

CMALU: Compact Fault-Tolerant Modular Arithmetic Logic Unit for Post-Quantum Cryptography

YoungBeom Kim¹, Malik Imran², Zain Ul Abideen³, Ciara Rafferty², Ayesha Khalid²,
Máire O'Neill², and Seog Chung Seo^{1,*}

¹Kookmin University, Seoul, Republic of Korea ²Queen's University, Belfast, UK ³University of Idaho, USA
{darania,scseo}@kookmin.ac.kr,{m.imran,c.m.rafferty,a.khalid}@qub.ac.uk,m.oneill@ecit.qub.ac.uk,zabideen@uidaho.edu

Abstract

The rise of quantum computing threatens widely deployed public-key cryptosystems, driving the adoption of post-quantum cryptography (PQC) algorithms that rely heavily on modular arithmetic. Existing hardware accelerators of the PQC algorithms for resource-constrained Internet-of-Things (IoT) devices remain limited and lack integrated fault detection mechanisms. In this work, we present CMALU, a Compact, fault-tolerant Modular Arithmetic Logic Unit supporting six operations on a single reconfigurable datapath, with a 2-bit input selecting **Mode-0** (un-protected baseline), **Mode-1** (on-the-fly parity and invariant checking with a formal single-bit detection guarantee), and **Mode-2** (extending **Mode-1** along with hardware-reuse recomputation for deterministic silent data corruption (SDC) elimination without datapath duplication). Under system-level fault injection into CMALU internal registers on an NTT accelerator and an Ibex RISC-V core running ML-KEM-512, **Mode-1** achieves 100% single-bit and stuck-at detection at zero latency overhead, and **Mode-2** achieves 0% SDC. The synthesis results after the post-place-and-route stage on a field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) implementations with the NTT accelerator targeting 65nm CMOS and the Ibex RISC-V integration targeting Nangate45 45nm confirm CMALU's suitability for resource-constrained IoT deployment.

CCS Concepts

• **Hardware** → **Hardware reliability**; • **Security and privacy** → *Embedded systems security*;

Keywords

Fault-Tolerant, ALU, PQC, IoT Security, FPGA, ASIC, Montgomery modular multiplication, Fault Detection.

ACM Reference Format:

YoungBeom Kim¹, Malik Imran², Zain Ul Abideen³, Ciara Rafferty², Ayesha Khalid², Máire O'Neill², and Seog Chung Seo^{1,*}. 2026. CMALU: Compact Fault-Tolerant Modular Arithmetic Logic Unit for Post-Quantum Cryptography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26)*, November 08–12, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3831252.3834115>

*Corresponding Author.



This work is licensed under a Creative Commons Attribution 4.0 International License.

ICCAD '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2873-0/2026/11

<https://doi.org/10.1145/3831252.3834115>

1 Introduction

Quantum computing poses a fundamental threat to classical public-key cryptography. Shor's algorithm [36] can efficiently factor large integers and resolve discrete logarithm problems, rendering widely deployed schemes such as RSA and Elliptic-Curve Cryptography (ECC) vulnerable. In response, the National Institute of Standards and Technology (NIST) has standardized ML-KEM [30] and ML-DSA [31] as Post-Quantum Cryptography (PQC) schemes, with Falcon [33] forming the basis for the forthcoming FN-DSA standard. These lattice-based constructions rely heavily on modular arithmetic operations, including addition, subtraction, multiplication, and reduction. Efficiently realizing such operations on resource-constrained Internet of Things (IoT) devices remains a considerable open challenge because of limited processing power and available memory resources.

1.1 Hardware Accelerators + Their Limitations for IoT Deployments

Implementing the computationally intensive PQC schemes in software on IoT devices does not meet practical performance requirements. Therefore, by targeting ultra-lightweight devices, recent works have introduced various algorithmic optimizations to mitigate such computational complexity [21, 27]. For example, the RISC-V implementations of ML-KEM and ML-DSA in [38] achieve significant speedups by offloading polynomial multiplication to dedicated hardware. Similarly, a wide range of impressive hardware accelerators has been explored in the literature and benchmarked on platforms such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). These include a translated architecture through High-Level Synthesis (HLS) from C/C++ to HDL [25], software-hardware co-design [28, 32, 38], dedicated implementations [3, 9, 37], and RISC-V ISA extensions [4, 12, 14]. Most of these designs rely on Digital Signal Processor (DSP)-based multipliers or modulus-specific reduction hardware. Moreover, these accelerators reveal that polynomial multiplication is the computational bottleneck in ML-KEM and ML-DSA and is commonly accelerated using the efficient Number Theoretic Transform (NTT) approach. The existing efforts have focused on reducing clock cycles and maximizing circuit frequency through deep pipelining to improve NTT performance [3, 28, 32, 37].

The most impressive fault-tolerant PQC accelerators are implemented in [1, 8, 26]. These architectures reveal that the high-speed NTT datapaths and bit-serial Montgomery loops are vulnerable to low-cost fault injection attacks, which can corrupt a single bit and compromise sensitive information. In addition, in these designs, several promising fault-tolerant measures are considered, such as

parity checking for NTT and interpolation-based countermeasures in protocols like ML-KEM and ML-DSA. Although these architectures address potential vulnerabilities, they incur significant overhead in resource-constrained environments.

Even if we have several hardware accelerators for the NTT as well as for the PQC standards (ML-KEM and ML-DSA) [3, 4, 9, 12, 14, 22, 25, 28, 32, 37, 38], these architectures have certain limitations for IoT deployments: (i) most of these architectures are modulus- and scheme-specific, (ii) the reported area/power costs in these reference designs are unsuitable for IoT devices, (iii) all these architectures do not incorporate fault detection mechanisms, which makes these designs vulnerable to fault attacks. Moreover, the existing fault-tolerant accelerators offer high-performance in [1, 8, 26] with substantial area and energy overheads, which prevent deployment in resource-constrained IoT environments.

1.2 Motivation + Contributions

The gaps in Section 1.1 motivate the need for a *compact, flexible, and fault-tolerant* arithmetic engine that unifies the computational needs of PQC, ECC, and classical modular cryptography within a single lightweight design. We introduce the CMALU architecture¹, supporting six operations: three integer (ADD, SUB, MUL) and three modular (MADD, MSUB, MMUL). CMALU is scoped as an arithmetic-core countermeasure: faults in memory, twiddle storage, control flow, or protocol comparisons are delegated to orthogonal mechanisms (Section 3.3). Our contributions are as follows:

- **Compact shared datapath.** We unify six integer and modular operations on a single Carry-Save Adder (CSA) and Carry-Lookahead Adder (CLA) datapath with a Bit-serial Montgomery loop to eliminate the modulus-specific multipliers and DSP slices used by prior PQC accelerators, resulting in an ultra-lightweight CMALU engine suitable for IoT devices.
- **Structure-aware fault detection.** We integrate carry-aware parity prediction and a Montgomery-evenness invariant checker that reuses the same CSA/CLA hardware as the arithmetic datapath, providing fault protection without duplicating arithmetic logic. This yields a mathematical proof that the checker catches any single-bit error in the values it monitors, and 100% detection of all single-bit and stuck-at faults in experiments where faults were deliberately injected into CMALU internal registers.
- **Configurable security-performance trade-off.** We provide the following three protection modes, selectable with a 2-bit input, to cover a wide spectrum:
 - **Mode-0** acts as a baseline (or un-protected CMALU datapath),
 - **Mode-1** for on-the-fly detection with zero latency overhead,
 - **Mode-2** for deterministic elimination of silent data corruption (SDC) by extending **Mode-1** with a hardware-reuse re-computation strategy.
- **Case study for benchmarking.** To target resource-constrained IoT, we validate CMALU on Artix-7 via a loosely-coupled NTT accelerator and a tightly-coupled Ibex RISC-V core. Benchmarking across ML-KEM, ML-DSA, and Falcon shows that **Mode-1** on Ibex achieves a 1.42× ML-KEM-512 speedup (2.56× NTT) using only 564 LUTs and zero DSPs. **Mode-2** yields 1.25× speedup

(1.85× NTT) with 651 LUTs. Both configurations are DSP-free and yield the lowest eSlice+ normalization among prior Ibex-based PQC accelerators, proving efficiency for area-limited embedded platforms.

The rest of this paper is organized as follows. Section 2 provides the necessary background. Section 3 describes the threat model. Section 4 presents the CMALU architectures along with the fault detection methodology. Section 5 presents case studies covering NTT acceleration and Ibex RISC-V integration. Section 6 reports performance and fault injection results.

2 Background

2.1 Number Theoretic Transform (NTT)

The NIST-standardized ML-KEM [30], ML-DSA [31], and FN-DSA [33] operate over polynomial rings $\mathbb{Z}_q[x]/(x^n + 1)$, where NTT-based polynomial multiplication dominates the computational cost. The NTT is computed via Cooley–Tukey (CT) [13] and Gentleman–Sande (GS) [16] butterflies:

$$\text{CT} = (a + b \cdot \zeta) \bmod q, \quad (a - b \cdot \zeta) \bmod q \quad (1)$$

$$\text{GS} = (a + b) \bmod q, \quad (a - b) \cdot \zeta \bmod q \quad (2)$$

where $\zeta \in \mathbb{Z}_q$ is a twiddle factor, and CT and GS compute the forward-NTT and inverse-NTT (iNTT), respectively. Both butterflies require modular multiplication, addition, and subtraction (MMUL/MADD/MSUB), motivating a unified modular arithmetic core.

2.2 Bit-Serial Montgomery Multiplication

2.2.1 State-of-the-Art methods. In the state-of-the-art hardware implementations, modular multiplications are frequently realized using Barrett [6], KRED [9], and Montgomery [38] approaches. Moreover, approaches like modulus-specific (q -dedicated) reduction hardware are also considered, as in [3, 28, 37]. However, these designs either rely on DSP-based multipliers unsuitable for resource-constrained IoT devices or employ q -specific reduction logic that lacks agility across different PQC parameter sets.

2.2.2 Utilized bit-serial multiplication (from [29]). In this work, we have used a bit-serial Montgomery-based modular multiplication (BSM) from [29] and formulated it according to our CMALU datapath in Algorithm 1. The related classical Montgomery reduction framework is described in [29]. It offers DSP-free multiplications and processes one multiplier bit per cycle over $W = \lceil \log_2 q \rceil$ iterations on our shared CSA/CLA datapath. Each iteration computes $U = C + E[0] \cdot D + F \cdot P$, where the correction-bit F enforces $U \bmod 2 = 0$ for exact right-shift; the five registers C, D, E, P, F are referenced throughout Sections 4.

3 Threat Model and Scope

In this paper, we assume a physical adversary who targets the arithmetic core during active computation using laser fault injection, electromagnetic fault injection (EM-FI), or voltage glitching with cycle-level precision [1, 2, 7, 23, 26]. This section formalizes the fault classes, attack scenarios, and scope that underlie the fault-detection claims of Section 4.4.

¹The CMALU Verilog source code is publicly available at <https://github.com/whYBeKim/CMALU-for-PQC>.

Algorithm 1 Bit-Serial Montgomery Multiplication

Input: Multiplicand D , multiplier E , odd modulus P , bit-width W
Output: $R = D \cdot E \cdot 2^{-W} \bmod P$

```

1:  $C \leftarrow 0$                                  $\triangleright$  Accumulator
2: for  $i \leftarrow 0$  to  $W - 1$  do
3:    $F \leftarrow C[0] \oplus (E[0] \cdot D[0])$        $\triangleright$  Correction bit
4:    $U \leftarrow C + (E[0] \cdot D) + (F \cdot P)$        $\triangleright$  CSA + CLA
5:    $C \leftarrow U \gg 1$                          $\triangleright$  Exact division by 2
6:    $E \leftarrow E \gg 1$                          $\triangleright$  Next multiplier bit
7: end for
8: if  $C \geq P$  then
9:    $C \leftarrow C - P$                          $\triangleright$  Final normalization to  $[0, P)$ 
10: end if
11: return  $R \leftarrow C$ 

```

3.1 Fault Model

We define the following five fault classes that span the practical injection landscape [1, 24]. Four are transient, and one is permanent.

- **Single-bit Flip (SF).** Indicates one bit inversion per cycle.
- **Multi-bit Flip (MF).** Denotes 2–5 simultaneous bit inversions at arbitrary positions per cycle.
- **Burst (BU).** k consecutive bit inversions ($2 \leq k \leq 8$) per cycle.
- **Stuck-at (ST).** Indicates a bit that is forced to 0 or 1 for the entire computation.
- **Random Transient (RT).** An arbitrary-weight perturbation affecting multiple bits per cycle.

According to [1], transient faults (SF, MF, BU, RT) manifest as an XOR bit-flip on the target word, $V' = V \oplus \Delta$, where Δ is a nonzero mask. In contrast, ST faults override the targeted bit value for the duration of the injection, producing an effective data-dependent mask that reduces to an XOR, which we have used in the checker analysis of Section 4.4. The injection targets are the internal datapath of the arithmetic core (adders, carry-propagation networks, and working registers used during modular arithmetic). In our experiments, we instantiate this model at the five working registers (C, E, D, P, F) that expose the protected checker window.

3.2 Attack Scenarios

Concerning prior threat studies published in [1, 20, 34], we identify three attack scenarios. Attacks 1–2 exploit transient faults (SF, MF, BU, RT), while Attack 3 targets permanent stuck-at faults (i.e., ST).

- **Attack 1 (differential fault analysis on NTT butterfly).** In this differential fault analysis (DFA) scenario, an adversary injects a fault into an internal word V during a butterfly at time t , yielding a faulty output $O'_t \neq O_t$. If undetected, the differential $O'_t \oplus O_t$ leaks secret polynomial coefficients [23, 34].
- **Attack 2 (fault replay on BSM).** A BSM loop iterates the same datapath for W cycles, exposing W injection windows per operation. This is far more than the single window of a parallel multiplier [24]. An adversary repeats the target operation N times, collecting undetected faulty outputs $O'_1, \dots, O'_N$ to mount a statistical or differential attack [20].
- **Attack 3 (Montgomery reduction skip).** An adversary forces the correction bit F (from Algorithm 1) to a stuck value $s \in \{0, 1\}$, producing $U' = C + E[0] \cdot D + s \cdot P$ instead of the correct even sum U . When $s \neq F$, the intermediate sum U' may be odd, invalidating the right-shift reduction and propagating a secret-correlated faulty value downstream [1, 19, 20].

3.3 Scope

We distinguish the broad fault-attack motivation above from the coverage this work demonstrates: CMALU is an arithmetic-core countermeasure, not an end-to-end defense, and detects faults only once an operand is captured into the modular-arithmetic datapath. The formal protection claims apply to transient faults on the modular arithmetic paths of CMALU. Out of scope are checker and comparator faults, common-mode recomputation errors, memory-I/O, Keccak-hash faults, control-flow hijacking, and Fujisaki–Okamoto check bypasses [18], consistent with prior arithmetic-core threat models [1, 8]. Additionally, the added checker and comparator logic may introduce power, EM, or timing side-channel leakage. Evaluating such leakage is beyond the scope of this work, which focuses solely on arithmetic-core fault detection and does not claim side-channel resistance. Furthermore, the checker comparisons are combinational and do not change the architectural cycle count of the protected operations.

4 CMALU: Architecture and Fault Detection

Section 4.1 introduces the un-protected CMALU baseline, while the fault-protected architecture is in Section 4.2. Sections 4.3–4.5 complete the security argument: the one-cycle fault response (Section 4.3), formal detection guarantees (Section 4.4), and per-attack coverage mapped to the threat model (Section 4.5).

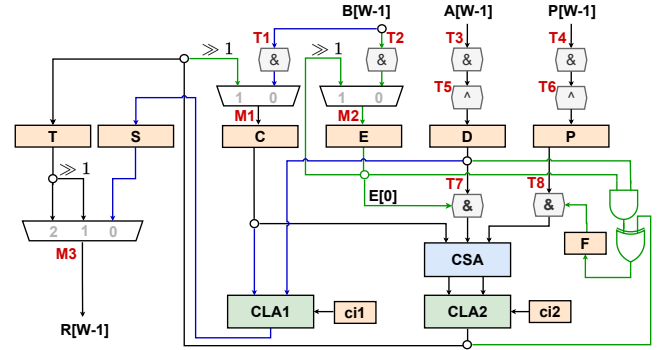


Figure 1: Mode-0: datapath of the un-protected CMALU .

4.1 Baseline: Un-protected Architecture

CMALU is a compact, DSP-free arithmetic unit implementing six operations, three integer (ADD, SUB, MUL) and three modular (MADD, MSUB, MMUL), on a single datapath comprising a 3:2 CSA, two Kogge–Stone carry-lookahead adders (CLA1, CLA2) [35], a BSM engine, shared registers, and a lightweight finite-state machine (FSM) controller. The ADD/SUB and MADD/MSUB operations complete in two cycles, while the MUL/MMUL executes in W cycles by reusing the same CSA/CLA hardware. Figure 1 shows the un-protected CMALU, where the blue paths serve ADD, SUB, MADD, and MSUB, while the green paths serve MUL and MMUL, along with the CSA, CLA1, CLA2, and shared registers. The black paths are common to all operations.

ADD, SUB, and MUL. The datapath comprises T1–T5 operand transformation units, M1–M3 control multiplexers, and a CSA/CLA computation core. For ADD and SUB, operands A and B pass through T1–T5, where they may be conditionally toggled or bit-shifted before entering CLA1 to produce the result in two cycles. MUL uses the

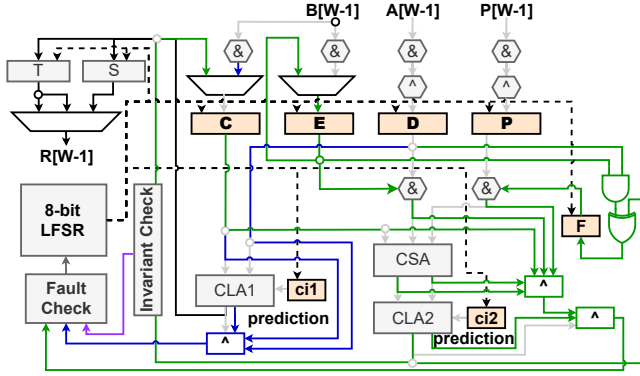


Figure 2: Fault-protected CMALU datapath.

BSM engine: register E shifts one bit at a time, with each bit $E[0]$ conditionally injecting multiplicand D into the CSA. Over W cycles, partial products accumulate via CSA and finalize using CLA2, with M3 selecting the output (bit-width doubled from W to $2W$).

MADD, MSUB, and MMUL. The modular operations extend the same datapath with the modulus register P. MADD and MSUB compute via CLA1 and conditionally correct using P through T4/T6 when overflow/underflow is detected, completing in two cycles through the CSA/CLA2 pipeline. The MMUL extends MUL with Montgomery reduction: each multiplier bit triggers both partial-product accumulation (via D) and correction (via P) inside the CSA, producing the Montgomery-domain product $R = D \cdot E \cdot 2^{-W} \bmod P$ after W iterations, fully reduced to $[0, P)$.

4.2 Fault-protected Architecture

Figure 2 illustrates the fault-protected CMALU datapath. To address the attack scenarios of Section 3.2, fault detection and response are integrated into the modular arithmetic subpaths without duplication, selectable at runtime via a 2-bit iMODE input:

- **Mode-0:** Un-protected design (a baseline from Figure 1).
- **Mode-1:** Online fault detection through parity prediction and Montgomery-evenness invariant checking with zero latency overhead.
- **Mode-2:** Extends **Mode-1** with hardware-reuse recomputation for deterministic SDC detection.

Parity and invariant monitors are active in both **Mode-1** and **Mode-2**, while recomputation is dedicated to Mode-2. The protected CMALU architecture is explained further through the incorporated fault detection mechanism in Sections 4.2.1–4.2.3.

4.2.1 Parity Check. A carry-aware parity predictor monitors the CSA–CLA datapath for bit-level corruption. For MMUL, the predicted parity is compared with the observed parity after CLA2 carry propagation at every iteration (green path in Figure 2). For MADD/MSUB, the same mechanism is applied to the CLA1 (via the blue path). Note that output-level parity alone is vulnerable to self-cancellation, where a fault in a source register shifts both actual and predicted parities equally. To address this, register-level shadow parity monitors are added for four registers (i.e., D, P, E, and C):

- D and P remain constant during MMUL, so their load-time parities are compared every iteration.

- E shifts right each cycle, so its expected parity is tracked via $p_{\text{new}} = p_{\text{old}} \oplus E[0]$, where $\oplus$ denotes the Exclusive-OR operation.
- C is verified against the predicted parity from the current CLA2 output.

For MADD/MSUB, a clean reference parity is captured from input ports ($\oplus A, \oplus B, \oplus P$) at the operation start and compared during the computation cycle, with toggle corrections for operand complementing (e.g., $D = \bar{B}$). This eliminates self-cancellation and achieves 100% SF fault detection on MADD/MSUB in system-level campaigns. The same parity mechanism extends to integer operations (ADD, SUB, MUL) on the shared datapath, though these paths are not exercised by the ML-KEM, ML-DSA, and Falcon evaluated in this paper.

4.2.2 Montgomery-Evenness Invariant. During each MMUL iteration, the intermediate sum $U = C + E[0] \cdot D + F \cdot P$ must be even, as the correction bit $F = C[0] \oplus (E[0] \cdot D[0])$ enforces $U \bmod 2 = 0$ for any odd modulus P (purple path, Figure 2). CMALU checks $U[0]$ every iteration, detecting any fault that corrupts F or breaks the even-sum invariant independently of parity.

4.2.3 Hardware-Reuse Recomputation. The **Mode-2** saves the input operands and operation type at load time, re-executes on the same datapath, and then compares results R_1 and R_2 ; a mismatch triggers the fault response (described in Section 4.3). The latency overhead is theoretically $2\times$, with a minimal area increment from saved-input registers, the result comparator, and one additional FSM state.

4.3 Fault Response

The fault response depends on the detection stage. The **Online checkers** (parity predictor, invariant checker, and shadow parity monitors) trigger all four responses within one cycle:

- **Output suppression.** The result is replaced by a linear-feedback shift register (LFSR)-derived value to break the arithmetic relation between the injected fault and the released output.
- **Session abort.** The FSM returns to IDLE and issues a one-cycle FAULT_ABORT pulse.
- **Sticky alert.** A latched FAULT_ALERT flag persists until a hardware reset.
- **State diffusion.** Sensitive registers, C, E, D, F, ci1, ci2 and the final result T and S, are overwritten with the LFSR output to prevent the replay of corrupted state.

Late-stage detectors (recomputation comparator) activate after the computation completes and contribute to output suppression and alerting; in-flight abort and state diffusion are handled by the online checkers. The 8-bit LFSR is deterministic and is triggered only upon detection. In CMALU, it is used as a fail-closed output-substitution mechanism to break the direct arithmetic relation required by DFA, not as a masking primitive or a source of information-theoretic hiding. Its width and polynomial are implementation choices for minimal area and are not used in any formal coverage claim, discussed in Section 4.4.

4.4 Formal Coverage Claims

We analyze coverage in the checker-observed word model. Following the XOR bit-flip fault model of [1], a fault manifests as a nonzero mask $\Delta \in \{0, 1\}^{W'}$ applied to the checked word, producing $V' = V \oplus \Delta$, where $W' = W+3$ is the internal accumulator width. During the Montgomery loop, the accumulator satisfies $0 \leq C < 2P$, while $E[0] \cdot D < P$ and $F \cdot P \leq P$. Hence $U = C + E[0] \cdot D + F \cdot P < 4P < 2^{W+2}$, so $W+2$ value bits suffice and $W' = W+3$ leaves one guard bit. Let V denote the checked word on an active protected path: for MMUL, $V = U$ (the intermediate sum); for MADD/MSUB, V is the corresponding CLA output word. For MMUL, the parity checker observes the post-shifted word $V \gg 1$, while the invariant checker observes $V[0] = U[0]$. Let $\pi(\cdot)$ denote XOR-reduction parity, and let $D_{M1} \in \{0, 1\}$ denote the **Mode-1** detection indicator.

LEMMA 1 (MONTGOMERY-EVENNESS). *For odd modulus P , if a fault flips the correction bit ($F' = F \oplus 1$) while $C[0]$, $E[0]$, and $D[0]$ remain intact, then $U' \bmod 2 = 1$, and the invariant checker detects the fault.*

PROOF. Since $P[0] = 1$, the fault-free relation gives $U \bmod 2 = C[0] \oplus (E[0] \cdot D[0]) \oplus F = 0$. Replacing F with $F' = F \oplus 1$ flips the parity; hence $U' \bmod 2 = 1$. $\square$

THEOREM 1 (SINGLE-BIT DETECTABILITY ON THE CHECKER-OBSERVED WORD). *In **Mode-1**, let $V' = V \oplus 2^j$ ($0 \leq j < W'$) be a single-bit corruption of V . If the parity predictor and invariant checker are fault-free, then $D_{M1} = 1$.*

PROOF. For MMUL: if $j=0$, $V[0]=U[0]=0$ by Lemma 1, so $V'[0]=1$ triggers the invariant checker. If $j>0$, the shifted-word parity flips ($\pi(V' \gg 1) = \pi(V \gg 1) \oplus 1$) while the predictor remains unchanged, triggering the parity checker. For MADD/MSUB: a single-bit flip inverts the checked-word parity of V , causing a mismatch with the predicted value. $\square$

For multi-bit faults ($\text{wt}(\Delta) \geq 2$), the **Mode-1** miss condition on the MMUL checked word is defined in Equation 3. Under a uniform random nonzero mask, the miss probability is defined in Equation 4. Similarly, under a uniformly random support of fixed Hamming weight k , detection is 1 for odd k and k/W' for even k . Thus, for MADD/MSUB (parity only, no invariant), the miss probability under a uniform nonzero mask is in Equation 5.

$$D_{M1} = 0 \iff \Delta[0] = 0 \wedge \text{wt}(\Delta[W'-1:1]) \equiv 0 \pmod{2} \quad (3)$$

$$\Pr[\text{miss}_{\text{MMUL}}] = \frac{2^{W'-2} - 1}{2^{W'} - 1} < \frac{1}{4} \quad (4)$$

$$\Pr[\text{miss}_{\text{MADD}}] = \frac{2^{W'-1} - 1}{2^{W'} - 1} < \frac{1}{2} \quad (5)$$

Equation 3–Equation 5 characterize the checker-observed word model and should not be interpreted as exact implementation-level coverage under arbitrary internal-net injections.

THEOREM 2 (RECOMPUTATION COVERAGE FOR NON-COMMON-MODE ONE-PASS TRANSIENTS). *In **Mode-2**, let $R_1 = \Phi(X) \oplus \epsilon_1$ and $R_2 = \Phi(X) \oplus \epsilon_2$ be the two-pass results for deterministic operation Φ , where ϵ_1, ϵ_2 denotes the architectural result deviations. If the save/compare logic is fault-free, then $\Pr[D_{\text{recomp}}=1] = 1 - \Pr[\epsilon_1=\epsilon_2]$. In particular, any effective transient fault that causes a nonzero result deviation in exactly one pass yields $\Pr[D_{\text{recomp}}=1] = 1$.*

PROOF. $R_1 \neq R_2 \iff \Phi(X) \oplus \epsilon_1 \neq \Phi(X) \oplus \epsilon_2 \iff \epsilon_1 \neq \epsilon_2$. A one-pass effective transient leaves the other pass clean ($\epsilon_2=0$), so $\epsilon_1 \neq \epsilon_2$. $\square$

4.5 Coverage Per Attack Scenario

The above results cover all three modular paths: MMUL benefits from both parity and invariant (miss $< 1/4$), while MADD/MSUB rely on parity alone (miss $< 1/2$); **Mode-2** recomputation applies uniformly. Table 1 maps the attack scenarios from Section 3 to the per-mode detection probability.

Table 1: Mode-1 theoretical per-run detection probability (checker-observed word model). Mode-0 provides no detection. Mode-2 deterministically detects any effective one-pass transient deviation (Theorem 2); for permanent faults it falls back to Mode-1.

Attack	Model scope	Fault	Mode-1
1*	MMUL/MADD/MSUB*	SF	1 (Theorem 1)
1*	MMUL [§]	MF/BU/RT	$> 3/4$ (Equation 4)
1*	MADD/MSUB [§]	MF/BU/RT	$> 1/2$ (Equation 5)
2 [†]	BSM-based paths	SF/MF/BU/RT	= Attack 1 (per run)
3 [‡]	MMUL [°]	ST	$1 - \Pr(F_1=s, \dots, F_m=s)$ (Equation 6)

*DFA on NTT butterfly [†]Fault replay on BSM [‡]Reduction skip

[§]Uniform nonzero mask ^{*}Checked word V [°]Correction-bit stuck

Attack 1 per-run detection follows Theorem 1 for single-bit faults and Equations 4–5 for multi-bit faults; **Mode-2** deterministically detects any effective one-pass transient deviation (Theorem 2). **Attack 2** has the same per-run p_{det} as **Attack 1**; assuming independent runs, the probability of collecting N undetected faulty outputs is $\Pr[N \text{ undetected}] = \prod_{r=1}^N (1 - p_{\text{det},r})$. Here, **Attack 2** is scoped to repeated executions of the same BSM-based protected path; the replay expression does not claim coverage for non-BSM arithmetic that lacks the W -cycle reuse structure defined in Section 3.2. **Attack 3** forces F to a stuck value s ; by Lemma 1, the invariant checker detects whenever $F_i \neq s$, giving cumulative detection:

$$\Pr[\text{detect}] = 1 - \Pr(F_1=s, \dots, F_m=s) \quad (6)$$

Under the toy assumption that the per-iteration correction bits are i.i.d. Bernoulli($1/2$), Equation 6 reduces to $1 - 2^{-m}$. We use this only as intuition for exponential decay of the miss event, not as a workload-faithful guarantee. In **Mode-2**, a permanent stuck fault affects both passes identically ($\epsilon_1=\epsilon_2$), so detection relies on the **Mode-1** invariant checker. The directional consistency of these theoretical values with empirical results is discussed in Section 6.3. **Limitations and Assumptions.** Theorem 1 applies to single-bit corruption of the checker-observed word V , not to arbitrary internal-net or timing-dependent injections; for the broader implementation-level fault space, we report empirical coverage in Section 6.3. An even-weight mask preserving $\Delta[0]=0$ can evade both checkers simultaneously (Equation 3). Theorem 2 does not cover permanent faults with identical corruption across both passes ($\epsilon_1=\epsilon_2$); all theorems assume fault-free checker logic. A twiddle constant corrupted before capture into register D enters as a valid-looking operand that arithmetic consistency cannot flag; exhaustive single-bit twiddle-ROM injection confirms 0% detection in both modes. This storage-layer gap is closed by a one-parity-bit-per-entry ROM extension (+4 LUT), restoring 100% single-bit ROM-content detection through the same fail-closed alert path.

5 CMALU Integration: Case Studies

For performance and fault-injection evaluation, CMALU is integrated into two resource-constrained platforms: a loosely-coupled NTT accelerator (Section 5.1) and a tightly-coupled Ibex RISC-V integration (Section 5.2). The NTT accelerator is synthesized independently for each PQC scheme (ML-KEM, ML-DSA, Falcon), with results reported for all three.

5.1 Case Study 1: CMALU as NTT Accelerator

Figure 3 shows the fault-tolerant CMALU integrated into a radix-2 butterfly supporting both CT and GS configurations for the NTT and iNTT. The design comprises an NTT controller, memory units, a radix-2 butterfly unit, and CMALU, along with a configuration register to initiate computations.

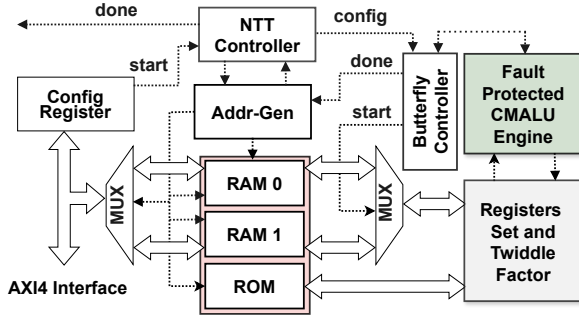


Figure 3: CMALU-based NTT/iNTT accelerator.

5.1.1 NTT Controller + Address Generator (Addr-Gen). The NTT controller and Addr-Gen implement the full NTT/iNTT dataflow: the controller manages stage transitions, butterfly scheduling, and memory-access sequencing, while Addr-Gen produces read/write addresses for storing intermediate and final results.

5.1.2 Memory Units (RAM 0, RAM 1, and ROM). The memory units are the storage elements that keep intermediate and final results for the NTT/iNTT computations. For FPGA implementation, two dual-port Block RAM (BRAM) IPs are used, each of size $32 \times N$, where 32 is the data width and N is the ring size of the target scheme. A read-only memory (ROM) unit of size $W \times \frac{N}{2}$ is utilized, where W denotes the coefficient size of the targeted PQC algorithm, and N is the ring size. For ASIC synthesis, the static random-access memory (SRAM)-based memory units of the same size are generated using commercial memory compilers.

5.1.3 Butterfly Unit (or Controller). The butterfly unit controller sequentially executes Equation 1 and Equation 2 via the MADD, MSUB, and MMUL operators of the CMALU. The NTT/iNTT operates entirely in the Montgomery domain, as in the reference ML-KEM, ML-DSA, and Falcon implementations.

5.2 Case Study 2: Integration with Ibex Core

To evaluate CMALU in a tightly-coupled processor context, we integrate it into the Ibex RISC-V core [15] (Figure 4), configured with RV32MSlow (iterative multiplier) and no DSP inference, matching the low-resource baseline of the standalone CMALU. Four custom instructions are added under the CUSTOM-0 opcode: MMUL,

MADD, MSUB, and CFG (scheme selection). The CMALU is instantiated inside the execution block (ibex_ex_block) alongside the existing multiply/divide unit, sharing the operand forwarding paths. A two-bit CMALUMode parameter propagates from ibex_core to ibex_ex_block to select **Mode-0/Mode-1/Mode-2** at synthesis time. A single $W=23$ datapath serves all three schemes; the CFG instruction programs the active modulus (ML-KEM $q=3329$, ML-DSA $q=8380417$, or Falcon $q=12289$) and the corresponding bit-width at run time.

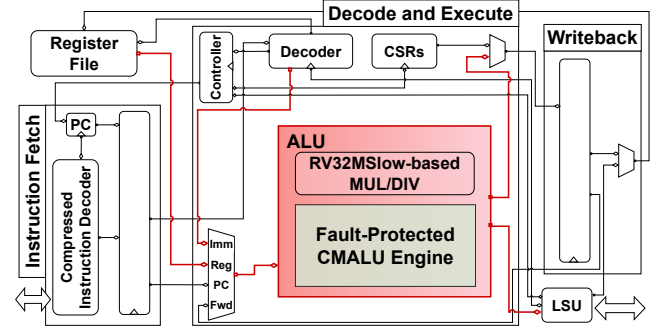


Figure 4: Integration of CMALU into the Ibex execution block with three custom modular instructions (MMUL, MADD, MSUB) and a CFG instruction for runtime scheme selection.

6 Experiments and Evaluations

The CMALU architecture is modeled in Verilog HDL using Vivado 2022.2. FPGA results are reported after the post-place-and-route stage on the Artix-7 (xc7a100tcs324-3) device. ASIC results for the NTT accelerator use 65 nm technology with Cadence Genus, while the Ibex core is synthesized with Yosys/OpenSTA on NanGate45 45 nm. Since CMALU targets resource-constrained IoT devices, the Ibex core is configured with a DSP-free iterative multiplier as a baseline. CMALU design is fully LUT/CARRY-based and does not utilize DSP slices, making it platform-agnostic. The NTT accelerator is synthesized independently for each scheme (ML-KEM, ML-DSA, Falcon), while the Ibex integration uses a single $W = 23$ CMALU instance covering all three schemes at runtime. Table 2 presents the FPGA results, where power is estimated using Vivado’s vectorless analysis, while the ASIC results are reported in Table 3.

Table 2: FPGA results @ Artix-7 after place-and-route. M0/M1 = Mode-0/Mode-1.

Platform	Scope	LUTs		FFs		BRAM	Power (W)	
		M0	M1	M0	M1		M0	M1
NTT Accelerator @ 200 MHz								
ML-KEM	NTT Top	473	596	337	341	1	0.106	0.109
	└ CMALU ($W = 12$)	244	362	128	123			
ML-DSA	NTT Top	745	991	502	525	2	0.134	0.143
	└ CMALU ($W = 23$)	473	697	184	208			
Falcon	NTT Top	553	701	374	378	1	0.119	0.128
	└ CMALU ($W = 14$)	298	446	130	135			
Ibex RISC-V Core (RV32MSlow, ML-KEM/ML-DSA/Falcon) @ 50 MHz								
Baseline	SoC	5,891		5,280		32	0.310	
	└ Ibex Core	3,459		1,370				
+CMALU	SoC	6,252	6,473	5,498	5,530	32	0.308	0.310
	└ Ibex Core ($W = 23$)	3,815	4,023	1,588	1,620			

Table 3: Synthesis results for ASIC. M0/M1 as in Table 2.

Platform	Scope	Area (μm^2)		Freq (MHz)	Power (mW)	
		M0	M1		M0	M1
NTT Accelerator (65 nm Cadence Genus)						
ML-KEM	NTT Top ($W = 12$)	164,435	171,673	970	90.7	103.9
ML-DSA	NTT Top ($W = 23$)	276,011	297,721	666	146.2	153.2
Falcon	NTT Top ($W = 14$)	367,854	369,469	715	158.2	169.0
Ibex Core (Nangate45 45 nm, Yosys/OpenSTA, $W=23$)						
Baseline	Ibex Core	15,864		250	19.13	
+CMALU	Ibex Core ($W = 23$)	19,190	20,116	250	18.52	17.10

6.1 NTT Accelerator Evaluation

6.1.1 FPGA Area, Latency, and Energy. As shown in Table 2, CMALU contributes 60–70% of the LUTs in the complete NTT/iNTT block in **Mode-1**, confirming that the arithmetic datapath dominates the area cost. In a radix-2 ping-pong configuration, **Mode-1** NTT/iNTT completes in 29,197, 44,557, and 79,373 cycles for ML-KEM, ML-DSA, and Falcon, respectively, which matches **Mode-0** and confirms zero latency overhead. **Mode-2** increases cycle count by $\approx 1.86\times$ (e.g., 54,285 cycles for ML-KEM). At 200 MHz, this corresponds to latencies of 145.98, 222.78, and 396.86 μs , with energy consumption of 15.9, 31.9, and 50.8 μJ per transform. Activity-based SAIF annotation (ML-KEM) puts the energy overhead over the unprotected baseline at **Mode-0** 15.05 $\rightarrow$ **Mode-1** 15.48 μJ (+2.9%, zero-latency) $\rightarrow$ **Mode-2** 29.18 μJ (+94%, recomputation); the vectorless per-scheme figures above are conservative upper estimates. Note that as all three modes in CMALU share the same arithmetic datapath and the checkers sit off the critical path, the added checkers did not reduce the target operating frequency in the evaluated configurations. The performance ordering directly reflects the operation schedule: MADD/MSUB completes in two cycles on the shared CSA/CLA pipeline, while MMUL executes for $W = \lceil \log_2 q \rceil$ iterations in the bit-serial Montgomery loop. Since CMALU targets compact IoT endpoints rather than line-rate accelerators, sub-millisecond NTT latency and whole-application speedup on Ibex (Section 6.2) are the primary deployment metrics.

6.1.2 ASIC Area, Latency, and Energy. In ASIC implementations, SRAMs dominate the area and energy budgets, representing 90–95% of total area across all schemes. At achieved frequencies of 970, 666, and 715 MHz for ML-KEM, ML-DSA, and Falcon, latencies reduce to 30.10, 66.90, and 111.01 μs with energies of 3.12, 10.24, and 18.76 μJ per transform, significant reductions over FPGA while preserving the same datapath schedule. The **Mode-2** latency overhead ($\approx 1.86\times$) applies equally to the ASIC, as the cycle count is determined solely by the datapath schedule. Its system-level resource impact is reported for the Ibex integration in Section 6.2, which serves as the complete reference platform for **Mode-2**.

6.1.3 Cross-Platform Robustness Metrics. Since the proposed CMALU is a general-purpose arithmetic unit for IoT devices rather than a specialized NTT engine, direct comparison with existing NTT accelerators is not meaningful. To characterize cross-platform efficiency, we define three internal normalization metrics. For a radix-2 NTT/iNTT with polynomial degree n , the total number of butterfly operations is $N_{\text{bf}} = \frac{n}{2} \log_2 n$. The average cycles per butterfly is $\text{CpB} = \text{CCs} / N_{\text{bf}}$, a technology-independent measure of computational cost. Using the measured per-transform energy $E_{\text{NTT}} = P_{\text{avg}}$,

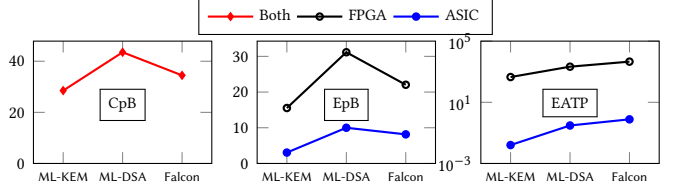


Figure 5: Cross-platform robustness metrics of the NTT accelerator (Mode-1). Cycles per butterfly (CpB) is technology-independent; energy per butterfly (EpB, nJ/butterfly) and the energy–area–throughput product (EATP, $\mu\text{J}\cdot\text{mm}^2\cdot\text{s}$) illustrate the ASIC energy advantage.

t_{NTT} , the energy per butterfly is $\text{EpB} = E_{\text{NTT}} / N_{\text{bf}}$, and the energy–area–throughput product is $\text{EATP} = (E_{\text{NTT}} \cdot A) / \text{Throughput}$, where A is the total area in mm^2 . As summarized in Figure 5, CpB remains consistent across FPGA and ASIC, confirming dependence on controller scheduling and datapath design, while EpB and EATP confirm superior energy efficiency of ASIC implementations.

6.2 Ibex Core Evaluation

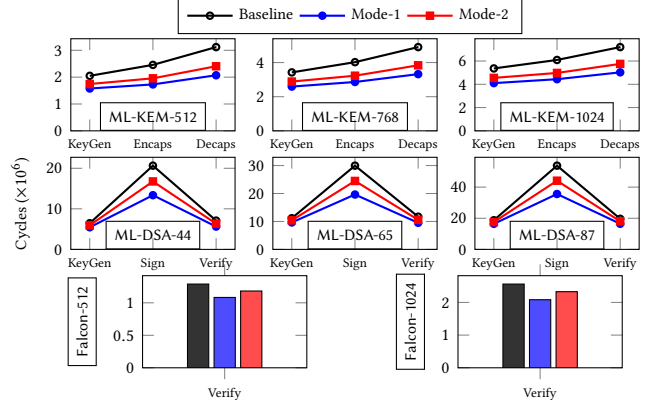


Figure 6: Clock cycles of PQC on the Ibex core (median of 100 runs at 50 MHz).

6.2.1 FPGA Area and Software Performance. Based on Table 2, integrating CMALU into `ibex_core` increases LUTs by 10.3%, 16.3%, and 18.8% for **Mode-0**, **Mode-1**, and **Mode-2**, respectively, with no measurable impact on total on-chip power (0.310 W). PQC schemes are compiled with -O3 and executed at 50 MHz on the Arty-A7-100T board. The reported cycle counts are the median of 100 runs. Figure 6 reports execution cycles across all security levels; **Mode-0** and **Mode-1** yield identical cycle counts, confirming zero latency overhead for online fault detection. In Falcon, the NTT is rarely utilized during KeyGen and not required for Sign; therefore, we only report Verify. **Mode-1** accelerates all reported operations over the Baseline, most notably 1.55 $\times$ on ML-DSA-44 Sign and 1.51 $\times$ on ML-KEM-512 Decaps. **Mode-2** incurs recomputation overhead but still outperforms the Baseline on every operation, achieving up to 1.30 $\times$ (ML-KEM-512 Decaps).

6.2.2 ASIC Area. From Table 3 on Nangate45 45 nm, the Ibex core area increases by +21.0% (**Mode-0**), +26.8% (**Mode-1**), and +28.6% (**Mode-2**) relative to the Baseline (15,864 μm^2), with a single $W=23$ CMALU instance covering all three schemes.

Table 4: Comparison with prior Ibex-based PQC research. None of the compared designs uses BRAM. ‘App.’ denotes ML-KEM-512 and ‘NTT’ denotes forward NTT speedup, both relative to each work’s own software baseline.

Design	Added Area				Speedup		Fault Det.
	LUT	FF	DSP	eSlice+	App.	NTT	
Celik* [11]	6,933	3,543	0	1,733	1.76× [§]	–	No
Gewehr* [17]	1,791	922	9	673	1.47× [*]	–	No
Bevin [‡] [10]	354	0	14	439	1.19× [§]	1.69× [°]	No
CMALU Mode-1 [‡]	564	250	0	141	1.42× [§]	2.56× [°]	Yes
CMALU Mode-2 [‡]	651	328	0	163	1.25× [§]	1.85× [°]	Yes

*ML-KEM [‡]ML-KEM/ML-DSA [‡]ML-KEM/ML-DSA/Falcon

[§]Total ML-KEM-512 [°]ML-KEM’s NTT ^{*}ML-KEM-512 encaps only

Table 5: System-level effective detection rate (% , ML-KEM W=12). Mode-0 provides no detection.

	SF	MF	BU	ST	RT	SDC
Mode-1 NTT	100.00	84.31	89.76	100.00	92.42	4.40
Mode-1 Ibex	100.00	83.15	86.36	100.00	78.57	9.78
Mode-2 (both)	100.00	100.00	100.00	100.00	100.00	0.00

Table 6: Direct comparison with prior Montgomery fault-detection. Differing widths/platforms locate CMALU’s design point rather than claim apples-to-apples superiority. Mode-1 area is the CMALU block (vs Mode-0); Mode-2 is the NTT-top increment (vs Mode-1).

	Aghapour [1]	CMALU Mode-1	CMALU Mode-2
Technique	Partial recomb. + input enc.	Parity + Mont.-evenness	Mode-1 + full recomb.
Target	2048-bit MP-Montgomery	PQC MMUL/MADD/MSUB	same
Added area	+3.0–5.7k LUT, 16 DSP	+118 LUT, 0 DSP	+69 LUT, 0 DSP
Latency	+10–25%	0%	+86%
Coverage	94–100% (C-model)	SF/ST 100%, MF/BU/RT 78–92%	all 100%, SDC 0%
Validation	C (GMP) simulation	RTL registers + processor	same

6.2.3 Comparison with Prior Ibex-Based PQC Accelerators. Table 4 compares CMALU with existing Ibex-based PQC accelerators. Since the compared designs use heterogeneous resource mixes (LUT, FF, DSP) and the original eSlice metric of [5] does not account for DSP usage, we heuristically extend it with a DSP term to avoid treating DSP-heavy and DSP-free designs as area-equivalent, reporting eSlice+ as a coarse normalization rather than a technology-exact area model: $\text{eSlice+} = \max(\frac{\text{LUT}}{4} + \text{BRAM} \cdot 128 + \text{DSP} \cdot 25, \frac{\text{FF}}{8})$. The area and speedup figures for [17] are measured from our reproduction of its Ibex integration. Our **Mode-1** achieves 1.42× overall and 2.56× NTT kernel speedup for ML-KEM with only 564 LUTs, 250 FFs, and zero DSPs, yielding the lowest eSlice+ (141) among the compared designs while additionally covering ML-DSA and Falcon. Unlike all prior works, CMALU integrates configurable fault detection (**Mode-1/Mode-2**) at zero latency overhead, addressing a security dimension not considered in existing Ibex-based PQC accelerators.

6.3 Fault Injection Evaluation

6.3.1 Target Selection for Fault Analysis. We evaluate CMALU fault detection on two case studies. The NTT accelerator is synthesized for ML-KEM ($W=12$) and targets CMALU during the CT butterfly in the forward NTT, exercising MMUL, MADD, and MSUB. The Ibex

core executes a full ML-KEM-512 decapsulation, with all modular arithmetic mapped to CMALU custom instructions. Decapsulation is chosen because key generation and encapsulation are dominated by public-matrix sampling that largely bypasses CMALU, whereas decapsulation concentrates polynomial multiplication on the CMALU datapath, maximizing the exploitable attack surface.

6.3.2 Fault Injection Methodology. Faults are injected into the five CMALU datapath registers (C, E, D, P, F) using a Verilator 4.228 C/C++ wrapper with 10,000 trials per fault-model configuration. We evaluate five fault models: SF, MF, BU, ST, and RT. An injection is classified as *effective* if it alters the architectural output. The detection rates are reported among effective faults only. Across all fault-free baseline (**Mode-0**) runs, no checker was triggered, confirming a 0% false-positive rate. The per-model effective fault counts are $n_{\text{eff}}=2,540$ –6,550 (NTT) and 1,680–2,040 (Ibex); Wilson 95% CI half-widths are $\leq \pm 0.2\%$ for SF/ST and $\leq \pm 2.0\%$ for MF/BU/RT.

6.3.3 Theory–experiment Consistency. Table 5 summarizes effective detection rates for **Mode-1** and **Mode-2**, consistent with the formal claims of Section 4.4. SF and ST achieve 100% detection in both case studies, confirming Theorem 1 and Lemma 1 under the checker-active injection window. The MF, BU, and RT faults achieve 78–92% detection, exceeding the theoretical lower bounds of Equation 4–Equation 5 ($> 3/4$ for MMUL and $> 1/2$ for MADD/MSUB). Three of the five target registers (C, E, F) are injected only during MMUL cycles, so the aggregate is dominated by the MMUL bound. The low-est observed rate (Ibex RT: 78.57%) still exceeds the 75% floor, reflecting the broader operand distribution of decapsulation. **Mode-2** achieves 100% effective detection with 0% SDC across the reported fault models on both evaluated platforms, confirming Theorem 2 for the campaigns considered: a one-pass transient produces $\epsilon_1 \neq \epsilon_2$, guaranteeing a mismatch, while permanent stuck-at faults ($\epsilon_1=\epsilon_2$) are already caught by the **Mode-1** invariant checker.

These register-level campaigns directly instantiate the arithmetic manifestation of the three motivating attacks of Section 3.2. **Attack 1** (butterfly DFA) is a transient perturbation of the butterfly operands (C, D, E), detected by **Mode-2** at 100% (SDC 0%) at the NTT system level. **Attack 2** (BSM replay) reuses the same W -cycle datapath, where SF/ST reach 100% (**Mode-1**) and all models 100% (**Mode-2**) on *every* run, blocking the repeated faulty-output collection the attack depends on. **Attack 3** (Montgomery reduction skip) is the stuck-at correction-bit F manifestation, covered by ST injection on F at 100%.

Mode-1 cannot eliminate all multi-bit misses: even-weight masks with $\Delta[0]=0$ can evade both checkers simultaneously (Equation 3). Nevertheless, **Mode-1** operates at zero latency overhead with only lightweight combinational logic and no datapath duplication, matching **Mode-0** cycle-for-cycle. For applications requiring complete transient coverage, **Mode-2** recomputation closes the residual gap at an 1.86× latency cost.

6.3.4 Comparison with Prior Montgomery Fault-Detection. Table 6 locates CMALU against the most recent Montgomery fault-detection design, Aghapour et al. [1]. As the target width (2048-bit RSA-class versus PQC-scale moduli) and validation level differ, we claim

no apples-to-apples superiority but a distinct design point: zero-latency, DSP-free detection covering add/sub alongside multiplication, validated on synthesizable RTL within an integrated processor rather than C-model simulation. In percentage terms **Mode-1**'s area overhead is higher, but its absolute LUT increase is below 1/25 of [1] with zero DSPs.

Acknowledgments

This work was partially supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00441762, Global Advanced Cybersecurity Human Resources Development, 50%) and the National Research Foundation of Korea (NRF) grant funded by MSIT (No. 2022R1C1C1013368, 50%), and the EPSRC's funded Integrated Quantum Networks (IQN) Hub in the UK (EP/Z533208/1).

References

- [1] Saeed Aghapour, Kasra Ahmadi, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2025. Partial Recomputation Fault Detection Architecture for Multiple-Precision Montgomery Modular Multiplication. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* (2025).
- [2] Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2025. Efficient Algorithm-Level Error Detection for Number-Theoretic Transform Used for Kyber Assessed on FPGAs and ARM. *ACM Trans. Embed. Comput. Syst.* 24, 5 (2025), 1–23.
- [3] Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, and Sujoy Sinha Roy. 2022. KaLi: A crystal for post-quantum security using Kyber and Dilithium. *IEEE Trans. Circuits Syst. I: Regul. Pap.* 70, 2 (2022), 747–758.
- [4] Erdem Alkim, Hülya Evkan, Norman Lahr, Ruben Niederhagen, and Richard Petri. 2020. ISA Extensions for Finite Field Arithmetic Accelerating Kyber and NewHope on RISC-V. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 3 (2020), 219–242.
- [5] Francesco Antognazza, Alessandro Barengi, Gerardo Pelosi, and Ruggero Susella. 2024. A high efficiency hardware design for the post-quantum KEM HQC. In *2024 IEEE Int. Symp. on Hardware Oriented Security and Trust (HOST)*. IEEE, 431–441.
- [6] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. 2019. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 4 (Aug. 2019), 17–61. doi:10.13154/tches.v2019.i4.17-61
- [7] Sven Bauer and Fabrizio De Santis. 2023. A differential fault attack against deterministic falcon signatures. In *Int. Conf. on Smart Card Research and Advanced Applications (CARDIS)*. Springer, 43–61.
- [8] Sven Bauer, Fabrizio De Santis, Kristjane Koleci, and Anita Aghaie. 2025. A Fault-Resistant NTT by Polynomial Evaluation and Interpolation: Application to ML-KEM and ML-DSA. In *Constructive Approaches for Security Analysis and Design of Embedded Systems (CASCADE 2025)*. Springer, 211–231. doi:10.1007/978-3-032-01405-4_9
- [9] Jonas Bertels, Quinten Norga, and Ingrid Verbauwhede. 2024. A Better Kyber Butterfly for FPGAs. In *2024 34th Int. Conf. on Field-Programmable Logic and Applications (FPL)*. IEEE, 171–177.
- [10] Ryan Bevin, Ayesha Khalid, Malik Imran, and Máire O'Neill. 2025. Accelerating CRYSTALS-Kyber and Dilithium via a Single Montgomery Reduction ISE on RISC-V. In *2025 IEEE 38th Int. System-on-Chip Conf. (SOCC)*. IEEE, 1–6.
- [11] Ahmet Celik, Fatih Yilmaz, Mehmet Anil Korkmaz, and Berna Ors. 2023. Implementation of CRYSTALS-Kyber Post-Quantum Algorithm Using RISC-V Processor. In *2023 30th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*. IEEE, 1–4.
- [12] Hao Cheng, Georgios Fotiadis, Johann Großschädl, Daniel Page, Thinh H Pham, and Peter YA Ryan. 2024. RISC-V Instruction Set Extensions for Multi-Precision Integer Arithmetic: A Case Study on Post-Quantum Key Exchange Using CSIDH-512. In *Proc. of the 61st ACM/IEEE Design Automation Conference (DAC)*. 1–6.
- [13] James W. Cooley and John W. Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.* 19, 90 (1965), 297–301.
- [14] Alessandra Dolmeta, Stefano Di Matteo, Emanuele Valea, Mikael Carmona, Antoine Loiseau, Maurizio Martina, and Guido Masera. 2025. TYRCA: A RISC-V Tightly-coupled accelerator for Code-based Cryptography. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.
- [15] ETH Zurich and University of Bologna and lowRISC. 2025. Ibex: An Open-Source 32-bit RISC-V Core. <https://ibex-core.readthedocs.io/en/latest/>. Accessed: 2025-11-17.
- [16] W. M. Gentleman and G. Sande. 1966. Fast Fourier Transforms: For Fun and Profit. In *Proc. of the 1966 Fall Joint Computer Conference*. ACM, 563–578.
- [17] Carlos Gewehr, Lucas Luza, and Fernando Gehm Moraes. 2024. Hardware acceleration of crystals-kyber in low-complexity embedded systems with RISC-V instruction set extensions. *IEEE Access* 12 (2024), 94477–94495.
- [18] Lewis Glabush, Felix Günther, Kathrin Hövelmanns, and Douglas Stebila. 2025. Verifiable Decapsulation: Recognizing Faulty Implementations of Post-quantum KEMs. In *Annual International Cryptology Conference*. Springer, 543–574.
- [19] Arash Hariiri and Arash Reyhani-Masoleh. 2010. Concurrent error detection in Montgomery multiplication over binary extension fields. *IEEE Trans. Comput.* 60, 9 (2010), 1341–1353.
- [20] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. 2021. Fault-enabled chosen-ciphertext attacks on Kyber. In *International Conference on Cryptology in India*. Springer, 311–334.
- [21] Vincent Hwang, YoungBeom Kim, and Seog Chung Seo. 2025. Multiplying Polynomials without Powerful Multiplication Instructions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 1 (2025), 160–202.
- [22] Malik Imran, Safiullah Khan, Ayesha Khalid, Ciara Rafferty, Yasir Ali Shah, Samuel Pagliarini, Muhammad Rashid, and Máire O'Neill. 2024. Evaluating NTT/INTT Implementation Styles for Post-Quantum Cryptography. *IEEE Embed. Syst. Lett.* 16, 4 (2024), 485–488. doi:10.1109/LES.2024.3410516
- [23] Sönke Jendral, John Preuß Mattsson, and Elena Dubrova. 2024. A single-trace fault injection attack on hedged module lattice digital signature algorithm (ML-DSA). In *2024 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 34–43.
- [24] Marc Joye, Michael Tunstall, et al. 2012. *Fault analysis in cryptography*. Vol. 147. Springer.
- [25] Kyungkyun Kang, Seulbee Yang, Giang Truong Le, and Hanho Lee. 2025. HLS-based Hardware/Software Co-design of ML-KEM Post-quantum Cryptosystem for Real-time Video Encryption. *J. Semicond. Technol. Sci.* 25, 5 (2025), 530–541.
- [26] Safiullah Khan, Ayesha Khalid, Ciara Rafferty, Yasir Ali Shah, Maire O'Neill, Wai-Kong Lee, and Seong Oun Hwang. 2023. Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators. In *2023 IFIP/IEEE 31st Int. Conf. on Very Large Scale Integration (VLSI-SoC)*. 1–6. doi:10.1109/VLSI-SoC57769.2023.10321885
- [27] YoungBeom Kim and Seog Chung Seo. 2025. An optimized instantiation of post-quantum mqtt protocol on 8-bit avr sensor nodes. In *Proc. of the 20th ACM Asia Conf. on Computer and Communications Security (ASIA CCS)*. 248–266.
- [28] Gaoyu Mao, Donglong Chen, Guangyan Li, Wangchen Dai, Abdurashid Ibrahim Sanka, Çetin Kaya Koç, and Ray CC Cheung. 2023. High-performance and configurable SW/HW co-design of post-quantum signature CRYSTALS-Dilithium. *ACM Trans. Reconfigurable Technol. Syst.* 16, 3 (2023), 1–28.
- [29] Peter L. Montgomery. 1985. Modular Multiplication Without Trial Division. *Math. Comp.* 44, 170 (1985), 519–521.
- [30] National Institute of Standards and Technology (NIST). 2024. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)*. Technical Report. NIST. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf>
- [31] National Institute of Standards and Technology (NIST). 2024. *FIPS 204: Module-Lattice-Based Digital Signature Algorithm (ML-DSA)*. Technical Report. NIST. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>
- [32] Yi Ouyang, Yihong Zhu, Wenping Zhu, Bohan Yang, Zirui Zhang, Hanning Wang, Qichao Tao, Min Zhu, Shaojun Wei, and Leibo Liu. 2025. FalconSign: An Efficient and High-Throughput Hardware Architecture for Falcon Signature Generation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 1 (2025), 203–226.
- [33] Thomas Prest, Pierre-Alain Fouque, Vadim Lyubashevsky, Tancrede Lepoint, Gregor Seiler, and Damien Stehlé. 2023. Falcon: Fast-Fourier Lattice-Based Compact Signatures. <https://falcon-sign.info/>.
- [34] Prasanna Ravi, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. 2023. Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023, 2 (2023), 447–481.
- [35] A Supraja Reddy, B. Khalelu Rehman, P. Anuradha, K. Venkata Siva Reddy, Mudasar Basha, and Mohammed Abdul Nasar. 2025. Designing a High-Speed, Low-Area Three-Operand Binary Adder using Kogge-Stone, Han-Carlson, and Ladner-Fischer VLSI Architectures. In *2025 4th Int. Conf. on Innovative Mechanisms for Industry Applications (ICIMIA)*. IEEE, 98–104. doi:10.1109/ICIMIA67127.2025.11200640
- [36] Peter W. Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* 41, 2 (1999), 303–332.
- [37] Tengfei Wang, Chi Zhang, Pei Cao, and Dawu Gu. 2022. Efficient implementation of dilithium signature scheme on FPGA SoC platform. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 30, 9 (2022), 1158–1171.
- [38] Tengfei Wang, Chi Zhang, Xiaolin Zhang, Dawu Gu, and Pei Cao. 2024. Optimized hardware-software co-design for Kyber and Dilithium on RISC-V SoC FPGA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024, 3 (2024), 99–135.