,684: when the lagging domain loads the operands of a butterfly, the leading one still holds the results of that butterfly in its output registers and bank latches, and the results are linear combinations of the operands. At $\delta = 18$, two iterations, the form view flags no sample, while the support view still flags 1,262 through lazy reduction. At $\delta = 576$, one stage, the form view flags 1,814 samples, in the iterations where one domain reads at a stage the coefficients that the other handles at the previous stage: the read-after-write coincidences of the stage boundaries. At $\delta = 4{,}034$, where the transforms do not overlap, neither view flags any sample. The univariate flags are the same whichever domain leads, which we checked for every δ; the bivariate flags depend on the order, and since the order is drawn afresh for every trace, Table 7 gives their union over both orders. With the LFSR reseeded (limitation (ii) of Section 6.6) the flagged samples fall on the same cycles in every trace; in deployment the shuffle would move the coincidences that depend on its order between executions. The bivariate test at offset δ is flagged on the whole transform for every δ, since the lagging domain repeats the leading one.

6.5 Results

Table 7 reports the three tests for every schedule; every one of the 8.08×10^6 executions of the campaigns and the controls recombines to the reference transform.

First order. No pair detects first-order leakage in any schedule up to 10^5 traces, the largest point of the grid. Single runs exceed the threshold at isolated samples, but never at the same sample as the other run of their pair; at 10^5 traces the largest $|t|$ of any run is 4.6.

Second order, from lock-step to sequential. The schedules form a scale. At one end, the lock-step core leaks in single samples: every pair detects it, from 2,000 traces in the median, with $|t|$ near 32 at 10^5 traces. At the other end, the sequential core and the staggered core at $\delta = 4{,}034$, whose transforms do not overlap, show no univariate leakage up to 10^5 traces, and the oracle flags no sample for them; the largest single-run $|t|$, 5.1 in the sequential core, falls on a different sample in each run that exceeds the threshold. These two schedules leak instead at the pair of samples their offset relates: every pair detects the bivariate test, from 1,000 traces in the median, with $|t|$ near 47 at 10^5 traces. In between, the staggered core with δ from 1 to 576 leaks in both tests. In single samples its median $|t|$ at 10^5 traces lies between 6.5 and 25.8, below that of the lock-step core at every delay but not decreasing with it: the leakage is weakest at $\delta = 9$ and 18, detected from 60,000 and 40,000 traces, and stronger again at $\delta = 576$, through the read-after-write

coincidences of the stage boundaries. The oracle states where a sample can depend on the secret, not how strongly: it flags more samples at $\delta = 9$ than at $\delta = 576$, but there a sample combines the results of a butterfly in one share with its operands in the other, not both shares of one value. At the pair of samples δ apart, the staggered core leaks with $|t|$ between 24.5 and 34.7. Separating the domains in time therefore does not remove second-order leakage from this model; it moves it from single samples to pairs of samples a public distance apart. The random order of the staggered core does not prevent this, since the product of the two samples is symmetric in the order. Combining the samples requires the adversary to align them, which the fixed offset makes easy; a delay drawn afresh for every execution would make it harder, and we did not evaluate it.

Agreement with the oracle. Pooled over the five pairs, the samples detected at second order at 10^5 traces, 2,368 in the univariate test and 9,285 sample pairs in the bivariate test, all lie in the support view of the oracle. This is not a narrow prediction: for the univariate test the support view flags 80% of the window of the lock-step core and of the staggered core at $\delta = 1$ and 4, 58% at $\delta = 9$, 25% at $\delta = 18$ and 33% at $\delta = 576$, and for the bivariate test between 80% and 100% of the 5,070 sample pairs. The agreement is most informative where the oracle flags little: in the sequential core and the staggered core at $\delta = 4{,}034$, where it flags no sample for the univariate test and none is detected, and at $\delta = 18$ and 576, where every univariate detection lies in a quarter to a third of the window. The form view does not bound the leakage. Thirteen univariate detections lie outside it, 5 of the 11 at $\delta = 9$ and all 8 at $\delta = 18$, where the form view flags no sample, and three samples at $\delta = 18$ are detected by every pair. There the two domains handle values whose linear forms are independent but whose supports share an input coefficient, a coincidence that only lazy reduction creates. Lazy reduction therefore matters at 10^5 traces, and only the support view bounds where a schedule can leak in this model.

Implemented netlist. All statistics above are on the RTL. We do not report a timing simulation of the routed netlists; timing closure rests on static timing analysis, in which every configuration closes setup with the slack of Table 2 and hold with 50 to 75 ps. The cell-level checks of Section 4.5 run on the routed netlist. A glitch-aware evaluation, recording the timed outputs of the LUTs, or PROLEAD [MM22] on a gate-level netlist under glitch- and transition-extended probes, is left to future work.

6.6 Limitations

(i) Simulation versus silicon. The toggle model is a standard first-order approximation [MOP07] but ignores rise and fall asymmetry, the non-linear response of the supply, clock-tree coupling and measurement noise, and at register transfer level it has no glitches. A pre-silicon pass is therefore necessary but not sufficient for deployed security: glitches make masked hardware leak on silicon even when every register holds a single share [MOP07], which motivated threshold implementations [NRR06] and the robust probing model [FGM+18].

(ii) LFSR reseeding. The LFSR that drives L_1 and L_2 is reset to a fixed seed before every trace. This does not affect L_3, whose security derives from the uniform input share, but it reproduces the dummy and shuffle sequence across traces. That removes the noise of the hiding layers from the statistics, which makes the tests of masking more sensitive, and it means that we do not evaluate the hiding layers themselves; deployed, the LFSR would run continuously or be reseeded per invocation.

Table 7: Leakage of the three schedules under the whole-design toggle model, five independent pairs of runs of 10^5 traces each. Pairs: pairs in which both runs exceed $|t| = 4.5$ at a common sample at some point of the grid of Section 6.1. N: median and range over the five pairs of the first such point ($> 10^5$: not detected). $\max|t|$: median and range over the ten runs of the largest $|t|$ over the window at 10^5 traces. Detected: samples, or sample pairs for the bivariate test, detected at 10^5 traces, pooled over the five pairs; every one lies in the support view. Flagged: samples or sample pairs that the oracle flags in the form and the support view, for the staggered core the union over both orders in which the domains start, out of a window of 5,070 samples for the lock-step core, $5{,}070 + \delta$ for the staggered core and 9,105 for the sequential core, and out of 5,070 sample pairs for the bivariate test, which pairs the samples c and $c + \delta$ ($c + 4{,}035$ for the sequential core).

					Flagged	
Schedule	Pairs	N	$\max\|t\|$	Detected	forms	supports
First order						
Lock-step	0/5	$> 10^5$	3.9 (3.6–4.4)	0	–	–
Staggered, $\delta = 1$	0/5	$> 10^5$	3.7 (3.3–4.2)	0	–	–
Staggered, $\delta = 4$	0/5	$> 10^5$	3.8 (3.3–4.2)	0	–	–
Staggered, $\delta = 9$	0/5	$> 10^5$	3.7 (3.5–4.6)	0	–	–
Staggered, $\delta = 18$	0/5	$> 10^5$	4.0 (3.3–4.5)	0	–	–
Staggered, $\delta = 576$	0/5	$> 10^5$	3.9 (3.3–4.3)	0	–	–
Staggered, $\delta = 4{,}034$	0/5	$> 10^5$	3.8 (3.4–4.3)	0	–	–
Sequential	0/5	$> 10^5$	3.9 (3.6–4.3)	0	–	–
Second order, univariate						
Lock-step	5/5	2,000 (2,000–3,000)	31.8 (29.8–32.7)	1,522	4,034	4,034
Staggered, $\delta = 1$	5/5	5,000 (3,000–5,000)	25.8 (25.2–27.6)	667	4,033	4,033
Staggered, $\delta = 4$	5/5	20,000 (10,000–20,000)	14.8 (13.8–15.9)	112	4,030	4,030
Staggered, $\delta = 9$	5/5	60,000 (60,000–60,000)	6.5 (5.6–7.0)	11	2,684	2,939
Staggered, $\delta = 18$	5/5	40,000 (40,000–60,000)	7.7 (7.1–8.7)	8	0	1,262
Staggered, $\delta = 576$	5/5	7,500 (5,000–20,000)	13.7 (12.1–14.5)	48	1,814	1,838
Staggered, $\delta = 4{,}034$	0/5	$> 10^5$	3.9 (3.6–4.5)	0	0	0
Sequential	0/5	$> 10^5$	4.1 (3.7–5.1)	0	0	0
Second order, bivariate at the offset of the schedule						
Staggered, $\delta = 1$	5/5	5,000 (3,000–5,000)	24.5 (23.5–26.3)	1,031	4,034	4,034
Staggered, $\delta = 4$	5/5	3,000 (1,000–5,000)	34.6 (32.2–35.4)	914	4,042	4,044
Staggered, $\delta = 9$	5/5	3,000 (2,000–5,000)	29.1 (27.3–30.6)	1,008	4,062	4,063
Staggered, $\delta = 18$	5/5	5,000 (3,000–5,000)	26.2 (24.2–27.2)	1,004	4,083	4,090
Staggered, $\delta = 576$	5/5	3,000 (2,000–5,000)	34.7 (32.8–36.1)	1,176	4,067	4,557
Staggered, $\delta = 4{,}034$	5/5	1,000 (500–2,000)	46.6 (44.8–47.9)	2,077	4,034	5,062
Sequential	5/5	1,000 (1,000–2,000)	46.9 (45.6–47.4)	2,075	4,035	4,293

(iii) Order of the claims. The design targets $d = 1$; the second-order tests are measurements that distinguish the schedules, not security claims.

(iv) Glitches. The structural argument and the cell-level checks cover glitch-extended probes, but the statistical evidence is at register transfer level, and we have no timed simulation of the netlist (Section 6.5).

(v) Physical coupling. The toggle model adds the activity of the cells linearly and misses coupling through the supply network or between routes and what happens inside the DSP48E1 primitives; physical separation of the domains, for instance one placement block per domain, could reduce such coupling and was not applied.

(vi) Single-trace attacks. Our TVLA averages over traces, so it does not assess single-trace or few-trace attacks on the NTT [PPM17, PP19, HHP+21], which first-order masking does not prevent and against which the 16-bit shuffle is BASTION's only layer. Assessing them requires a profiled attack with belief propagation, which we leave to future work.

7 Conclusion and Future Work

We presented BASTION, a first-order masked NTT core for ML-KEM on FPGA in three share schedules built on one datapath and synthesised under one flow. None of them needs fresh randomness, since the butterfly is linear in its public twiddle factor; what they change is the cost, and when the two shares of an intermediate meet.

Against an iso-architectural unmasked control and at matched synthesis regime, the lock-step core costs 1.76× the LUTs, 1.86× the slices and twice the DSPs at an unchanged cycle count. Under the preservation attributes, which the cell-level checks show to be needed, the sequential core costs 9 LUTs more than the control at twice its cycles, and the staggered core 1.38× the LUTs of the lock-step core and δ cycles.

We stated when a sample combines both shares of a secret, namely when the forms that the two domains handle in it are linearly dependent, measured the forms on the RTL and used the criterion to predict, cycle by cycle, where each schedule can leak at second order. Calibrated with injected flaws, the harness finds no first-order leakage in any schedule in five pairs of runs of 10^5 traces. At second order the schedules form a scale. The lock-step core leaks in single samples. The staggered core leaks in single samples too, more weakly, at every delay we tested up to one stage, and at the pair of samples its delay relates. The sequential core and the staggered core at a delay of one transform leak only at that pair. Separating the domains in time thus moves the leakage from single samples to pairs of samples a public distance apart rather than removing it. Every detected sample lies in the support view of the oracle, and at a delay of two iterations every univariate detection lies outside the form view, so the coincidences that lazy reduction creates do leak. The evidence is pre-silicon; its six limitations are stated in Section 6.6, and single-trace attacks, fault injection and decapsulation-oracle attacks are not treated here.

Future work. The first step is to validate the three schedules on silicon, which would expose what no simulation reaches, namely coupling through the supply and the routing, the internals of the DSP48E1 primitives and transients inside cells, and to mount correlation, template, deep-learning and soft-analytical attacks [BCO04, CRR02, MPP16, CWZ+25, PPM17, PP19], since a t-test does not replace an adversarial evaluation. The inverse transform is linear as well and follows the same discipline, while compression and decoding need masked non-linear gadgets, for which full-system implementations [KNAH22, FVB+22, LMRBG24, CRNTC26] provide templates. Three changes follow from the oracle. We made the first two, a flush that drains the pipeline between the passes of the sequential core and a read-back that waits for the pipeline to drain; the third, a delay drawn afresh for every execution of the staggered core, which would deny the bivariate adversary a fixed offset, remains to be evaluated. Finally, the harness and the oracle apply unchanged to a streaming datapath such as that of HOPE-MLKEM.

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting docu-

mentation (version 3.02). NIST PQC Round 3 submission, 2021. URL: https://pq-crystals.org/kyber/.

[Alf96] Peter Alfke. Efficient shift registers, LFSR counters, and long pseudo-random sequence generators. Xilinx Application Note XAPP052, v1.1, 1996.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, pages 16–29, 2004.

[BNAMK21] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *IEEE 28th Symposium on Computer Arithmetic (ARITH)*, pages 94–101, 2021.

[BV25] Jonas Bertels and Ingrid Verbauwhede. A high throughput Kyber NTT. Cryptology ePrint Archive, Paper 2025/1763, 2025. URL: https://eprint.iacr.org/2025/1763.

[CBVB25] Rafael Carrera Rodriguez, Florent Bruguier, Emanuele Valea, and Pascal Benoit. Cracking the mask: SASCA against local-masked NTT for CRYSTALS-Kyber. *IACR Communications in Cryptology*, 2(2), 2025. doi:10.62056/aesgbnja5.

[CMBM26] Siddhartha Chowdhury, Nimish Mishra, Sarani Bhattacharya, and Debdeep Mukhopadhyay. Unified FPGA design of Kyber and Dilithium with provable fault tolerance. Cryptology ePrint Archive, Paper 2026/1008, 2026. URL: https://eprint.iacr.org/2026/1008.

[CMJ23] Zhaohui Chen, Yuan Ma, and Jiwu Jing. Low-cost shuffling countermeasures against side-channel attacks for NTT-based post-quantum cryptography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(1):322–326, 2023.

[CRNTC26] Eros Camacho-Ruiz, Pablo Navarro-Torrero, and Alejandro Cabrera Aldaya. A framework for designing high-order side-channel protected hardware implementations of ML-KEM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):272–295, 2026. doi:10.46586/tches.v2026.i2.272-295.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, pages 13–28, 2002.

[CVBB24] Rafael Carrera Rodriguez, Emanuele Valea, Florent Bruguier, and Pascal Benoit. Hardware implementation and security analysis of local-masked NTT for CRYSTALS-Kyber. Cryptology ePrint Archive, Paper 2024/1194, 2024. URL: https://eprint.iacr.org/2024/1194.

[CWZ+25] Munkhbaatar Chinbat, Liji Wu, Xiangmin Zhang, Yifan Yang, and Man Wei. Comparative deep learning-based side-channel analysis of an FPGA-based CRYSTALS-Kyber NTT accelerator. *Cryptography*, 9(4):64, 2025. doi:10.3390/cryptography9040064.

[Del22] Jeroen Delvaux. Roulette: A diverse family of feasible fault attacks on masked Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2022(4):637–660, 2022. doi:10.46586/tches.v2022.i4.637-660.

[FGM+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults and the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, 2018. doi:10.13154/tches.v2018.i3.89-120.

[FVB+22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):414–460, 2022.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop (NIAT)*, 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Theory of Implementation Security Workshop (TIS@CCS)*, 2016.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(4):88–113, 2021. doi:10.46586/tches.v2021.i4.88-113.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *Applied Cryptography and Network Security (ACNS)*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2006.

[IKR+26] Malik Imran, Ayesha Khalid, Ciara Rafferty, Safiullah Khan, Muhammad Rashid, and Maire O'Neill. PIP-NTT: Towards a scalable memory-parallelized accelerator for iterative NTT in PQC. arXiv:2607.18533, 2026. URL: https://arxiv.org/abs/2607.18533.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology – CRYPTO 2003*, pages 463–481, 2003.

[JGCS24] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, and Somitra Kumar Sanadhya. A configurable CRYSTALS-Kyber hardware implementation with side-channel protection. *ACM Transactions on Embedded Computing Systems*, 23(2):1–25, 2024. doi:10.1145/3587037.

[KBMSH24] Dur-e-Shahwar Kundi, Jose Maria Bermudo Mera, Pierre-Yves Strub, and Michael Hutter. High-performance NTT hardware accelerator to support ML-KEM and ML-DSA. In *Workshop on Attacks and Solutions in Hardware Security (ASHES)*, pages 100–105, 2024. doi:10.1145/3689939.3695785.

[KNAH22] Tendayi Kamucheka, Alexander Nelson, David Andrews, and Miaoqing Huang. A masked pure-hardware implementation of Kyber cryptographic algorithm. In *IEEE International Conference on Field-Programmable Technology (ICFPT)*, 2022.

[LMRBG24] Georg Land, Adrian Marotzke, Jan Richter-Brockmann, and Tim Güneysu. Gadget-based masking of streamlined NTRU prime decapsulation in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2024(1):1–26, 2024. doi:10.46586/tches.v2024.i1.1-26.

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

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[MÖS20] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(2):353–362, 2020.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering (SPACE)*, pages 3–26, 2016.

[Nat24] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication 203, 2024. doi:10.6028/NIST.FIPS.203.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security (ICICS)*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked Ring-LWE implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2018(1):142–174, 2018. doi:10.13154/tches.v2018.i1.142-174.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology – LATINCRYPT 2019*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019. doi:10.1007/978-3-030-30530-7_7.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(2):37–60, 2021. doi:10.46586/tches.v2021.i2.37-60.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017. doi:10.1007/978-3-319-66787-4_25.

[PQC] PQClean contributors. PQClean: Clean, portable, tested implementations of post-quantum cryptography. https://github.com/PQClean/PQClean. Accessed 2026.

[RBSC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. Drop by drop you break the rock: Exploiting generic vulnerabilities in lattice-based PKE/KEMs using EM-based physical attacks. Cryptology ePrint Archive, Paper 2020/549, 2020. URL: https://eprint.iacr.org/2020/549.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT: A performance evaluation study over Kyber and Dilithium on the ARM Cortex-M4. In *Security, Privacy, and Applied Cryptography Engineering (SPACE 2020)*, volume 12586 of *Lecture Notes in Computer Science*, pages 123–146. Springer, 2020. doi:10.1007/978-3-030-66626-2_7.

[RSVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-LWE implementation. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 683–702. Springer, 2015. doi:10.1007/978-3-662-48324-4_34.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–134, 1994.

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology: Extended version. *Journal of Cryptographic Engineering*, 6(2):85–99, 2016.

[SRM+26] Omar S. Sonbul, Muhammad Rashid, Muhammad I. Masud, Mohammed Aman, and Amar Y. Jaffar. Deeply pipelined NTT accelerator with ping-pong memory and LUT-only barrett reduction for post-quantum cryptography. *Electronics*, 15(3):513, 2026. doi:10.3390/electronics15030513.

[TAM25] Bardia Taghavi, Reza Azarderakhsh, and Mehran Mozaffari Kermani. ParallelNTT: Maximizing performance of forward and inverse NTT on FPGA for ML-DSA and ML-KEM. In *Great Lakes Symposium on VLSI (GLSVLSI)*, New Orleans, LA, USA, 2025.

[TKKA24] Tolun Tosun, Selim Kırbıyık, Emre Koçer, and Ersin Alaybeyoğlu. Optimized FPGA architecture for modular reduction in NTT. Cryptology ePrint Archive, Paper 2024/1890, 2024. URL: https://eprint.iacr.org/2024/1890.

[WSKB22] Zhehong Wang, Dennis Sylvester, Hun-Seok Kim, and David Blaauw. Hardware acceleration for third-generation FHE and PSI based on it. arXiv:2204.11334, 2022. URL: https://arxiv.org/abs/2204.11334.

[XL21] Yufei Xing and Shuguo Li. A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(2):328–356, 2021.

A Use of AI Tools

This work was carried out with Claude, a generative AI assistant by Anthropic, used through claude.ai and through its desktop application between March and November 2026, with the models that claude.ai offered at each point; this revision was made with Claude Opus 5.5 and Claude Sonnet 5.5.

What the assistant did. It proposed the separation of the two shares into disjoint cells of the FPGA, on which BASTION rests, Observation 1 with its proof, and the oracle built on it. It drafted and revised the text of every section of this paper and generated Figure 1 and the tables; the measured values in the tables come from the simulation and synthesis runs described in this appendix, and those of other designs from the cited papers. It wrote the RTL of every core: Baselines A, B and C, the lock-step core and its variant without preservation attributes, and the staggered and sequential cores, together with their testbenches and simulation monitors, and the synthesis, period-sweep and cell-level separation scripts. It wrote the leakage harness of Section 6: the testbench, the toggle model, its Verilator build and the cross-check between simulators, as well as the oracle and the statistical analysis. It implemented the changes of this revision, among them the three-cycle flush of the sequential core, the read rule, the per-execution seeding and the analysis over five pairs of runs. In the previous revision it launched the first campaigns of the harness, the synthesis sweeps noted below and existing simulations when the paper was restructured; in this revision it ran the functional simulations and the oracle.

What the author did. The author proposed comparing the three share schedules on one core, and ran the synthesis and implementation in Vivado of the earlier versions of the cores and the period sweep of the revised sequential core in Table 2; the sweeps of the other six rows of that table were launched by the assistant on the author's machine in the previous revision. The author carried out the earlier TVLA evaluation that remains in the artifact under `tvla/` and is not used in this paper, and found and debugged errors. The author took the decisions of this revision: the statistical protocol (five independent pairs of runs of 10^5 traces and the detection grid), the fix of the pipeline drain, the convention $f_{\max} = 1/T$ and the route of publication, and launched the leakage campaigns of Section 6.

Verification. Another session of the same assistant audited the previous revision of this paper and its artifact, and reproduced its campaigns; the problems it found led to this revision.

Prompts. No complete record of the prompts exists. The work is spread over more than ten conversations, some of which have been deleted. It proceeded as a dialogue: the author stated goals, constraints and decisions, the assistant proposed and wrote designs, code, analyses and text, and the author ran the tools named above, reported the results and errors back, and chose among the alternatives the assistant laid out. The author takes full responsibility for the content of the paper.