POLITECNICO DI TORINO
Repository ISTITUZIONALE

Raise the Shields: A Modular RISC-V Extension for Post Quantum Cryptography

Original
Raise the Shields: A Modular RISC-V Extension for Post Quantum Cryptography / Dolmeta, A., Piscopo, V., Martina, M., Masera, G., Hutter, M.. - ELETTRONICO. - (2026). (Design and Automation Conference 2026 Long Beach (USA) 26 - 29 July 2026) [10.1145/3770743.3804373].

Availability:
This version is available at: 11583/3016929 since: 2026-10-24T10:30:48Z

Publisher:
Association for Computing Machinery

Published
DOI:10.1145/3770743.3804373

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

17 November 2026

Raise the Shields: A Modular RISC-V Extension for Post Quantum Cryptography

Alessandra Dolmeta
alessandra.dolmeta@polito.it
Politecnico di Torino
Torino, Italy

Valeria Piscopo
valeria.piscopo@polito.it
Politecnico di Torino
Torino, Italy

Maurizio Martina
maurizio.martina@polito.it
Politecnico di Torino
Torino, Italy

Guido Masera
guido.masera@polito.it
Politecnico di Torino
Torino, Italy

Michael Hutter
michael.hutter@unibw.de
University of the Bundeswehr
Munich & PQShield

Abstract

This work presents a unified RISC-V extension for Post-Quantum Cryptography (PQC) that emphasizes versatility and openness. The design exposes a compact set of custom scalar instructions via the Core-V eXtension Interface (CV-X-IF) and a modular PQ-ALU that accelerates the dominant kernels across *hash-*, *lattice-*, and *code-*based schemes: Keccak processing, randomness sampling, modular and polynomial arithmetic, finite-field operations, and coefficient compression. The system supports algorithms standardized by NIST, ML-KEM, ML-DSA, SLH-DSA, and HQC, as well as candidates under evaluation. The full hardware and software stack is released as open source to enable reproducibility and community reuse. We provide ASIC results from post-synthesis characterization in 65 nm CMOS, reporting instruction-cycle counts, along with power and energy estimates for each custom operation. Overall, the proposed extension delivers a compact (26.3 kGE) and practical path to crypto-agile, energy-efficient PQC on RISC-V while preserving software compatibility and a clean integration into existing cores.

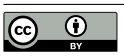
CCS Concepts

• **Hardware → Very large scale integration design**; • **Security and privacy → Cryptography**.

Keywords

RISC-V, ISE, Post-Quantum Cryptography, ASIC.

ACM Reference Format:
Alessandra Dolmeta, Valeria Piscopo, Maurizio Martina, Guido Masera, and Michael Hutter. 2026. Raise the Shields: A Modular RISC-V Extension for Post Quantum Cryptography. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804373



This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804373

1 Introduction

In the last two decades, quantum computing has progressed from theory to practice. Grover's [1] and Shor's [2] algorithms pose concrete risks, the former providing quadratic speedup for exhaustive search, the latter enabling efficient integer factorization and discrete logarithm. Grover can be countered by enlarging symmetric keys, but Shor fundamentally breaks RSA, DSA, Diffie–Hellman, and ECC. The potential collapse of today's public-key infrastructure has led to an intense global research effort to develop cryptographic mechanisms that remain secure in the presence of quantum adversaries[3] [4]. This field, known as Post-Quantum Cryptography (PQC), encompasses five main families of schemes: *code-based*, *hash-based*, *lattice-based*, *multivariate*, and *isogeny-based* cryptography. Since 2016, the National Institute of Standards and Technology (NIST) has conducted a multi-round standardization process for PQC, culminating in the selection of leading schemes now being standardized—namely ML-KEM (Kyber), ML-DSA (Dilithium), SLH-DSA (SPHINCS+), and HQC —while continuing to assess additional candidates and emerging signature schemes such as CROSS and LESS [5]. Beyond their theoretical resilience, these algorithms introduce significant implementation challenges. In contrast to classical public-key schemes that perform relatively simple modular arithmetic on very large integers (as in RSA or ECC), PQC schemes rely on *dense sequences of fine-grained operations* over small finite fields or rings, often involving thousands of polynomial coefficients. This leads to a much higher level of computational parallelism and data movement, which can heavily burden general-purpose processors. At the same time, the limited operand size (typically below 20 bits) makes these micro-operations well-suited for acceleration through dedicated Instruction-Set Extensions (ISEs) and lightweight hardware datapaths. As a result, PQC acceleration focuses not on enlarging arithmetic precision, but on efficiently mapping the vast number of small, structured arithmetic operations that dominate these quantum-resilient algorithms. Existing implementations of PQC can be classified into three main categories: software-based, dedicated hardware, and hardware–software co-designs. Software libraries provide portable and maintainable reference implementations for general-purpose processors. Projects such as PQClean[1], liboqs[2], and libpqcrypto[3] include efficient and verified code for many

[1] https://github.com/PQClean/PQClean
[2] https://openquantumsafe.org/#liboqs
[3] https://libpqcrypto.org/

NIST candidates. While these frameworks are crucial for algorithm benchmarking, they are limited by the performance and energy constraints of general-purpose CPUs. To overcome these limitations, numerous hardware accelerators and Instruction-Set Extensions (ISEs) have been proposed. ISEs enhance general-purpose ISAs with domain-specific arithmetic primitives, such as modular multiplication, reduction, or NTT (Number Theoretic Transform) computation, offering a middle ground between software flexibility and hardware speed. Early initiatives by the RISC-V Foundation in 2018 led to the creation of a Security Standing Committee to standardize cryptographic extensions. This was followed by the *XCrypto* project in 2019, which proposed instructions for classical cryptography[4]. However, comparable standardized support for post-quantum algorithms is still lacking, motivating recent research on PQC-oriented ISEs. Several works have explored this direction. [6], [7–10], and [11] introduced RISC-V extensions and co-processors targeting *lattice-based* schemes, which are NTT-based algorithms, while [12] and [13] extended this concept to SLH-DSA and general-purpose polynomial arithmetic. Hybrid hardware–software co-designs such as [14] for *code-based* HQC, and [15] for multiple PQC algorithms, combine instruction-level acceleration with dedicated submodules, achieving strong performance–flexibility trade-offs.

Our contribution. This work presents one of the first unified Post-Quantum Cryptography Instruction-Set Extensions (PQC-ISE) integrated with a tightly coupled coprocessor architecture. The proposed design systematically addresses all the main computational bottlenecks across standardized and under-standardization schemes, including ML-KEM, ML-DSA, SLH-DSA, Falcon, NewHope, HQC, CROSS, and LESS, by supporting hashing, sampling, modular reduction, polynomial arithmetic, and finite-field operations within a single ISA framework. Unlike prior FPGA-only studies, this work includes a full ASIC synthesis and characterization, with post-synthesis results featuring detailed power and energy analysis for each instruction. This evaluation highlights the efficiency and scalability of the proposed extension as a practical foundation for quantum-resilient RISC-V systems. Moreover, the source code for all our implementations is available open-source. [5]

2 Background

Subection 2.1 reviews standardized PQC and their kernels, while Subsection 2.2 surveys RISC-V PQC hardware solutions.

2.1 Post-Quantum Cryptography Overview

In response to the threat posed by quantum computers to classical public-key cryptography, the U.S. National Institute of Standards and Technology (NIST) launched a multi-round process to standardize quantum-resistant algorithms. The initiative aims to define new primitives that remain secure against quantum attacks while remaining efficient on modern hardware; specifically, Key Encapsulation Mechanisms (**KEM**) for secure key exchange and Digital Signature (**DS**) schemes for authentication. After several evaluation rounds, NIST selected three algorithms for standardization:

- ML-KEM and ML-DSA, both *lattice-based*, now published as **FIPS-203** [16] and **FIPS-204** [17], respectively;
- SLH-DSA, hash-based standardized as **FIPS-205** [18].

In parallel, HQC [19] and Falcon [20], a code-based KEM and a lattice-based signature scheme, respectively, have recently been selected for future inclusion. Together, these families represent the most promising foundations for PQC. Recently, NIST has launched an additional *on-ramp* process for further post-quantum DS schemes, to both diversify beyond the currently selected *lattice-based* and *hash-based* standards, and solicit proposals with short signatures or fast verification (i.e., CROSS [21] and LESS [22]). Although the algebraic principles behind these schemes differ, they share a set of recurring computational motifs that dominate execution time and hardware cost. The main performance bottlenecks of the different algorithms mentioned are summarized below.

Hash functions and Keccak-based primitives. All NIST PQC algorithms make extensive use of cryptographic hash functions for pseudorandom generation, key derivation, and signature construction. This involves multiple invocations of SHA-3 and its extendable-output variants SHAKE, both derived from the Keccak permutation[23]. In ML-KEM and ML-DSA, Keccak is used to generate public matrices, sample error polynomials, and compute message digests; in HQC, it drives seed expansion and error vector generation. Hashing is even more critical in SLH-DSA, which builds almost entirely on Keccak-based or related primitives for Winternitz chains and Merkle authentication paths. Emerging schemes such as Falcon, CROSS, and LESS also rely on intensive hash operations for sampling, compression, and signature verification, confirming hashing as a universal bottleneck across both standardized and under-standardization PQC algorithms.

Sampling and randomness generation. Many PQC schemes require the generation of small random values following specific statistical distributions to ensure security. *Lattice-based* schemes use Centered-Binomial Distributions (CBD) to sample error polynomials. Signature schemes use rejection sampling to enforce norm bounds, adding control-flow variability and extra hashing; *code-based* schemes draw error vectors with fixed Hamming weight. All rely on Keccak-driven pseudorandom expansion [24].

Finite-field and modular arithmetic. Arithmetic over finite rings and fields forms the backbone of many PQC computations. In *lattice-based* algorithms, polynomial multiplications over rings such as $\mathbb{Z}_q[x]/(x^n+1)$ are accelerated via the *Number Theoretic Transform (NTT)* and its inverse (*INTT*), with modular reductions (e.g., Montgomery or Barrett) applied at each stage to keep intermediates bounded. In *code-based* schemes, the dominant cost is large-degree binary polynomial multiplication over $\mathbb{F}_2[x]$, often implemented with vectorized bitwise operations or Galois-field accelerators. Together, these operations define the core compute kernels that benefit most from domain-specific acceleration [25].

In summary, PQC workloads are characterized by a tight combination of hash-centric operations, statistical sampling, and ring/-field arithmetic, each exhibiting distinct instruction-level and memory-access patterns. These patterns directly motivate the Instruction-Set Extensions proposed in section 3.

2.2 RISC-V and PQC Instruction Set Extentions

The open-source **RISC-V** Instruction Set Architecture (ISA) has become a cornerstone for hardware–software co-design in security and cryptography. Its modular structure and standardized support

[4] https://github.com/scarv/xcrypto
[5] https://github.com/vlsi-lab/HORCRUX/tree/asic_branch

for custom Instruction-Set Extensions (ISEs) make it particularly suitable for domain-specific acceleration, allowing designers to extend the base ISA with dedicated operations while maintaining toolchain and software compatibility. Unlike fixed commercial ISAs, RISC-V provides reserved opcode spaces that can be leveraged to directly interface the processor pipeline with external accelerators or co-processors, enabling near-zero-latency offloading of complex cryptographic functions. A broad effort has explored RISC-V acceleration for PQC, with a growing shift from FPGA prototypes to **ASIC** implementations to assess silicon cost, frequency, and energy. Early Kyber co-processors at 65 nm demonstrated compact NTT/reduction engines [8], followed by larger unified Kyber–Dilithium designs with shared hashing/polynomial hardware [26]. Several 28–55 nm Kyber-focused works refine the same bottlenecks (NTT, Barrett/-Montgomery reduction, and sampling), ranging from ultra-compact to high-throughput pipelines [9, 27, 28], and a joint Kyber/Dilithium co-processor with unified datapaths [29]. For Dilithium specifically, lightweight ISEs map core polynomial/modular ops into a scalar ALU [11]. Beyond lattices, versatile processors target SPHINCS+ hashing pipelines and system integration [12, 13], while a 28 nm microcoded design supports all four NIST standards (ML-KEM, ML-DSA, SLH-DSA, HQC) via custom opcodes [15]. These ASIC implementations collectively highlight the growing maturity of RISC-V–based PQC hardware. They share a set of unifying design themes: (i) modular arithmetic cores, (ii) hash acceleration, and (iii) flexible integration via ISA extensions rather than memory-mapped units. Each represents a step toward general-purpose, crypto-agile PQC co-processors capable of supporting multiple algorithm families within a unified hardware substrate.

In this landscape, the proposed design realizes a crypto-agile ASIC co-processor in 65 nm CMOS. The co-processor is controlled through a single, compact PQC-ISE and features a modular datapath that spans the dominant kernels of *hash-*, *lattice-*, and *code-based* cryptography, including Keccak-based hashing, sampling, *NTT/INTT* with Montgomery/Barrett reduction, finite-field arithmetic, and binary-polynomial operations. Rather than tailoring separate accelerators to individual schemes, the architecture is explicitly structured to reuse these blocks across multiple algorithms. Tight integration with the RISC-V core is achieved via the **Core-V eXtension Interface** (CV-X-IF), which allows the co-processor to be coupled at the instruction level without modifying the CPU RTL [30]. CV-X-IF exposes well-defined sub-interfaces for issuing operations, exchanging operands, and committing results, so that complex PQC kernels can be offloaded through custom opcodes instead of using memory-mapped peripherals. In line with recent practice [14, 15, 31–34], our ASIC implementation includes post-synthesis characterization with per-instruction power and energy, demonstrating efficiency across diverse PQC workloads.

3 Design

The proposed architecture integrates a dedicated coprocessor tightly coupled to the RISC-V core through the CV-X-IF interface. This coprocessor, internally referred to as the **PQ-ALU**, serves as the computational heart of the system, executing the PQC-specific instructions decoded from the main pipeline. It comprises a collection of lightweight combinational and sequential modules—each optimized for a distinct cryptographic primitive. The coprocessor schematic is shown in Figure 1 and explained in subsection 3.1 and subsection 3.2. For clarity, in the following notations, $\oplus$ denotes bitwise XOR, $\neg$ denotes bitwise NOT, $\wedge$ denotes bitwise AND, and $>>$ represents logical shift-right operations.

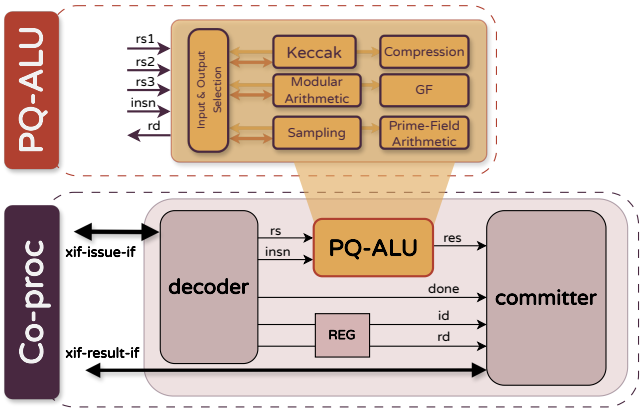


Figure 1: Coprocessor and PQ-ALU Schematics.

3.1 Decoder and Committer

Two control stages manage the coprocessor's interaction with the RISC-V core: the **decoder** and the **committer**. The decoder is responsible for parsing the incoming instruction word and selecting the corresponding execution module. It extracts operand values from the `xif-issue-if` structure and forwards them to the selected unit. This stage performs no arithmetic itself, it acts purely as a routing controller that guarantees the correct matching between instruction type and hardware block. Since the instruction map is defined in a SystemVerilog package, extending the ISA requires only the addition of a new case entry without altering the core logic. Once the computation is completed, the committer stage validates and synchronizes the output value, ensuring that results are written back to the RISC-V CPU register file in the correct program order and maintaining precise exception semantics and data consistency with standard RISC-V execution. From the processor's perspective, the completion of a custom instruction is indistinguishable from that of any native arithmetic operation.

3.2 PQ-ALU

The PQ-ALU is a modular datapath exposing multiple specialized computation units behind a uniform 32-bit interface. A custom instruction (`insn`) selects the target unit, while operands (`rs1`, `rs2`, `rs3`) are routed accordingly. All units operate in parallel but remain idle unless explicitly enabled, ensuring minimal dynamic power consumption. When the operation completes, the result (`rd`) is placed on `xif-result-if` and written back by the committer in program order. The current ALU integrates the following blocks:

Keccak Unit. Two dedicated modules implement the core bitwise transformations used in the ρ, θ, and χ steps of the Keccak-f[1600] permutation. The `rol32` block performs variable rotations on 64-bit words represented as two 32-bit halves (instructions `rolh32`, `roll32` in Table 1), supporting arbitrary offsets through a compact barrel-shifter structure. The `bcop32` block fuses the

Boolean function $A = B \oplus (\neg C \wedge D)$ into a single-cycle operation. Together, these modules enable the efficient execution of SHA-3 and SHAKE primitives within post-quantum schemes such as ML-KEM, ML-DSA, HQC, SLH-DSA, Falcon, CROSS, LESS, eliminating the need for multi-instruction bitwise sequences.

Modular Arithmetic Unit. The reduction module supports both Montgomery and Barrett-style modular reduction. Montgomery reduction is implemented through a compact, multiplier-based datapath optimized for lattice-based schemes such as ML-KEM, ML-DSA, Falcon, NTRU, and NewHope, using small signed multipliers together with shifts and sign extensions. In contrast, Barrett reduction follows a multiplier-free, shift-and-add formulation tailored to special primes, avoiding full-width products. Both modes share the same configurable datapath, with the instruction word selecting modulus parameters and the corresponding reduction algorithm.

The Montgomery reduction is employed for most lattice-based schemes due to its efficiency in modular arithmetic, computing:

$$t = a \cdot Q_{\text{inv}} \pmod{2^{16}}, \quad r = (a - t \cdot Q) \gg 16$$

where Q_{inv} is the precomputed modular inverse of the modulus Q with respect to 2^{16}, eliminating costly division by relying only on additions and bit shifts. The same datapath is reused across several parametrizations by loading the appropriate constants $\{Q, Q_{\text{inv}}\}$, making the unit configurable for multiple PQC schemes and supporting different Reduction instructions taken from Table 1.

- ML-KEM: $Q = 3329$, $Q_{\text{inv}} = 62209$ (`montgk`)
- NewHope: $Q = 12287$, $Q_{\text{inv}} = 12289$ (`montgn`)
- Falcon: $Q = 12289$, $Q_{\text{inv}} = 12287$ (`montgf`)
- NTRU: $Q = 49905$, $Q_{\text{inv}} = 4591$ (`montgt`)

These configurations cover power-of-two and near-Mersenne moduli, enabling a single hardware structure to handle diverse lattice arithmetic. For ML-DSA, a 64-bit version of the Montgomery reduction is used (`montgd`), as its modulus $Q = 58728449$ requires higher dynamic range. In this case, the input and intermediate products are 64-bit wide, but the reduction formula remains equivalent.

Barrett reduction is provided as a complementary mode and is implemented without multipliers, using a shift-and-add decomposition adapted to the special structure of ML-KEM 's modulus. The reduction is realized by expanding the term $(a \cdot \mu)/2^k$ into a sequence of controlled shifts and additions: $r = a - \left\lfloor \frac{a \cdot \mu}{2^k} \right\rfloor Q$, avoiding the need for a full product. This formulation exploits the pseudo-Mersenne (Parsenne-style) structure of $Q = 3329$ [35]. Barrett is the critical normalization step inside every NTT and INTT butterfly, ensuring that intermediate values remain in range. Its hardware implementation is therefore essential for accelerating ML-KEM 's polynomial arithmetic, motivating the dedicated NTT/INTT performance tests included in our evaluation.

Sampling Unit. We implement two hardware blocks to cover sampling patterns: centered–binomial noise for lattice schemes and uniform modulo-q candidates with in-hardware rejection. Both blocks follow standard constructions for lattice-based schemes [36, 37]. Both units are word–oriented and consume 32-bit chunks of pseudorandom data, but are profiled via a compact instruction so that the same RTL supports multiple parameter sets.

Centered–Binomial Sampling (CBD_η). For $\eta \in \{1, 2, 3, 4\}$, the classical CBD construction draws coefficients $a_i \in \mathbb{Z}$ as the difference of two Hamming weights over disjoint η-bit windows of a random bit-stream:

$$x = \sum_{j=0}^{\eta-1} b_{i\eta+j}, \qquad y = \sum_{j=0}^{\eta-1} b_{i\eta+\eta+j}, \qquad a_i = x - y.$$

Our `cbd_eta` module realizes exactly this (instructions `cbd1`–`cbd4`) in Table 1). The design is fully combinational, branchless, and constant-work per call (the variability in a_i is only in the *data*).

Uniform Mod-q Sampling with Rejection. Uniform coefficients in $\{0, \ldots, q-1\}$ are obtained by slicing fixed-width candidates from the Pseudo-Random Generator (PRG) stream and rejecting those that fall outside the target range. Our `rej` block realizes the common "fixed-slice, fixed-lane" pattern:

$$\texttt{rej}(r, m, b, k) \Rightarrow (t_0, \ldots, t_{k-1}), t_\ell = \begin{cases} c_\ell, & c_\ell < m, \\ 0, & \text{otherwise.} \end{cases}$$

where r is the 32 bit random word, m is the bound (typically q), b is the candidate bit-width, and k is the number of lanes sliced from r. Internally, the unit simply decodes the profile (b, k), slices k b-bit candidates from r, and writes those lower than m into fixed lane positions. Notably, rejected lanes are *zeroed in place* rather than compacted, yielding deterministic control flow and fixed write positions; any compaction can be deferred to a later lightweight pass. Profiles are table-driven, so adding new (b, k) tuples (e.g., `rej12`, `rej8`) requires only opcode decoding updates.

Compression Unit. The `compress` block quantizes ML-KEM coefficients (q=3329) to d bits by approximating $\lfloor (2^d/q), x \rfloor$ via fixed-point constants: a scaled multiply and right shift replace division, with rounding by adding $q/2$. To minimize area, the reciprocal multiply is implemented as shift-and-add. Four instructions, as shown in Table 1, cover ML-KEM bit-widths $d \in 4, 5, 10, 11$, optimized for low latency and area. The same mechanism can be easily reused in other lattice-based schemes (i.e., NTRU).

Galois-Field Unit. The Galois-Field (GF) block provides binary-field arithmetic for code-based/lightweight crypto (e.g., HQC). It comprises a 64-bit Karatsuba engine (instructions `kar1`–`kar4` in Table 1) for $\mathbb{F}_2$ polynomial products and a byte-wise carry-less multiplier (instruction `gfclmul` in Table 1) for $\mathbb{F}_{2^8}$. Karatsuba reduces partial products by splitting 64-bit operands into halves and computing three sub-products (z_0, z_1, z_2) with a fixed XOR/shift recombination. The `gfclmul` block, corresponding to `gfclmul` instruction in Table 1, performs byte-wise $\mathbb{F}_{2^8}$ multiplication using parallel XOR-and-shift slices with a fixed reduction. Both engines are constant-time and use compact, multiplier-free datapaths, ensuring high throughput with minimal area and power overhead.

Prime-Field Arithmetic Unit. This block provides $\mathbb{F}_p$ arithmetic for CROSS, covering R-SDP and R-SDP(G). A single datapath (parameterized for $p \in \{127, 509\}$) supports restricted mapping/normalization (instructions `r2v`, `r2vg` Table 1), add/mul and modular reduction via pointwise and matrix–vector kernels (instructions `vpw`, `vmul`, `rvmul`, `vpwg`, `vmulg`, `rvmulg` in Table 1), and scaled syndrome updates (instructions `syupd`, `syupg` in Table 1). This replaces multiple software routines with a compact, constant-time hardware implementation. All modules produce a single 32-bit result, which is then multiplexed into the shared output path according to the instruction identifier. The design adheres to RISC-V's scalar instruction formats (R-type and R4-type) and maintains full compatibility with the CV-X-IF communication protocol. This separation between

decoding, computation, and commitment allows the coprocessor to remain fully modular and easily extensible. Each instruction class can be developed, tested, or synthesized independently. At the same time, the surrounding control structure guarantees correct interaction with the RISC-V pipeline.

Table 1: Custom instruction mnemonics, brief computation, and operand type.

Instr.	Computation	Type
Keccak		
rolh32	$rd \leftarrow \mathrm{ROL64}(rs1 \,\|\| rs2, rs3)[63:32]$	R4
roll32	$rd \leftarrow \mathrm{ROL64}(rs1 \,\|\| rs2, rs3)[31:0]$	R4
bcop32	$rd \leftarrow rs1 \oplus (\neg rs2 \wedge rs3)$	R4
Reductions		
montgk	Montgomery mod 3329 (ML-KEM)	R
montgn	Montgomery mod q (NewHope)	R
montgf	Montgomery mod q (Falcon)	R
montgt	Montgomery mod q (NTRU)	R
montgd	Montgomery mod 8380417 (ML-DSA)	R
barrk	Barrett mod 3329 (ML-KEM)	R
Sampling		
cbd1	Centered binomial η=1	R
cbd2	Centered binomial η=2	R
cbd3	Centered binomial η=3	R
cbd4	Centered binomial η=4	R
rej12	Rejection: 12-bit pairs → coeffs	R4
rej23	Rejection: 23-bit → coeff	R4
rej8	Rejection: 8-bit quads → coeffs	R4
Compression *(ML-KEM)*		
compr1	Quantize coeff to d=4 bits	R
compr2	Quantize coeff to d=5 bits	R
compr3	Quantize coeff to d=10 bits	R
compr4	Quantize coeff to d=11 bits	R
GF Arithmetic		
gfclmul	$rd \leftarrow \mathrm{CLMUL}_{8\times8}(rs1, rs2)$ (byte-wise)	R4
kar1	Karatsuba64: low part of $rs1 \cdot rs2$	R
kar2	Karatsuba64: high part of $rs1 \cdot rs2$	R
kar3	Karatsuba64: mid (upper 32)	R
kar4	Karatsuba64: mid (lower 32)	R
Vector/Matrix *(CROSS)*		
r2v	Restricted-to-value mapping (RESTR_TO_VAL)	R4
vpw	Pointwise vector multiply + red.	R4
vmul	Vector–matrix multiply over $\mathbb{F}_p$	R4
rvmul	Restricted vector–matrix multiply	R4
syupd	Syndrome update $s \leftarrow s - \alpha v$	R4
r2vg	Restricted→value map (gen. powers)	R4
vpwg	G-variant pointwise vector multiply + red.	R4
vmulg	Vector–matrix multiply (p=509)	R4
rvmulg	Restricted vector–matrix mult. (p=509)	R4
syupg	Syndrome update (p=509 path)	R4

4 Results

To provide technology-independent insight into the cost of our ISA extensions, we focus on ASIC results rather than FPGA-dependent evaluations. This allows a fair and consistent comparison with prior PQC accelerators and highlights the real overhead of integrating a programmable coprocessor into a modern embedded SoC. The architecture is synthesized using **Synopsys' Design Compiler 2020.09** in a **65nm CMOS technology**, at a **target clock frequency of 100 MHz**. This value is chosen as a convenient common operating point for comparison across designs, and does not correspond to the maximum achievable frequency of the implementation, which could be further increased with more aggressive timing closure. A clock uncertainty of 5% is assumed. Automatic optimization is enabled through the `compile_ultra` flag, while boundary optimization between blocks is disabled. Additionally, the `-gate_clock` and `-retime` options are applied to further improve timing and efficiency. Gate counts are reported in kGE, computed by normalizing the total cell area to the area of a 2-input NAND gate in the same technology library. Under this metric, the coprocessor accounts for 26.3 kGE, less than 2% of the total SoC area [38]. Instruction-level results in terms of performance and efficiency are detailed in Table 2. Here we report the clock cycles (C.C.) required by N tests of each instruction; the total ($\mathbf{P_{tot}}$), switching ($\mathbf{P_{sw}}$) and internal ($\mathbf{P_{int}}$) power contributions averaged across 100 simulations, the energy, both in absolute terms (**E**) and as a difference between the baseline and the custom instruction (δE), the energy saving, and the speedup with respect to a pure software execution over the baseline core. The s and c pedices for each column represent the software baseline and the custom instruction, respectively. Post-synthesis power estimates are generated using **Synopsys' Prime Power 2024.09**, based on the switching activities on the synthesized netlist from specifically triggered operations. The energy is obtained as $\mathbf{E} = (\mathbf{P_{tot}} \times \mathbf{C.C.} \times 1/freq)/N$, where the operating frequency is equal to 100 MHz and N is the number of tests (second column of Table 2). Code-size savings, measured with **riscv32-unknown-elf-size**, are modest: most kernels see only a 1–5% reduction, with larger gains (≈15–18%) for Keccak and some vector/matrix routines. Since latency and energy are our main targets, we report these results only qualitatively rather than in a dedicated table.

State-of-the-Art Comparison. As shown in Table 3, our design is among the smallest reported solutions, despite supporting the broadest set of algorithms. Coprocessors with lower gate counts target only a single scheme—typically ML-KEM or ML-DSA —whereas our ISA covers all NIST-standard algorithms (ML-KEM, ML-DSA, Falcon, SLH-DSA), all round-4 candidates (BIKE, HQC, Classic McEliece), and additional schemes such as NewHope, CROSS, and LESS. This level of coverage is achieved with an area footprint that remains comparable to, or smaller than, many single-purpose designs. The main reason is that the decode and committer stages, together with the CV-X-IF interface, are shared across all blocks, so their relative overhead shrinks as more instructions are added. In addition, we deliberately accelerate fine-grained computational bottlenecks using narrow, often multiplier-free datapaths that are reused by multiple instructions, aiming for a balanced area–performance trade-off rather than purely peak throughput. At one extreme, very compact cores of around 9–10 kGE [9, 11] focus on a single lattice or signature scheme, while, at the other extreme, large PQC subsystems integrated into high-performance SoCs reach 100 kGE [26, 28] to maximize throughput. Our coprocessor sits between these extremes: it remains an order of magnitude smaller than the largest designs while offering broader algorithm coverage and a richer

Table 2: Instruction-level characterization. Legend: s = software baseline (white columns), c = custom instruction (grey columns). Columns report cycles (C.C.), power (P_{tot}, P_{sw}, P_{int}), energy (E), energy difference (δE), and speedup.

test	N	C.C.$_s$ [cycles]	C.C.$_c$ [cycles]	P_{tot_s} [mW]	P_{tot_c} [mW]	P_{sw_s} [mW]	P_{sw_c} [mW]	P_{int_s} [mW]	P_{int_c} [mW]	E_s [nJ]	E_c [nJ]	δE [nJ]	Energy Saving × Speedup
keccak	1	24,787	9,061	12.3	12	1.277	1.305	10.4	9.998	3,048.8	1,087.3	1,961.5	×**2.80**
montgk	20	269	204	9.007	11	1.118	1.253	7.192	9.041	1.211	1.122	0.089	×**1.08**
montgn	20	328	204	9.007	11	1.118	1.251	7.192	9.040	1.477	1.122	0.355	×**1.32**
montgf	20	328	204	9.007	11	1.118	1.252	7.192	9.046	1.477	1.122	0.355	×**1.32**
montgt	20	270	204	9.007	11	1.118	1.26	7.192	9.06	1.216	1.122	0.094	×**1.08**
montgd	20	608	282	9.007	11.4	1.118	1.238	7.192	9.446	2.738	1.607	1.131	×**1.70**
nntk	1	19,772	14,270	11.5	11.4	1.238	1.205	9.513	9.535	2,273.8	1,626.8	647	×**1.40**
nntd	1	45,657	24,135	10.7	11	1.217	1.201	9.045	8.817	5,022.3	2,582.4	2,439.9	×**1.94**
inttk	1	30,413	18,241	11.2	11.7	1.233	1.252	9.221	9.729	3,406.3	2,134.2	1,272.1	×**1.59**
inttd	1	55,651	29,279	11	10.8	1.224	1.215	9.035	8.852	6,121.6	3,162.1	2,959.5	×**1.94**
cbd1	100	21,321	4,423	10.8	12.6	1.196	1.342	8.894	10.5	23.028	5.573	17.455	×**4.13**
cbd2	100	10,912	2,814	10.8	12.2	1.2	1.319	8.913	10.2	11.785	3.433	8.352	×**3.43**
cbd3	100	5,411	2,010	10.8	11.7	1.184	1.158	8.917	9.799	5.844	2.352	3.492	×**2.48**
cbd4	100	5,712	2,010	11	11.8	1.209	1.257	9.081	9.89	6.283	2.372	3.911	×**2.65**
rej12	100	1,627	907	9.007	11.4	1.118	1.240	7.192	9.426	1.465	1.034	0.431	×**1.42**
rej8	100	5,682	907	9.007	11.8	1.118	1.241	7.192	9.846	5.118	1.070	4.048	×**4.78**
compr1	100	1,415	1,004	9.007	11.3	1.118	1.221	7.192	9.383	1.275	1.135	0.139	×**1.12**
compr2	100	1,414	1,004	9.007	11.3	1.118	1.22	7.192	9.374	1.274	1.135	0.139	×**1.11**
compr3	100	3,013	1,004	9.007	11.3	1.118	1.221	7.192	9.394	2.714	1.135	1.579	×**2.39**
compr4	100	3,412	1,004	9.007	11.3	1.118	1.22	7.192	9.404	3.073	1.135	1.938	×**2.71**
`gfclmul`	100	27,611	1,301	10.9	11.9	1.217	1.189	8.964	10.1	30.096	1.548	28.548	×**19.4**
kar	1	2,805	13	10.8	10.8	1.198	1.169	8.948	8.947	302.94	1.404	301.54	×**216**
r2v	50	830	456	9.007	10.9	1.118	1.205	7.192	8.958	1.495	0.994	0.501	×**1.50**
vpw	50	908	652	9.007	11.7	1.118	1.201	7.192	9.828	1.636	1.526	0.110	×**1.07**
vmul	50	2,030	1,207	9.007	11.5	1.118	1.198	7.192	9.631	3.657	2.776	0.881	×**1.32**
rvmul	50	1,538	665	9.007	11.7	1.118	1.189	7.192	9.779	2.771	1.556	1.215	×**1.78**
syupd	50	1,508	567	9.007	11.5	1.118	1.204	7.192	9.62	2.717	1.304	1.413	×**2.08**
r2vg	50	5,310	605	9.007	11.3	1.118	1.235	7.192	9.407	9.565	1.367	8.198	×**7.00**
vpw	50	1,109	605	9.007	12	1.118	1.367	7.192	9.955	1.998	1.452	0.546	×**1.38**
vmulg	50	1,160	605	9.007	11.7	1.118	1.36	7.192	9.655	2.090	1.416	0.674	×**1.48**
rvmulg	50	6,312	605	9.007	12	1.118	1.358	7.192	9.637	11.370	1.416	9.954	×**8.03**
syupg	50	3,662	605	9.007	12	1.118	1.37	7.192	9.953	6.597	1.452	5.145	×**4.54**

instruction set than most of the sub-30 kGE solutions. Compared with the closest work, [15], our coprocessor operates at a similar frequency yet occupies less area when normalized by technology node, even though [15] already uses a 28 nm process while our design targets 65 nm. Moreover, while [15] reports only area and performance, our analysis includes post-synthesis power, per-instruction energy, and a detailed breakdown across Keccak, NTT/MR, sampling, and GF(2) arithmetic. This enables a significantly more complete evaluation of PQC workloads, showing that our ISA delivers competitive performance while remaining one of the most compact and fully featured coprocessor-based solutions in the state of the art.

Table 3: Comparison of ASIC PQC accelerators.

Ref.	Tech. [nm]	f [MHz]	Gate Counts [kGE]	Supported Algorithm
[7]	40	72	106	ML-KEM, NewHope
[8]	65	45	51	ML-KEM, NewHope
[9]	28	500	9.3	ML-KEM
[11]	28	400	9.3	ML-DSA
[12]	28	200	115.2	SLH-DSA
[13]	22	100	-	ML-DSA
[15]	28	150	54	ML-KEM, ML-DSA, SLH-DSA, HQC, Falcon, BIKE, Classic McEliece
[26]	65	280	769	ML-KEM, ML-DSA
[28]	28	300	942	ML-KEM
[29]	28	200	119.4	ML-KEM, ML-DSA
Ours	**65**	**100**	**26.3**	ML-KEM, ML-DSA, SLH-DSA, HQC, SLH-DSA, Falcon NewHope, CROSS, LESS

5 Conclusions

This novel PQC-ISE delivers a lightweight and energy-efficient acceleration framework for both standardized and under-evaluation post-quantum schemes, with an area of only 26.3 kGE and substantial energy reductions over the software baseline. Across all kernels, the proposed instructions achieve an average speedup of roughly 5× compared to pure software execution, reaching more than two orders of magnitude (over 200×) for the specific case of Karatsuba multiplication. Leveraging the CV-X-IF, the proposed co-processor can be seamlessly integrated into a wide range of RISC-V cores. Its modular design enables straightforward extensibility, allowing new cryptographic kernels to be incorporated with minimal hardware changes, thus demonstrating the inherent crypto-agility of the proposed design.

Acknowledgments

This work is supported in parts by the EU project titled RIGOLETTO and from the EU's Horizon Europe research and innovation programme under grant agreement No. 101225722 (FORTRESS). The RIGOLETTO project is funded by the Chips Joint Undertaking (Chips JU) under Grant Agreement No. 101194371.

References

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 212–219, 1996.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[3] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*. Springer, 2009.

[4] National Institute of Standards and Technology, "Post-quantum cryptography: Nist's plan for the future of encryption," https://csrc.nist.gov/projects/post-quantum-cryptography, 2016.

[5] ——, "Post-Quantum Cryptography Standardization," https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2016, accessed: 17 November 2025.

[6] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic - accelerating kyber and NewHope on RISC-v," Cryptology ePrint Archive, Paper 2020/049, 2020. [Online]. Available: https://eprint.iacr.org/2020/049

[7] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 4, p. 17–61, Aug. 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8344

[8] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 4, p. 239–280, Aug. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8683

[9] C. Gewehr, L. Luza, and F. G. Moraes, "Hardware Acceleration of Crystals-Kyber in Low-Complexity Embedded Systems With RISC-V Instruction Set Extensions," *IEEE Access*, vol. 12, pp. 94 477–94 495, 2024.

[10] T. Wang, C. Zhang, X. Zhang, D. Gu, and P. Cao, "Optimized Hardware-Software Co-Design for Kyber and Dilithium on RISC-V SoC FPGA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, p. 99–135, Jul. 2024. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11671

[11] K. Miteloudi, J. Bos, O. Bronchain, B. Fay, and J. Renes, "PQ.V.ALU.E: Post-Quantum RISC-V Custom ALU Extensions on Dilithium and Kyber," Cryptology ePrint Archive, Paper 2023/1505, 2023. [Online]. Available: https://eprint.iacr.org/2023/1505

[12] Z. Ye, X. Li, C. Wang, R. C. C. Cheung, and K. Huang, "RVSLH: Acceleration of Postquantum Standard SLH-DSA With Customized RISC-V Processor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–5, 2025.

[13] T. Stelzer, F. Oberhansl, J. Schupp, and P. Karl, "Enabling Lattice-Based Post-Quantum Cryptography on the OpenTitan Platform," in *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 51–60. [Online]. Available: https://doi.org/10.1145/3605769.3623993

[14] A. Dolmeta, S. Di Matteo, E. Valea, M. Carmona, A. Loiseau, M. Martina, and G. Masera, "TYRCA: A RISC-V Tightly-Coupled Accelerator for Code-Based Cryptography," in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–7.

[15] J. Lee, W. Kim, and J.-H. Kim, "A Programmable Crypto-Processor for National Institute of Standards and Technology Post-Quantum Cryptography Standardization Based on the RISC-V Architecture," *Sensors*, vol. 23, no. 23, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/23/9408

[16] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) Standard (FIPS 203)," NIST, FIPS 203, 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[17] ——, "Module-Lattice-Based Digital Signature Algorithm (ML-DSA) Standard (FIPS 204)," NIST, FIPS 204, 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.204

[18] ——, "Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) Standard (FIPS 205)," NIST, FIPS 205, 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.205

[19] ——, "NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption (NIST IR 8545)," NIST, Internal Report IR 8545, 2025. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2025/NIST.IR.8545.pdf

[20] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over NTRU," Submission to the NIST post-quantum cryptography standardization process, 2018. [Online]. Available: https://falcon-sign.info/falcon.pdf

[21] M. Baldi, S. Bitzer, A. Pavoni, P. Santini, A. Wachter-Zeh, and V. Weger, "Zero knowledge protocols and signatures from the restricted syndrome decoding problem," Cryptology ePrint Archive, Paper 2023/385, 2023. [Online]. Available: https://eprint.iacr.org/2023/385

[22] J.-F. Biasse, G. Micheli, E. Persichetti, and P. Santini, "Less is more: Code-based signatures without syndromes," in *Progress in Cryptology - AFRICACRYPT 2020*, A. Nitaj and A. Youssef, Eds. Cham: Springer International Publishing, 2020, pp. 45–65.

[23] National Institute of Standards and Technology (U.S.), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," National Institute of Standards and Technology, NIST FIPS 202, 2015. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.202

[24] A. Karmakar, "Design and implementation aspects of post-quantum cryptography," Ph.D. dissertation, KU Leuven, Leuven, Belgium, 2020, phD thesis. [Online]. Available: https://lirias.kuleuven.be/2957665

[25] H. Nejatollahi, N. D. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Computing Surveys*, vol. 51, no. 6, pp. 129:1–129:41, 2019. [Online]. Available: https://doi.org/10.1145/3292548

[26] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A Crystal for Post-Quantum Security Using Kyber and Dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 2, pp. 747–758, 2023.

[27] L. Li, G. Qin, Y. Yu, and W. Wang, "Compact Instruction Set Extensions for Kyber," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 3, pp. 756–760, 2024.

[28] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2672–2684, 2020.

[29] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A Highly-efficient Lattice-based Post-Quantum Cryptography Processor for IoT Applications," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, p. 130–153, Mar. 2024. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11423

[30] OpenHW Group, *Core-V eXtension Interface (CV-X-IF) Specification*, OpenHW Group, 2024, version f08d951d (rev. latest). [Online]. Available: https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/en/latest/intro.html

[31] C.-A. Popovici, A. Stan, N.-A. Botezatu, and V.-I. Manta, "RiscADA: RISC-V Extension for Optimized Control of External D/A and A/D Converters," *Electronics*, vol. 14, no. 15, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/15/3152

[32] A. Dolmeta, M. Martina, and G. Masera, "Exploring the New CV-X-IF Interface to Customize RISC-V Instruction Sets: A Case of Study in Cryptography," in *Applications in Electronics Pervading Industry, Environment and Society*, M. Ruo Roch, F. Bellotti, R. Berta, M. Martina, and P. Motto Ros, Eds. Cham: Springer Nature Switzerland, 2025, pp. 39–47.

[33] K. Hepola, T. R. Arachchige, J. Multanen, and P. Jääskeläinen, "Fully Automatic Compiler Retargeting and CV-X-IF Hardware Interface Generation for RISC-V Custom Instructions," in *2024 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2024, pp. 1–7.

[34] L. Waucquez and A. Rodriguez, "EROS: Extensible Reliable Offloading Solution," in *Proceedings of the 22nd ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, ser. CF '25 Companion. New York, NY, USA: Association for Computing Machinery, 2025, p. 82–85. [Online]. Available: https://doi.org/10.1145/3706594.3726982

[35] J. A. Solinas, "Generalized Mersenne numbers," National Security Agency, Communications Security Establishment, Tech. Rep. CORR 99-39, 1999. [Online]. Available: https://cacr.uwaterloo.ca/techreports/1999/corr99-39.pdf

[36] L. Ducas and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Advances in Cryptology – CRYPTO 2013*, ser. LNCS, vol. 8042. Springer, 2013, pp. 40–56.

[37] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) Standard," U.S. Department of Commerce, Tech. Rep. FIPS 203, 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[38] P. D. Schiavone, S. Machetti, M. Peón-Quirós, J. Miranda, B. Denkinger, T. C. Müller, R. Rodríguez, S. Nasturzio, and D. A. Alonso, "X-heep: An open-source, configurable and extendible risc-v microcontroller," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 379–380. [Online]. Available: https://doi.org/10.1145/3587135.3591431

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009