NIST Post-Quantum Cryptography- A Hardware Evaluation Study

Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri

***Abstract*—Experts forecast that quantum computers can break classical cryptographic algorithms. Scientists are developing post-quantum cryptographic (PQC) algorithms, that are invulnerable to quantum computer attacks. The National Institute of Standards and Technology (NIST) started a public evaluation process to standardize quantum-resistant public key algorithms. The objective of our study is to provide a hardware-based comparison of the NIST PQC candidates. For this, we use a High-Level Synthesis (HLS)-based hardware design methodology to map high-level C specifications of round 2 PQC candidates into both FPGA and ASIC implementations.**

I. Introduction

Public key cryptography is a fundamental security protocol for all forms of digital communication, wired or wireless. Public key cryptography has three main cryptographic functions, namely (a) public key encryption, (b) digital signatures, and (c) key exchange [1]. RSA and Elliptic Curve-based public key cryptography algorithms provide security guarantees based on the difficulty of solving the integer factorization and discrete logarithm problems. Peter Shor from Bell Labs showed that quantum computers can factorize integers in polynomial time rendering traditional public key cryptography algorithms ineffective [2]. Hence, researchers are investigating robust alternatives such as lattice and code cryptography algorithms.

We conduct a hardware assessment of post-quantum cryptographic (PQC) algorithms that were submitted to the NIST evaluation. As a historical perspective, in 1997 NIST sought guidance from the public to identify a replacement for the Data Encryption Standard (DES), Advanced Encryption Standard (AES) [3]. Since then, open cryptographic assessments have become the way to selecting cryptographic standards. For example, NESSIE (2000-2002), eSTREAM (2004-2008), CRYPTREC (2000-2002), SHA-3 (2007-2012) and CAESAR (2013-2019) embraced this approach. In these assessments, security was the principal yardstick. Performance in software, performance in application specific integrated circuits (ASIC), performance in FPGA, and feasibility of implementation using limited resources (small microprocessors and low-power hardware) are secondary criteria. In the AES competition, Rijndael had the fastest ASIC and the second fastest FPGA implementation with identical security guarantees as its competitors [4].

We report a hardware-implementation comparison of NIST round 2 PQC candidates. The contributions of this study are:

1) Developed systematic FPGA and ASIC design flows for PQC evaluation starting from a C specification.
2) Studied performance vs area trade-offs for 11 PQC algorithms, including lattice, code, hash, and multivariate based KEM and Signature algorithms.
3) Improved the latency of PQC implementations using optimizations such as loop unrolling and loop pipelining.
4) Performed a detailed study of three signature algorithms to explore area vs performance vs security trade-offs.

The paper is organized as follows. Section II gives a background on Post-Quantum Cryptography. Section III describes the design flow and Section IV presents experimental results. Section V describes case studies using three signature-based algorithms and Section VI enumerates the key takeaways.

K. Basu, D. Soni and R. Karri are with the NYU center for cybersecurity (http://cyber.nyu.edu), New York University, New York, NY, USA. e-mail: (kb150@nyu.edu, dss545@nyu.edu, rkarri@nyu.edu).

M. Nabeel is with NYU, Abu Dhabi, UAE. email:(mohammed.nabeel@nyu.edu).

II. Post-Quantum Cryptography

Researchers are developing cryptographic algorithms to resist attacks by classical and quantum computers. The major classes of post quantum cryptography algorithms are:

- **Lattice-based cryptography** algorithms offer the best performance, but are the least conservative among all [5]. Lattice cryptography builds on the hardness of the shortest vector problem (SVP) which entails approximating the minimal Euclidian length of a lattice vector. Even with a quantum computer SVP is shown to be polynomial in n [1]. Other lattice cryptography algorithms are based on Short Integer Solutions (SIS). These are secure in the average case if the SVP is hard in the worst-case [6].
- **Code-based cryptography** uses error correcting codes and offers a conservative approach for public key encryption/key encapsulation, as it is based on a well-studied problem for over 4 decades [7]. This class of algorithms use large keys and some attempts at reducing their key size have made these algorithms vulnerable to attacks [8]. Researchers proposed techniques to reduce the key size without compromising the security [9], [10].
- **Multivariate polynomial cryptography** relies on the difficulty of solving the multivariate polynomial (MVP) algorithm over finite fields. Solutions of MVP problems are NP-hard over any field and NP-complete even if all equations are quadratic over $GF(2)$ [11]. MVPs are preferred as signature schemes as they offer the shortest signatures. However, some schemes are broken [12].
- **Hash-based digital signatures** resist quantum-computer attacks. These schemes are based on the security properties of the underlying cryptographic hash functions (collision resistance and second pre-image resistance).

- **Other cryptographic methods** include evaluating isogenies on super singular elliptic curves. Shor's algorithm is ineffective against evaluating isogenies [13].

PQC	The hard problem	Sign	KEM	Total
Lattice	Find shortest vector, closest vector	5 (3)	23 (9)	28 (12)
Code	Decode random linear code	3 (0)	17 (7)	20 (7)
Multivariate	Solve multivariate quadratic equations	8 (4)	2 (0)	10 (4)
Hash	Second pre-image resistance of hash function	3 (2)	0 (0)	3 (2)
Isogeny	Find isogeny map btwn elliptic curves with same number of points	0 (0)	1 (1)	1 (1)
Other	-	2 (0)	5 (0)	7 (0)
Total	-	**21 (9)**	**48 (17)**	**69 (26)**

TABLE I: A summary of PQC digital signature and key encapsulation submissions and the underlying hard mathematical problems. We show the number of algorithms for Round 1 and Round 2 (within braces) NIST PQC evaluation.

A. Digital signatures and key encapsulation

The ongoing NIST PQC standardization process is consolidating the invulnerable candidates after each successive round. Each candidate in the NIST PQC contest realizes one of three functions: public key encryption, digital signature, and key encapsulation mechanism (KEM). Table I shows the number of PQC round 2 submissions and summarizes their functionality and mathematical complexity.

NIST expects to standardize more than one algorithm in order to provide different trade-offs depending on the application (speed vs memory vs power consumption etc.). A direct comparison of the security guarantees provided by the candidate PQC algorithms is challenging absent a standard quantum computing platform, differences in the mathematical functions that underlie the algorithms, and the sophisticated algorithm specifications compared to prior cryptographic contests.

1) Digital signatures: The PQC digital signatures work on the principle that the sender signs the message with a private key and the receiver verifies the signature using the sender's public key. These algorithms use three functions.

1) **crypto_sign_keypair** generates the public key pk and the secret key sk.
2) **crypto_sign** takes in sk and the message m plus its length $mlen$ and outputs the signature sm appended to the message.
3) **crypto_sign_open** takes in pk, sm and length $smlen$, and outputs message m.

Popular PQC digital signature algorithms that we considered are: qTESLA, MQDSS and Crystals-Dilithium.

2) Key Encapsulation: Traditional encryption-decryption protocols encrypt a message using the public key of the sender, which is then decrypted by the receiver using his private key. Classic public-key cryptographic algorithms based on ECC and RSA work on this principle. However, they have been shown to be vulnerable to quantum attacks [2]. One way to counter this problem is to encrypt and decrypt a message m using a symmetric key M. The symmetric key is encrypted by the sender and sent to the receiver along with the encrypted message. The receiver decrypts and recovers the symmetric key M first and then decrypts the message m using M. The first challenge with this approach is that an attacker can reconstruct a small M. Hence, the sender needs to make M large. Second, if the attacker derives M, she can recover m. KEM schemes counter these two challenges. In KEM, the sender first generates a random number rn and then generates a symmetric key $M = KDF(rn)$ using the Key Derivation Function (KDF). A cryptographic hash is an example KDF. KEM obviates padding and this way reduces the key size. Since KDF is one-way, the attacker can not generate rn, even if she recovers M. KEMs use three functions:

1) **crypto_kem_keypair** is used to generate the public and private keys pk and sk.
2) **crypto_kem_enc** takes in public key pk and outputs key k and encapsulated key ck.
3) **crypto_kem_dec** takes in secret key sk and encapsulated key ck and outputs key k.

Initially, we consider six KEM PQC algorithms: Newhope, Frodokem, Crystals-KYBER, NTRU-HRSS, Classic McEliece, and Saber[1].

B. Security classification of PQC algorithms

NIST specified five security strength categories. **Security level 1** $\implies$ equivalent to AES-128 key search. **Security level 2** $\implies$ equivalent to SHA-256/SHA3-256 collision search. **Security level 3** $\implies$ AES-192 key search. **Security level 4** $\implies$ SHA-384/SHA3-384 collision search. **Security level 5** $\implies$ AES-256 key search. The security strengths of the NIST round 2 PQC algorithms are in Table II.

C. Hardware Implementations

There has been sporadic hardware implementations for PQC algorithms. [14], [15] provide a detailed survey on hardware implementations of various PQC algorithms. For example, Stratix V FPGA implementation of Classic McEliece was presented in [16]. The PQC algorithm Saber was implemented on an ARM Cortex-M micro-controllers in [17] (they used a Cortex-M4 micro-controller to implement a speed-optimized version of Saber and a Cortex-M0 micro-controller to implement an area-optimized version). The Signature algorithm SPHINCS+ was implemented on an ARM CORTEX-M3 processor [18]. Existing cryptographic co-processors were repurposed for KYBER KEM in [19]. Design-space exploration of hardware accelerators for NewHope and BLISS-BI (which is not selected in NIST PQC Round 2) have been performed by [20]. The other algorithms (see Table II) do not have hardware implementations. **This is the first hardware benchmarking and uses a common evaluation framework to study area vs performance vs security trade-offs.**

[1]Why these algorithms?: We started our evaluation with round 1 PQC algorithms based on an informal survey of a few PQC investigators and their comments informed this shortlist. We considered KEM and Signature algorithms to have a heterogeneous case study. Our initial study had 14 algorithms. After Round 2 winners were declared, we pared down four more.

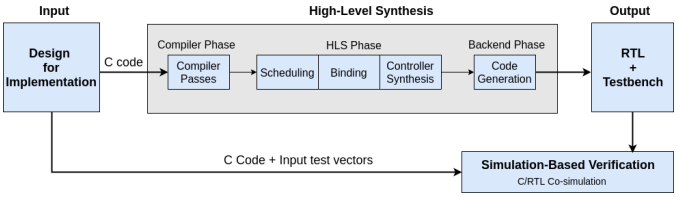


Fig. 1: High-Level Synthesis (HLS) design flow.

III. C-based Synthesis of (PQC) Accelerators

Dedicated PQC accelerators have vital practical applications. These accelerators can be either low-area and low-power crypto cores used in small IoT devices or low-latency implementations used in servers [16]. As described in Section II-C, there are no hardware implementations of PQC algorithms. This paper is an early study of hardware implementations of 11 PQCs employing a uniform hardware implementation flow.

We use HLS-based design space exploration. HLS starts from a software specification (e.g., C, C++, and SystemC) and generates a Register Transfer Level (RTL) description (in Verilog or VHDL) ready for the backend design flow (i.e., logic synthesis and physical design). HLS-based design space exploration is popular for two reasons:

1) **Easy verification:** A high-level testbench written in C or C++ can be re-used to verify/validate the RTL. There is no need to synthesize verilog/VHDL testbenches.
2) **Extensive design-space exploration:** The C/C++ code can be re-used to explore multiple design points guided by the constraints such as area and latency.

Leading electronic design automation vendors like Mentor Graphics, and Cadence, and FPGA vendors like Xilinx [21] and Intel [22] offer HLS tools and designs flows. HLS organizes components hierarchically according to the sub-function organization in the C specification to manage design complexity. Each hardware unit adopts the classical Finite State Machine with Data path (FSMD) model [23] with two components: *controller* and *data path*. The controller orchestrates the operations in each clock cycle. The finite state machine (FSM) represents the control flow signals the data path resources based on a set of conditions. The data path includes the functional units and registers to hold temporary values. Multiplexers drive the values based on the control flow.

HLS-based accelerator design has three parts, as shown in Figure 1: The compiler phase investigates the input C description and applies compiler-level transformations. The HLS phase determines the microarchitecture. The back end generates the Verilog/VHDL RTL and test benches. RTL simulation is performed on a set of pre-defined inputs to determine if the results match the golden ones obtained from software execution.

A. C compiler optimizations

A typical HLS flow uses GCC or LLVM compilers to parse the input C code, apply compiler optimizations, and generate an intermediate representation (IR). HLS uses the Single Static Assignment form so that the IR can be easily manipulated and turned into RTL hardware by the HLS steps. One can apply optimizations such as loop unrolling, constant propagation, and function inlining on the IR to permit downstream HLS optimizations. The call graph representation of the algorithms describes the relationships between the functions in the algorithms and aids the HLS determine the components and the hierarchical interconnections between them.

B. HLS

Each function in the IR is transformed into an RTL hardware module. During HLS *allocation* step, resources are selected. The HLS *scheduling* step determines the operations to be executed in each clock cycle and this determines the latency of the circuit. The HLS scheduling step generates the finite-state machine (FSM) controller, which implements the control-flow management of the accelerator. Operations scheduled in different clock cycles reuse the same resources. In the HLS *binding* step, scheduled operations are bound to functional units and temporary values crossing the clock boundaries are stored in registers. Next, the functional units and registers are interconnected using multiplexers. The controller synthesis step creates the controller from the FSM. Based on the operations to execute on the microarchitecture, the controller generates signals that route the data in the accelerator data path through the multiplexers in each clock cycle. We use Xilinx Vivado HLS as it supports C design [21]. The HLS hardware design flow can also use Mentor Catapult HLS [24], LegUp HLS [25], and Intel HLS [22].

C. Backend

The RTL Verilog/VHDL description is generated, together with the library components (e.g., custom operators or memory interfaces) used in the design as outputs of HLS. HLS produces the hardware test bench or an interface for co-simulation with the software test bench. A pre-defined set of inputs can be used to generate the golden output values, which are then matched with the simulation results. We used an FPGA back-end flow (Xilinx FPGA synthesis tools) [21] and the ASIC back-end flow (Synopsys Design Compiler [26]).

IV. PQC Hardware Assessment

A. HLS-based Assessment Methodology

Figure 2 shows the HLS design exploration flow for PQC algorithms. We modify the original C specification to make it HLS-suitable (e.g., change pointers to fixed dimension arrays and remove recursions). Next, we perform HLS on the synthesizable C code to generate RTL using Xilinx Vivado HLS. Vivado provides a detailed synthesis report identifying which modules/loops in the design are the cause of the longest latency. If there are loops, we optimize them using loop unrolling and pipelining[2].

[2]Since there are numerous tables and graphs in this study, the position of the tables may not be close to the text that discusses them. Hence, we made their captions self-contained.

Algorithm	Basis hard problem	PQC Primitive	Supported Security Level (public key size in bytes)				
			1	2	3	4	5
Classic McEliece [27]	Code	KEM	-	-	-	-	**1047319** 1357824
LEDAcrypt [28]	Code	KEM	**27779**	57557	57557	99053	99053
Saber [29]	Lattice	KEM	672	-	**992**	-	1312
Crystals-KYBER[30]	Lattice	KEM	**736**	-	1088	-	1440
NewHope [31]	Lattice	KEM	**928**	-	-	-	1824
FrodoKEM [32]	Lattice	KEM	**9616**	-	15632	-	-
NTRU-HRSS [33]	Lattice	KEM	**9100**	-	-	-	-
Crystals-Dilithim [34]	Lattice	Signature	**1184**	1472	1760	-	-
SPHINCS+ [35]	Hash	Signature	**32**	-	48	-	64
MQDSS [36]	Multivariate	Signature	-	**62**	-	88	-
qTESLA [37]	Lattice	Signature	**1504** 14880	-	3104 2976 39712	-	-

TABLE II: **PQC algorithms used in this study:** A high-level analysis of 11 NIST PQC implementations. Of these, seven are KEM primitives and four are signature primitives. Of the KEM primitives, two are code and the remaining six are Lattice. Two of the PQC signature primitives are lattice, one is hash and one is multivariate . Algorithms not supporting a particular security level are indicated with a '-'. Classic McEliece has two implementations, both of which are of security level 5. The security level of each algorithm used in this paper are shown in bold. Among these, eight are at security level 1, one is ar security level 2, one is at security level 3 and one is at the highest security level 5. Algorithms across various security levels are chosen for two purposes. First, many algorithms do not support some security levels. Furthermore, we wanted to perform a heterogeneous case study across various security levels.

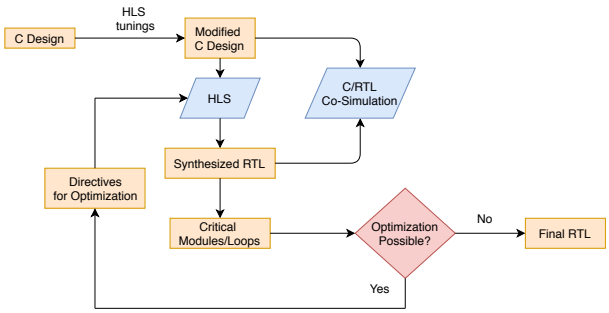


Fig. 2: HLS-based design exploration of PQC algorithms.

B. PQC Algorithms in this Study

We evaluated eleven NIST PQC implementations. The PQC algorithms and their implementation and security characteristics are summarized in Table II. In this paper, we focused on PQC encapsulation/signature and decapsulation/verification components of these algorithms. Synthesis of keypair generation is left as a future exercise. We used Xilinx Virtex-7 FPGA as target device for our synthesis[3]

C. Performance metrics

The considered performance indicators are: latency, area and latency-area product. Latency is the time required by system to produce the output from the time the input is provided. Throughput is the maximum speed at which the outputs can be provided. The minimum number of clock cycles between two successive inputs is the initiation interval (II) and is a measure of throughput. A lower II indicates higher throughput. Figure 3 shows a system with 3 modules. The latency of the system is 10 clock cycles as the whole computation should be completed in each module one after the other. The most time consuming module 1 consumes 5 clock cycles and is the bottleneck. After 5 clock cycles, the next input is given to module A. Therefore, the system II is 5 clock cycles. In this paper, latency and II are used interchangeably. Therefore, a design with lower latency indicates better throughput.

We will use latency to measure the performance of the designs. We will use Flip-Flops and LUTs for FPGA implementations and chip area for ASIC implementationsThere is a trade-off between speed and area. Reduction in latency increases the total area. Hence, the Latency-area product (LAP) is used to check the efficiency and resource utilization. A lower LAP corresponds to a superior implementation.

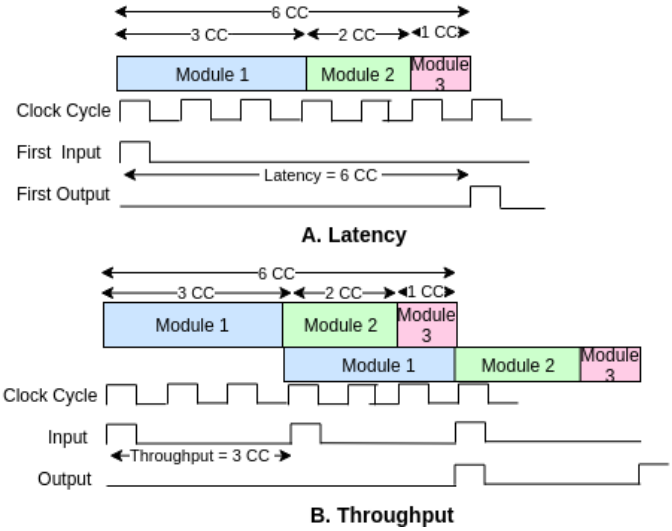


Fig. 3: Illustration of Latency and Throughput metrics.

[3]We started this study before NIST announced Artix-7 as the target platform. Our experiments in Section V use Artix-7 board.

Algorithm	Sec. Level	FF	LUT	Clock (nsec)	Latency
KEM algorithms					
CRYSTALS Kyber	1	40720	230540	15	56345
Newhope	1	26257	135689	15	681191
FrodoKEM	1	14516	82265	10	469217
NTRU-HRSS	1	6633	33845	15	1496914
LEDACrypt	1	8774	90308	10	37832
Saber	3	38495	214764	15	499812
Classic McEliece	5	60264	840384	10	5128978
Signature algorithms					
CRYSTALS Dilithium	1	25926	133461	10	609828
SPHINCS+	1	8641	31147	10	628778326
qTELSA	1	41978	232582	10	125374
MQDSS	2	35263	193320	15	49365597

TABLE III: **Description:** Security versus area versus the timing of PQC encapsulation algorithms, without optimizations (i.e., baseline). **Analysis:** Among the security level 1 KEM algorithms, NTRU-HRSS has a latency of over 1 million cycles and NTS-KEM is the fastest. LIMA has low-latency and the strongest security level 5. Among the security level 1 signature algorithms, CRYSTALS-Dilithium and qTESLA has latencies fewer than a million cycles. **Takeaway:** CRYSTALS-Dilithium (for signature generation), and FrodoKEM (for KEM) are good candidates for IoT devices. CRYSTALS-Dilithium signature generation algorithm has security level 1 and is the second fastest and the second smallest among the Signature algorithms.

D. Baseline hardware implementations

Tables III and IV report the hardware and timing overhead for implementing the PQC encapsulation, decapsulation and keypair generation algorithms, respectively when synthesized without any additional constraints (latency). Figure 4 shows how the KEM and Signature encapsulation algorithms can be ordered when ranked on the basis of least latency.

E. Critical Functions

In this section, we will examine the critical functions and loops in them that result in high latency for the PQC KEMs. The results are shown in Table V.

F. Performance Optimizations

We use loop unrolling for encapsulation/signature and decapsulation/verification and report the results in Table VI and Table VII respectively. The ranking of the encapsulation/signature functions after loop unrolling is shown in Figure 5(a). NTRU-HRSS is the fastest KEM and Crystals-Dilithium is the fastest Signature.

The results for encapsulation/signature, using loop pipelining, are presented in Table VIII. The ranking of the encryption functions after pipelining is shown in Figure 5(b). Among the algorithms with security level 1, CRYSTALS-Kyber is the fastest KEM and Crystals-Dilithium is the fastest Signature.

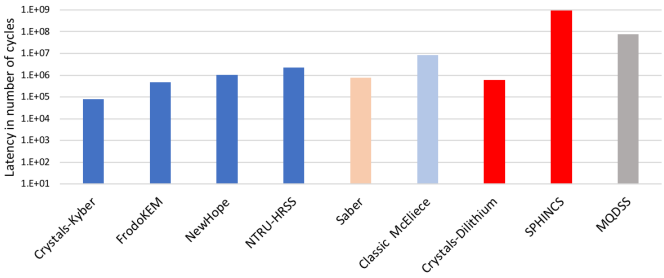


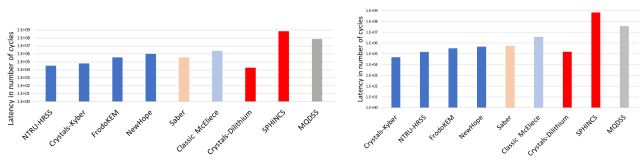
Fig. 4: We sorted the PQC encapsulation algorithms by latency, normalized for a clock cycle of 10 ns for each security level. The KEM algorithms are in blue, orange and sky blue, corresponding to security levels 1, 3, and 5. The Signature algorithms are in red and gray, corresponding to security levels 1 and 2. Among the KEM algorithms, two have latency less than 100,000 cycles and two more have latency less than a million cycles. Two KEM algorithms have latency more than a million cycles. Among the signature algorithms, one security level 1 has a latency less than a million cycles and one algorithm of security level 1 has more than a million cycles latency. The one security level 2 signature algorithm has more than a million cycles latency.

G. Latency-area product Comparisons

In this section, we compare the latency-area product (LAP) for the encapsulation algorithms for both the baseline and

Algorithm	Sec. Level	FF	LUT	Clock (nsec)	Latency
KEM- algorithms					
CRYSTALS Kyber	1	33030	186244	15	53553
Newhope	1	19635	92250	15	723027
FrodoKEM	1	14461	82307	10	220344
NTRU-HRSS	1	5292	29532	15	1003222
LEDACrypt	1	1891	5631	10	45964
Saber	3	33751	189597	15	89392
Classic McEliece	5	70112	847949	10	146126996
Signature algorithms					
CRYSTALS-Dilithium	1	20865	108878	10	5380
SPHINCS+	1	3335	11438	10	937975
qTELSA	1	29875	168570	10	71223
MQDSS	2	26423	147359	15	25124450

TABLE IV: **Description:** Security versus area versus the timing of PQC decapsulation algorithms, without optimizations (i.e., baseline). **Analysis:** Among the KEM algorithms, NTRU-HRSS (security level 1) and Classic McEliece (security level 5) have latency of more than 1 million cycles. Among the Signature algorithms, qTESLA and CRYSTALS-Dilithium have a latency less than a million cycles. CRYSTALS-Dilithium is the fastest among signature algorithms. **Takeaway:** SPHINCS+ (for Signature generation) and LEDACRYPT (for KEM) are good candidates for IoT devices. None of the level 5 security decapsulations in this study have low latency and hence are not appropriate for servers. All the low latency algorithms have level 1 security – LEDACRYPT (KEM) and CRYSTALS-Dilithium (Signature).



(a) (b)

Fig. 5: **Description:** PQC encapsulation algorithms sorted by latency, normalized for the clock cycle of 10 ns, when (a) loop unrolling (b) loop pipelining are performed. The KEM algorithms are in blue, orange, and sky blue, corresponding to security levels 1, 3, and 5. The signature algorithms are in red and gray, corresponding to security levels 1 and 2. **Analysis:** (a) Loop unrolling: Five KEM algorithms have latency fewer than 100,000 cycles – three have security level 1 and one has security level 5. Three KEM algorithms have latency less than a million cycles (one of level 1 and two of level 3) and two KEM algorithms have latency over a million cycles (one of level 1 and the other of level 5). Two signature algorithms have latency fewer than a million cycles (both are of level 1) and two have latency higher than a million cycles (one each of levels 1 and 2). (b) Loop pipelining: Four KEMs have latency less than 100,000 cycles (three of level 1, one of level 3 and one of level 5). Four KEMs have latency less than a million cycles (three of level 1 and one of level 3). Only one security level 5 KEM has latency more than a million cycles. For signature algorithms, two have latency fewer than a million cycles (both of level 1) and two higher than a million cycles (one of level 1 and the other of level 2.)

the optimization techniques. Similar to [38], we consider the area as the number of FPGA LUTs required to synthesize the design. The results are shown in Table IX. A lower LAP corresponds to better performance in terms of latency. From Table IX, the number of algorithms for which the lowest LAP is obtained using unrolling is about similar to that of pipelining. This is because of the many loop dependencies in some algorithms like CRYSTALS-Dilithium, which restricts the speedup due to pipelining. The PQC algorithms are ranked in terms of LAP in Figure 6.

Algorithm	Critical Functions	# Loops
KEM algorithms		
NTRU-HRSS	`poly_Rq_mul`	2
Saber	`vectormul`	2
CRYSTALS-Kyber	`gen_matrix`	3
Newhope	`poly_uniform`	2
FrodoKEM	`frodo_sample_n, frodo_mul_add`	1,1
Classic McEliece	`syndrome`	1
LEDACrypt	`KeccakF1600_StatePermute`	3
Signature algorithms		
CRYSTALS-Dilithium	`expand_mat`	2
MQDSS	`crypto_sign`	2
qTESLA	`sparse_mul16`	1
SPHINCS+	`treehash`	5

TABLE V: Critical functions of the PQC KEMs.

Algorithm	Sec. Level	FF	LUT	Clock (nsec)	Latency
KEM algorithms					
CRYSTALS Kyber	1	237182	2414748	15	42823
Newhope	1	26257	135689	15	680150
FrodoKEM	1	44284	136998	10	366609
NTRU-HRSS	1	9035	65356	15	22594
LEDACrypt	1	288530	783651	10	6660
Saber	3	93234	376313	15	236812
Classic McEliece	5	69795	934492	10	2373772
Signature algorithms					
CRYSTALS Dilithium	1	158313	584742	10	18525
SPHINCS+	1	8931	42604	10	464961626
qTESLA	1	97235	328106	10	59854
MQDSS	2	45135	230273	15	49365597

TABLE VI: **Description:** Security versus area versus the timing of PQC encapsulation algorithms, after loop unrolling. **Analysis:** Compared to Table III, virtually all algorithms have an improvement in latency, with an ensuing increase in area. CRYSTALS-KYBER incurs the most area among KEM algorithms and CRYSTALS-Dilithium among the Signature algorithms. Among Signature PQC algorithms, loop unrolling doesn't reduce the latency of MQDSS. NTRU-HRSS has the most speedup in terms of latency 66× reduction in latency compared to baseline (in Table III). **Takeaway:** Loop unrolling reduces the latency of all PQC encapsulation algorithms. However, it also results in an increase in area. Among the other high security algorithms, MQDSS (Signature, security level 2) can be used for IoT devices, since it provides relatively low hardware overhead. Among the lower security (level 1) algorithms, NTRU-HRSS (KEM) and SPHINCS+ (Signature) can be used for IoT devices, since the area overhead is low.

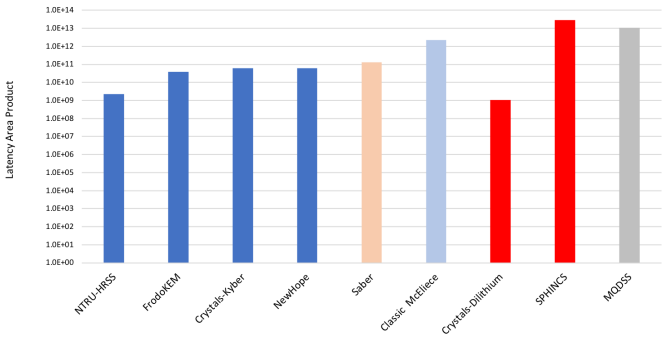


Fig. 6: PQC KEMs sorted by LAP, normalized for clock cycle of 10ns. The KEMs are in blue, orange and sky blue, corresponding to security levels 1, 3, 5. Signature algorithms are red and gray, corresponding to security levels 1 and 2. Among the KEMs, one has a LAP of less than 10^{10}, of security level 1. Four are in the range $10^{10} - -10^{12}$, three of security level 1 and one of security level 3. One security level 5 algorithm has a LAP of over 10^{12}. Among signatures, one has LAP $< 10^{10}$ – of security level 1. Two – one security level 1 and another of security level 2 have LAP $> 10^{12}$.

H. Security level vs hardware tradeoffs

The PQC algorithms have different implementations depending on the security strength (i.e. the implementations

Algorithm	Sec. Level	FF	LUT	Clock (nsec)	Latency
KEM algorithms					
CRYSTALS Kyber	1	194126	1977896	15	43018
Newhope	1	28999	164937	15	721986
FrodoKEM	1	97355	128031	10	117736
NTRU-HRSS	1	11514	97791	15	21996
LEDACrypt	1	164230	406135	10	18079
Saber	3	231549	2350000	15	365015
Classic McEliece	5	79962	870908	10	10659024
Signature algorithms					
CRYSTALS Dilithium	1	108154	388991	10	5380
SPHINCS+	1	3335	11438	10	937975
qTESLA	1	77567	247726	10	36423
MQDSS	2	93945	323734	15	25084906

TABLE VII: **Description:** Security versus area versus the timing of PQC decapsulation algorithms, after loop unrolling. **Analysis:** Loop unrolling provides significant reduction in latency. Except for Classic McEliece and MQDSS, none of them have a latency of over 1 Million cycles. The maximum reduction in latency of 45× is for NTRU-HRSS. **Takeaway:** Similar to encapsulation, loop unrolling reduces latency for PQC decapsulations. This comes with extra hardware. For Saber, the increase in hardware overhead is 12×, in # of LUTs. If an IoT device requires high security (security level 5), Classic McEliece can be used, since the area overhead is low. For IoT devices where high level of security is not required, security level 1 algorithm with low area overhead like SPHINCS+ (Signature) can be used. On the other hand, CRYSTALS-Dilithium (Signature), providing security of level 1, can be used in servers, owing to its low latency.

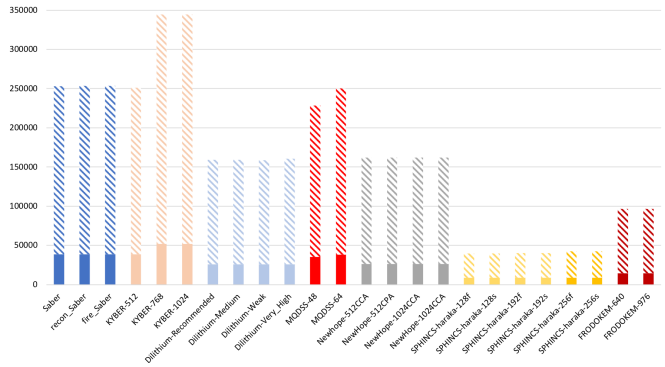


Fig. 7: Hardware overhead in terms of number of LUTs and FFs for various implementations of PQC encapsulation algorithms. FFs are shown in solid colors and LUTs as hashed patterns. Implementations of the same algorithm are marked with the same color for ease of comparison. KYBER encapsulation is the largest and has the maximum variation in area relative to security levels. KYBER-768 requires 40% more hardware over KYBER-512.

vary in key sizes). We examine how the hardware overhead varies with security strength. We run experiments on a baseline implementation, i.e., with no optimizations. Figure 7 plots the hardware overhead of PQC algorithms as flip-flops and LUTs.

Algorithm	Sec. Level	FF	LUT	Clock (nsec)	Latency
KEM algorithms					
CRYSTALS Kyber	1	11699	1307815	15	31669
Newhope	1	25639	136457	15	307847
FrodoKEM	1	105875	179290	10	335891
NTRU-HRSS	1	12225	75141	15	100208
LEDACrypt	1	13157	102496	10	11075
Saber	3	40824	234171	15	367099
Classic McEliece	5	60270	840430	10	3787729
Signature algorithms					
CRYSTALS Dilithium	1	146076	1327355	10	155166
SPHINCS+	1	20628	66750	10	468789803
qTESLA	1	112657	346020	10	63736
MQDSS	2	47441	270713	15	25825918

TABLE VIII: **Description:** Security versus area versus the timing of PQC encapsulation algorithms, after loop pipelining. **Analysis:** Similar to loop unrolling, pipelining also reduces the overall latency for the PQC encapsulation algorithms. Among the KEM algorithms, only Classic McEliece has a latency of more than 1 million cycles. The major difference with Table VI is with respect to the signature algorithm, MQDSS. While loop unrolling could not modify its latency, pipelining can reduce the latency by 50%. **Takeaway:** Loop pipelining reduces the latency of PQC encapsulation algorithms, with an increase of hardware area. The improvement in latency compared to loop unrolling is not consistent. After pipelining, LIMA (KEM, security level 5) emerges as an ideal candidate for both IoT devices and servers, with low area and low latency. For low security IoT devices, security level 1 algorithms with low area overhead like NTRU-HRSS (KEM) and RLIZARD (Signature) may be used. None of the Signature algorithms provide low latency after pipelining. Among the KEM algorithms, BIG_QUAKE provides high security (level 3) as well as low latency. Hence it is ideal for server applications. CRYSTALS-Kyber also provides low latency; however, its security level is only 1. Among the KEM algorithms, pipelining generates a faster design compared to loop unrolling for only 3-of-11 algorithms. On the other hand, for signature algorithms, pipelining provides better latency for 2-of-4 algorithms compared to unrolling.

Figure 8 reports the hardware overhead in terms of the number of flip-flops and LUTs, for the PQC decapsulation algorithms.

I. ASIC Implementation of PQC decapsulation algorithms.

In this section, we report the ASIC implementations of PQC decapsulation algorithms. All the designs are synthesized with a 5ns clock period, 65 nm GF LPE library and 2-stage compilation using Synopsys DC ASIC synthesis tool. The synthesis results, indicating the maximum operational frequency, the area and power are shown in Table X. The ASIC synthesis flow accepts the RTL generated by the HLS tool with some RTL changes done manually before Synopsys DC is able to synthesize them.

V. Deep Dive using three Signature PQCs

In this section, we will look into three Signature-based PQC algorithms and analyze the hardware implementations across various security levels. Furthermore, we will explore the impacts of various design optimizations, for both signature and verification. In this section, all implementations have been performed on a Xilinx Artix-7 FPGA board. This is the platform that NIST is considering as the comparison platform. The three algorithms are qTESLA, Crystals-Dilithium, and MQDSS. Each has been verified using KATs in NIST submissions.

Algorithm	Sec. Level	Baseline	Loop Unrolling	Loop Pipeline
KEM algorithms				
CRYSTALS Kyber	1	1.3×10^{11}	1.0×10^{11}	$\mathbf{4.1 \times 10^{10}}$
Newhope	1	9.2×10^{10}	9.2×10^{10}	$\mathbf{4.2 \times 10^{10}}$
FrodoKEM	1	$\mathbf{3.9 \times 10^{10}}$	5×10^{10}	6×10^{10}
NTRU-HRSS	1	5.1×10^{10}	$\mathbf{1.5 \times 10^{9}}$	7.5×10^{9}
LEDACrypt	1	3.4×10^{9}	5.2×10^{9}	$\mathbf{1.1 \times 10^{9}}$
Saber	3	1.1×10^{11}	8.9×10^{10}	$\mathbf{8.6 \times 10^{10}}$
Classic McEliece	5	1.0×10^{13}	$\mathbf{2.2 \times 10^{12}}$	3.2×10^{12}
Signature algorithms				
CRYSTALS Dilithium	1	8.1×10^{10}	$\mathbf{1.1 \times 10^{9}}$	2.1×10^{11}
SPHINCS+	1	$\mathbf{2.0 \times 10^{13}}$	2.0×10^{13}	3.1×10^{13}
qTESLA	1	2.9×10^{10}	$\mathbf{2.0 \times 10^{10}}$	2.2×10^{10}
MQDSS	2	9.5×10^{12}	1.1×10^{13}	$\mathbf{7.0 \times 10^{12}}$

TABLE IX: **Description:** Security vs. LAP for optimizations on PQC encapsulations. The minimum values for each algorithm are in bold. **Analysis:** For most KEM algorithms, pipelining produces the best latency-area product. Only for two (NTRU-HRSS and Classic McEliece) loop unrolling provides the best LAP. For FrodoKEM and SPHINCS+, the baseline implementation has the best LAP, i.e., optimizations actually deteriorate the results. **Takeaway:** LEDACRYPT (KEM) and CRYSTALS-Dilithium (Signature) have low LAPs.

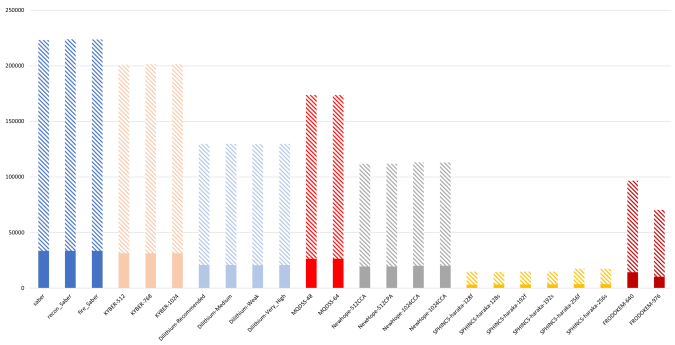


Fig. 8: # of flip-flops (FF) and LUTs used by PQC decapsulation algorithms. FFs are shown in solid colors, while LUTs as hashed patterns. Implementations of the same algorithm have the same color. Except FRODO-KEM, none of the decapsulations have noticeable difference in FF/LUT count. The variation in LUT and FFs for the two security levels of Frodo-KEM is ~40%. Saber uses the most LUTs and FFs.

A. Comparison across security level 1

We will compare the area and performance of the three algorithms for security level 1. The parameters we analyze are number of FFs, number of LUTs, the latency and the LAP. The comparisons for the signature and verification parts are shown in Table XI and XII.

Algorithm	Sec. Level	Clock (MHz)	Area (μm^2)	K Gates	Power (mW)
KEM algorithms					
CRYSTALS-Kyber	1	200	3378515	1340.68	39.21
Newhope	1	168.6	3208999	1273	38.02
FrodoKEM	1	200	10721	4.25	0.14
NTRU-HRSS	1	169.5	1246869	495	14.3
Saber	3	137.75	4774529	1895	54.49
Signature algorithms					
CRYSTALS-Dilithium	1	157.7	4774529	1602.6	51.24
SPHINCS+	1	200	19477.8	7.73	0.28
MQDSS	2	100	9341007	3706	120

TABLE X: **Description:** ASIC synthesis of some of the studied PQC decapsulation algorithms. **Analysis:** FrodoKEM (security level 1, KEM) and SPHINCS+ (security level 1, Signature) have small decapsulation modules which consume the least power and hence can be used in small IoT devices. **Takeaway:** Big_QUAKE (security level 3) is a good compromise of security, performance, and power for use in servers.

Algorithm	FF	LUT	Clock (ns)	Latency	LAP
qTESLA	26299	126732	12.65	537092	6.8×10^{10}
CRYSTALS-Dilithium	27132	123655	8.738	485963	6.0×10^{10}
MQDSS	21841	106035	17.05	34502428	3.6×10^{12}

TABLE XI: **Description:** Analysis of hardware implementations of "crypto_sign" components of Signature-based PQC algorithms. **Analysis:** While area overheads are more or less the same (MQDSS occupying 83% less area than the others), the overall latency and LAP for MQDSS is extremely high. MQDSS takes 71 $\times$ more latency and 61 $\times$more LAP compared to CRYSTALS-Dilithium. **Takeaway:** At security level 1, CRYSTALS-Dilithium is the best algorithm for "crypto_sign", since it occupies the least LAP.

Algorithm	FF	LUT	Clock (ns)	Latency	LAP
qTESLA	17780	87067	12.58	80422	7.0×10^{9}
CRYSTALS-Dilithium	14712	63863	8.738	149950	9.5×10^{9}
MQDSS	23072	117097	17.045	25686731	3.0×10^{12}

TABLE XII: **Description:** Analysis of hardware implementations of "crypto_sign_open" components of Signature-based PQC algorithms. **Analysis:** CRYSTALS-Dilithium has the least area, while qTESLA has the least latency shows significantly better performance. **Takeaway:** At security level 1, qTesla is the best algorithm for "crypto_sign_open", since it occupies the least LAP.

B. Comparison across security level 3

In this section, we compare the hardware implementation cost and performance for both "crypto_sign" and "crypto_sign_open" for the three Signature-based algorithms at security level 3. The results for signature and verification are shown in Table XIII and XIV respectively.

C. Comparison across various security levels

In this section, we will compare the area overhead and performance for the algorithms across various security levels. Section V-C1 and Section V-C2 compare the results for "crypto_sign" and "crypto_sign_open" components of the algorithms.

1) Signature: In this section, we compare the area (in terms of FF and LUT), performance (in terms of latency) and LAP for implementing "crypto_sign" portion of the three algorithms for security levels 1 and 3. The results are shown in Figure 9. For qTESLA and CRYSTALS-Dilithium, the area overhead at security levels 1 and 3 are similar. For MQDSS, the are overhead is 13-16% more in security level 3 compared to security level 1. For qTESLA, the latency reduces at security level 3, compared to level 1. For the other two algorithms, the latency increases drastically at higher security level. Comparing both area and performance, we can observe that only qTESLA has lower LAPs at security level 3 compared to level 1. qTESLA at security level 3 has the lowest LAP and provides the highest security among all the alternatives discussed in this section. Therefore, we suggest it as the best algorithm to use among these three.

Algorithm	FF	LUT	Clock (ns)	Latency	LAP
qTESLA	26011	126311	12.65	347655	4.3×10^{10}
CRYSTALS-Dilithium	27308	123933	8.738	826832	1.0×10^{11}
MQDSS	24748	123170	16.378	119353597	1.4×10^{13}

TABLE XIII: **Description:** Analysis of hardware implementations of "crypto_sign" components of Signature-based PQC algorithms at security level 3 **Analysis:** While area overheads are more or less the same for all algorithms, similar to Table XI, the overall latency and LAP for MQDSS is extremely high. **Takeaway:** At security level 3, qTESLA is the best algorithm for "crypto_sign", since it occupies the least LAP.

Algorithm	FF	LUT	Clock (ns)	Latency	LAP
qTESLA	17754	86142	12.65	201027	1.7×10^{10}
CRYSTALS-Dilithium	14783	63980	8.738	297592	1.9×10^{10}
MQDSS	23149	117574	17.045	87861777	1.0×10^{13}

TABLE XIV: **Description:** Analysis of hardware implementations of "crypto_sign_open" components of Signature-based PQC algorithms. **Analysis:** CRYSTALS-Dilithium has the least area, while qTESLA has the least latency. **Takeaway:** At security level 3, qTESLA is the best algorithm for "crypto_sign_open", since it occupies the least LAP. On the other hand, MQDSS has the worst LAPs.

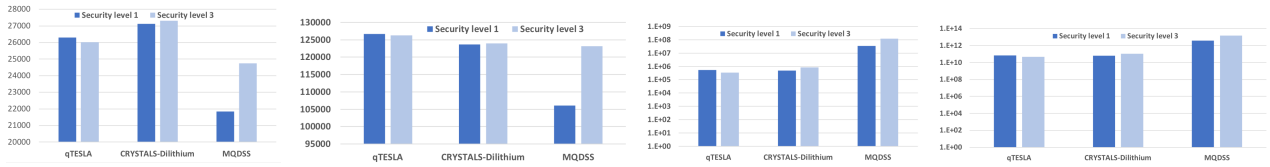
2) Signature Verification: In this section, we compare the area, performance and LAPs for the signature verification, i.e., "crypto_sign_open" component of the three algorithms across the two security levels. The results are shown in Figure 10. For all the algorithms, the area overhead remains similar for the two security levels. On the other hand, consistently, the latency and LAP is higher at security level 3 compared to security level 1. qTESLA has the least LAP at both the security levels.

D. Optimization

In this section, we will analyze how the various optimization techniques (Loop unrolling and Loop pipelining) help to reduce the overall latency of the PQC algorithms. We will compare the same parameters we used in Section V-C for three types of implementations – baseline, loop unrolling and loop pipelining, at security level 1. The results for implementation of "crypto_sign" and "crypto_sign_open" are shown in Figure 11 and Figure 12, respectively.

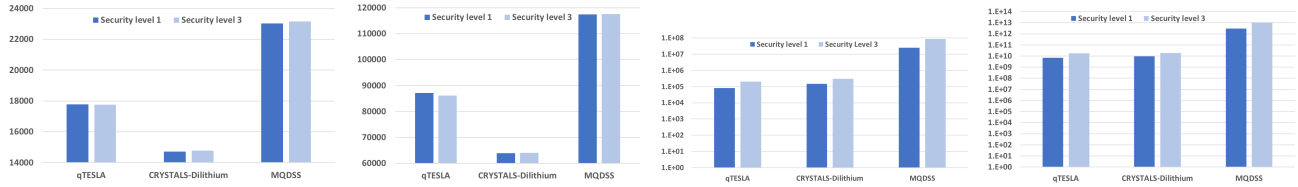
E. Implications of hardware-optimized $keccak$

$keccak$ is a family of sponge functions used by the three algorithms and by many NIST round 2 PQCs. Over the years, researchers developed hardware-optimized variants of $keccak$. In this section, we study the effect of one of these and observe how the area and performance overhead changes.We use $keccak$ implementation from [39]. They developed a $KeccakF1600\_StatePermute$ function implementation, with separate functions used for internal operations like θ, ρ, π, χ and ι. Figure 13 compares the parameters for implementation of "crypto_sign", while Figure 14 compares "crypto_sign_open". For consistency, all implementations are for security level 1. In Figures 15, 16, we compare overheads with loop unrolling. For fair comparison, we use the same directives for both implementations. Similar comparison for loop pipelining are shown in Figures 17, 18.



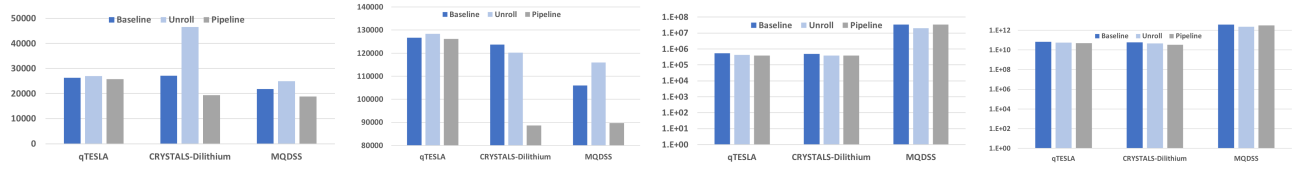
(a) (b) (c) (d)

Fig. 9: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign" component of the three PQC algorithms across different security levels. (a) For qTESLA and CRYSTALS-Dilithium, the difference in FF count is not significant, while for MQDSS, there is a 13% increase in number of FFs for implementing at security level 3 compared to security level 1. (b) Similar to FFs, for qTESLA and CRYSTALS-Dilithium, the difference in LUT count is not significant, while for MQDSS, there is a 16% increase in number of FFs for implementing at security level 3 compared to security level 1. (c) Since the latency of MQDSS is two orders of magnitude more than either qTESLA or CRYSTALS-Dilithium, this graph is plotted in the logarithmic scale. qTESLA at security level 3 and CRYSTALS-Dilithium at security level 1 have the least latency. (d) qTESLA at security level 3 has best LAP and MQDSS at both security levels have worst LAP.



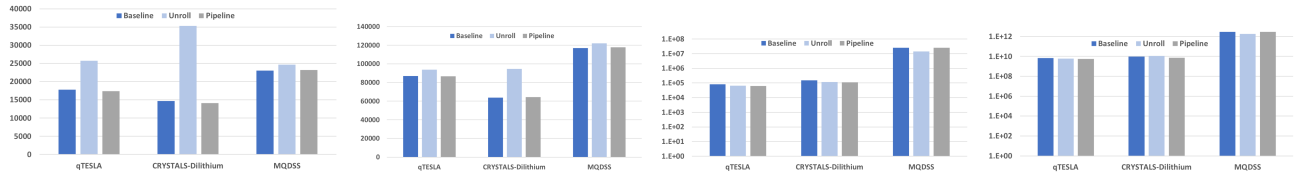
(a) (b) (c) (d)

Fig. 10: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign_open" component of the three PQC algorithms across different security levels. (a) Compared to signature, the difference in FF count is not significant for any algorithm. While in Figure 9, CRYSTALS-Dilithium occupies the highest number of FFs, when implementing "crypto_sign_open", it requires the least number of FFs. (b) Compared to Figure 9, the difference in LUT count is not significant for any algorithm. (c) qTESLA at security level 1 requires the least latency. MQDSS requires the highest latency for both "crypto_sign" and "crypto_sign_open". (d) qTESLA at security level 1 has best LAP and MQDSS at both security levels have worst LAP.



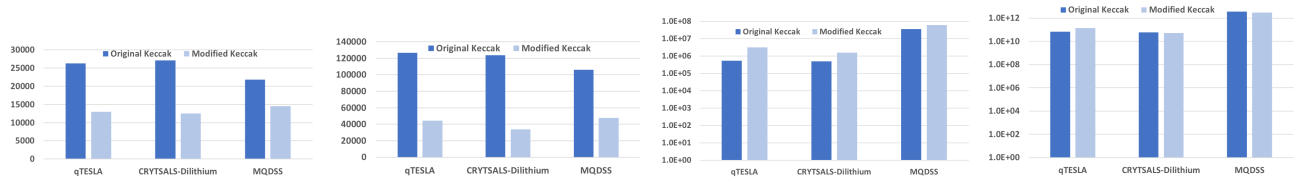
(a) (b) (c) (d)

Fig. 11: (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign" of the three PQC algorithms at security level 1 for different optimizations. (a) Loop pipelining yields the least number of FFs across, while loop unrolling requires the maximum number of FFs. (b) Loop pipelining yields the least number of LUTs for all designs. Except for CRYSTALS-Dilithium, the number of LUTs increases with loop unrolling. (c) Loop unrolling and pipelining reduce the latency compared to the baseline. (d) CRYSTALS-Dilithium and qTESLA have least LAP when loop pipelined. For MQDSS, the least LAP is obtained when loops are unrolled.



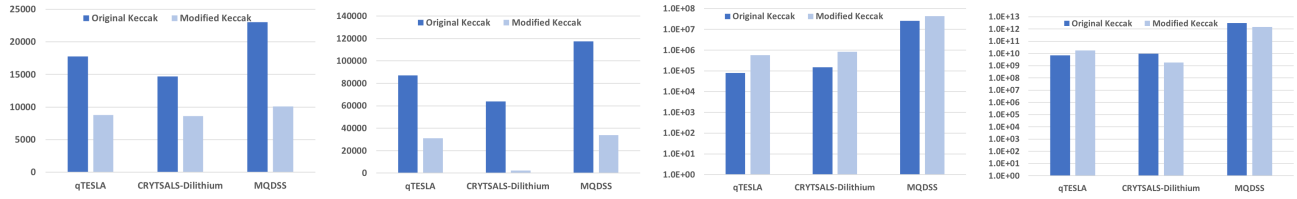
(a) (b) (c) (d)

Fig. 12: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign_open" component of the three PQC algorithms at security level 1 for different optimizations. Optimization with loop pipelines occupy the least number of FFs and LUTs across all designs, while loop unrolling requires the maximum number of FFs and LUTs. Optimization with loop unrolling and pipelines reduce the latency compared to baseline implementation. Except for MQDSS, the other two algorithms have least LAP when loop pipelining is performed. For MQDSS, the least LAP is obtained when loops are unrolled.



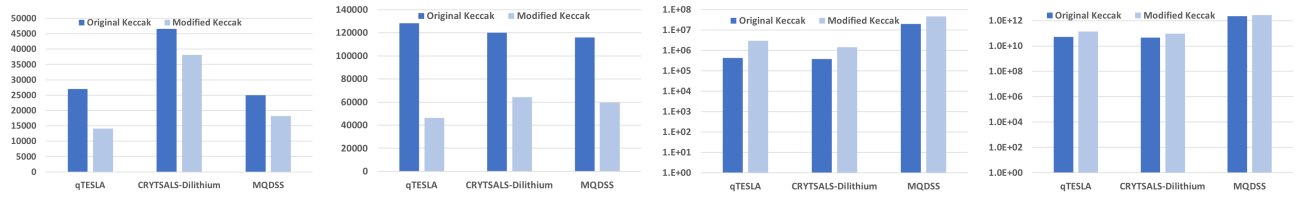
(a) (b) (c) (d)

Fig. 13: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign" component of the three PQC algorithms at security level 1 for two versions of $keccak$. The number of FFs and LUTs reduce considerably compared to the original $keccak$. CRYSTALS-Dilithium has 54% reduction in FF count and 73% reduction in LUTs. On the other hand, the latency increases with the modified $keccak$. The worst affected is qTESLA, which has a 5.8× increase in latency, while MQDSS incurs only 1.7× increase in latency. The LAP value, on the other hand, decreases for MQDSS and CRYSTALS-Dilithium and increase only for qTESLA. For MQDSS, the LAP is reduced by 21%.



(a) (b) (c) (d)

Fig. 14: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement"crypto_sign_open" component of the three algorithms at security level 1 for different versions of $keccak$. Similar to Figure 13, the number of FFs and LUTs reduce considerably compared to the original $keccak$. MQDSS has a maximum of 56% reduction in FF count and CRYSTALS-Dilithium has about 96% reduction in LUTs. The latency increases with the modified $keccak$. The worst affected is again qTESLA, which has a 7× increase in latency while MQDSS has 1.7× increase in latency. The LAP decreases considerably for all algorithms except qTESLA. CRYSTALS-Dilithium has 82% reduction in LAP compared to original $keccak$.



(a) (b) (c) (d)

Fig. 15: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement the "crypto_sign" component of the three PQC algorithms at security level 1 for different versions of $keccak$, after loop unrolling. The number of FFs and LUTs reduce considerably compared to the original $keccak$ version. qTESLA has about 48% reduction in FF count and 64% reduction in number of LUTs. On the other hand, the latency increases with the modified $keccak$ implementation. The worst affected is qTESLA, which has a 7.15 × increase in latency. In contrast to Figure 13, the LAP values consistently increase for all three algorithms. For MQDSS, the increase in LAP is minimum – 20%.

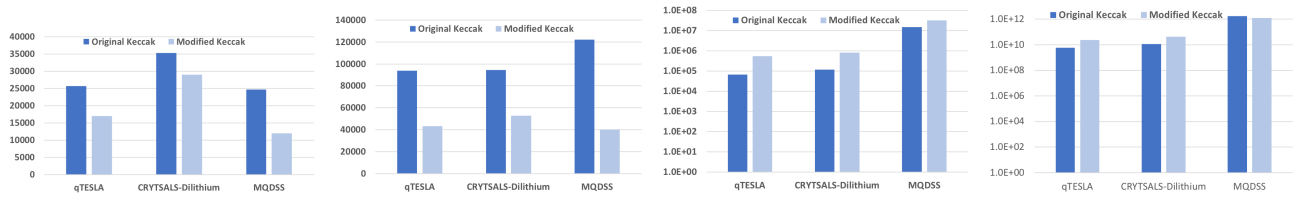


Fig. 16: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement "crypto_sign_open" component of the three PQC algorithms at security level 1 for different $keccak$ after loop unrolling. The number of FFs and LUTs reduce considerably compared to the original $keccak$. MQDSS has about 52% reduction in FF count and 61% reduction in number of LUTs. On the other hand, the latency increases with the modified $keccak$. The worst affected is qTESLA, which has a 8.4× increase. The LAP increases for qTESLA and CRYSTALS-Dilithium but reduces by 28% for MQDSS.

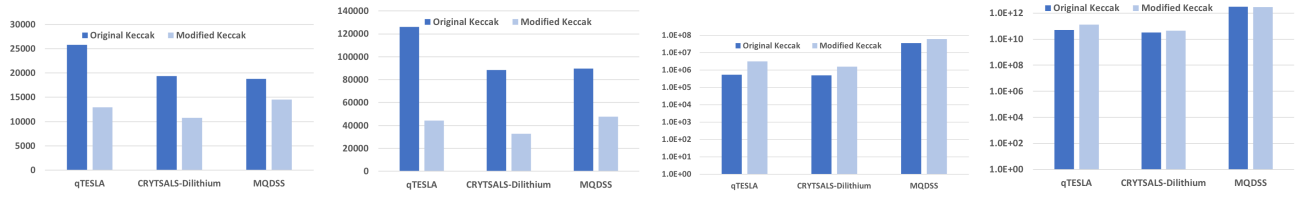
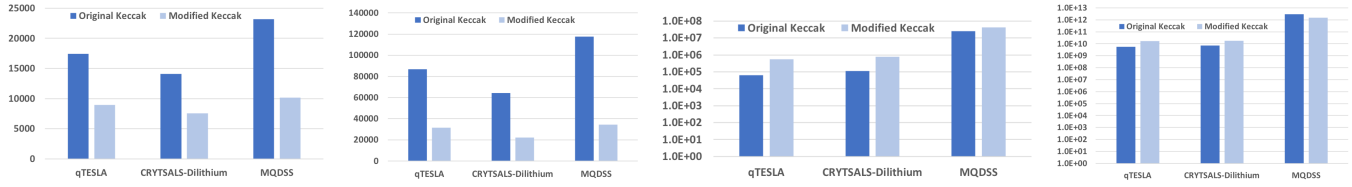


Fig. 17: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement "crypto_sign" component of the three PQC algorithms at security level 1 for different versions of $keccak$, after loop pipelining. The number of FFs and LUTs reduce considerably compared to the original $keccak$. qTESLA has a 50% reduction in FF count and 65% reduction in number of LUTs. On the other hand the latency increases with the modified $keccak$ implementation. The worst affected is qTESLA, which has a 7.7 × increase in latency. The LAP values increase for qTESLA and CRYSTALS-Dilithium, but reduce by 5% for MQDSS. Overall, the relative increase in LAPs is less than when loop unrolling in Figure 15.



(a) (b) (c) (d)

Fig. 18: Comparison of (a) FFs, (b) LUTs, (c) Latency and (d) LAP to implement "crypto_sign_open" of the three algorithms at security level 1 using different versions of $keccak$, after loop pipelining. The number of FFs and LUTs reduce considerably compared to the baseline $keccak$. MQDSS has 56% reduction in FF count and 70% reduction in number of LUTs. Although the latency of MQDSS increases by 67% with the modified $keccak$, the overall LAP value reduces by 52%.

VI. Takeaways of Systematization of Knowledge

In this systematization of knowledge (SOK) study, we implemented PQC algorithms using a common design framework and a common target FPGA platform. This is an ongoing SOK study. In the end, we expect to assess the hardware implementations of all 26 NIST PQC Competition Round 2 KEM and Signature algorithms. Key takeaways of this preliminary SOK study are:

1) Among the KEM algorithms of security level 1, NTRU-HRSS is superior in terms of latency and LAP.
2) Among the Signature algorithms of security level 1, CRYSTALS-Dilithium is superior for Signature and qTesla is superior for Signature verification. SPHINCS+ is the costliest in terms of latency and LAP.
3) This study has only two implementations each of security levels 3 and 5. We will continue analyzing the area-performance trade-offs for more high-security algorithms. Saber is ideal for high (level 3) security with low latency for server applications.
4) In low-power IoT devices, one needs low-area, compact designs. For example, without optimizations, NTRU-HRSS and SPHINCS+ FPGAs are ideal for IoT with security level 1. Low-latency Crystals-KYBER FPGA designs are good for servers with security level 1.
5) FrodoKEM and SPHINCS+ ASICs have small decapsulation modules which consume low power and useful in IoT devices, if security level 1 suffices. NTRU-HRSS and NewHope (security level 1) are the fastest ASICs and good for servers.
6) For KEM algorithms with high security level (3 and 5), loop unrolling is more effective in reducing latency. For Signature algorithms, loop pipelining is more effective, for algorithms of security level 1. We experimented on one signature algorithm of security level 2 for which, loop unrolling was more practical.
7) For low-security (level 1) KEMs, latency rankings change when optimized. When loop unrolling is performed, NTRU-HRSS is the fastest. However, pipelining directives render CRYSTALS-Kyber the fastest KEM.
8) We performed specific case studies using three Signature algorithms – qTESLA, CRYSTALS-Dilithium and MQDSS in Section V. We noted the area and performance overhead when hardware implementations of these algorithms are performed for various security levels as well as for various optimizations like loop unrolling and pipelining.
9) In our deep dive case study, qTESLA and CRYSTALS-Dilithium had the least LAP when loop pipelined, while MQDSS had the least LAP when loop unrolled. A future exercise will understand which optimization is appropriate for which algorithm.
10) Different components of the algorithms can be optimized for low-area by rewriting the code and running HLS. As shown in Section V-E, using a different implementation of $KeccakF1600\_StatePermute$ (from [39]) reduces the area.
11) Even when the same optimization directives are applied with the two different $keccak$ versions, the modified $keccak$ function (from [39]) reduces the overall area overhead. However, in this case, the LAPs don't decrease as much (in fact, the LAP increases in most cases), compared to the baseline (without optimization). The reduction in LAP is more for loop pipelining compared to loop unrolling.
12) PQC hardware implementations are not optimized for side-channel resistance. PQC researchers can use our implementations for further hardware-security analysis such as timing, power, and fault-attack side-channel analysis. This study is ongoing with collaborators.
13) Further optimizations can be done on these designs obtained using HLS. The RTL obtained in this study are directly from Xilinx Vivado HLS without any hand-tailored optimizations to reduce area/latency.
14) In this study, we used the same board (either Virtex or Artix) for implementing both low area and low latency variants of the PQC algorithms. As a future exercise, we will investigate which FPGA architectures are ideal for low area/low latency implementations. This will be similar to [17] that implemented a low area version on Cortex-M0 and a faster version on Cortex-M4.

REFERENCES

[1] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[3] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the development of the advanced encryption standard (AES)," 2001. Accessed 31 December 2018.

[4] K. Aoki and H. Lipmaa, "Fast implementations of aes candidates.," in *AES Candidate Conference*, pp. 106–120, 2000.

[5] D. Micciancio, "Lattice-based cryptography," in *Encyclopedia of Cryptography and Security*, pp. 713–715, Springer, 2011.

[6] R. Cramer, L. Ducas, and B. Wesolowski, "Short stickelberger class relations and application to ideal-svp," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 324–348, Springer, 2017.

[7] R. J. Mceliece, "A public-key cryptosystem based on algebraic," *Coding Theory*, vol. 4244, pp. 114–116, 1978.

[8] D. J. Bernstein, T. Lange, and C. Peters, "Attacking and defending the McEliece cryptosystem," in *Workshop on Post-Quantum Cryptography*, pp. 31–46, 2008.

[9] A. S. Al Abdouli, M. Al Ali, E. Bellini, F. Caullery, A. Hasikos, M. Manzano, and V. Mateu, "Drankula: a mceliece-like rank metric based cryptosystem implementation," in *International Joint Conference on e-Business and Telecommunications*, pp. 64–75, 2018.

[10] P. Loidreau, "A new rank metric codes based encryption scheme," in *International Workshop on Post-Quantum Cryptography*, pp. 3–17, Springer, 2017.

[11] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 392–407, 2000.

[12] V. Dubois, P.-A. Fouque, A. Shamir, and J. Stern, "Practical cryptanalysis of sflash," in *Annual International Cryptology Conference*, pp. 1–12, 2007.

[13] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Workshop on Post-Quantum Cryptography*, pp. 19–34, 2011.

[14] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, p. 129, 2019.

[15] H. Nejatollahi, N. Dutt, and R. Cammarota, "Special session: trends, challenges and needs for lattice-based cryptography implementations," in *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 1–3, IEEE, 2017.

[16] W. Wang, J. Szefer, and R. Niederhagen, "FPGA-based niederreiter cryptosystem using binary goppa codes," in *International Conference on Post-Quantum Cryptography*, pp. 77–98, Springer, 2018.

[17] A. Karmakar, J. M. B. Mera, S. S. Roy, and I. Verbauwhede, "Saber on arm," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 243–266, 2018.

[18] A. Hülsing, J. Rijneveld, and P. Schwabe, "Armed sphincs," in *Public-Key Cryptography–PKC 2016*, pp. 446–470, Springer, 2016.

[19] M. R. Albrecht, C. Hanser, A. Hoeller, T. Pöppelmann, F. Virdia, and A. Wallner, "Implementing rlwe-based schemes using an rsa co-processor," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 169–208, 2019.

[20] H. Nejatollahi, N. D. Dutt, I. Banerjee, and R. Cammarota, "Domain-specific accelerators for ideal lattice-based public key protocols.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 608, 2018.

[21] Xilinx Inc., "Vivado Design suite user guide - designing with ip (ug896)." https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. [Online; accessed 13-January-2019].

[22] R. Domingo, R. Salvador, H. Fabelo, D. Madroñal, S. Ortega, R. Lazcano, E. Juárez, G. Callicó, and C. Sanz, "High-level design using intel FPGA opencl: A hyperspectral imaging spatial-spectral classifier," in *IEEE International Symposium on Reconfigurable Communication-centric Systems-on-Chip*, pp. 1–8, 2017.

[23] J. Zhu and D. Gajski, "A unified formal model of ISA and FSMD," in *Proceedings of the International Workshop on Hardware/Software Codesign*, pp. 121–125, 1999.

[24] "Catapult." https://www.mentor.com/hls-lp/catapult-high-level-synthesis/. [Online; accessed 13-January-2019].

[25] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "Legup: high-level synthesis for fpga-based processor/accelerator systems," in *Proceedings of ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 33–36, 2011.

[26] "Synopsys DC." https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html. [Online; accessed 13-January-2019].

[27] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[28] G. Pelosi and P. Santini, "Ledakem: A post-quantum key encapsulation mechanism based on qc-ldpc codes," in *Post-Quantum Cryptography: 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, vol. 10786, p. 3, Springer, 2018.

[29] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *International Conference on Cryptology in Africa*, pp. 282–305, 2018.

[30] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-kyber: a CCA-secure module-lattice-based KEM," in *IEEE European Symposium on Security and Privacy*, pp. 353–367, 2018.

[31] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange-a new hope," in *USENIX Security Symposium*, 2016.

[32] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, D. Stebila, *et al.*, "FrodoKEM–learning with errors key encapsulation." https://frodokem.org/, 2017.

[33] A. Hülsing, J. Rijneveld, J. M. Schanck, and P. Schwabe, "NTRU-HRSS-KEM: algorithm specifications and documentation." https://ntru-hrss.org/, 2017.

[34] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals–dilithium: Digital signatures from module lattices." https://pq-crystals.org/dilithium/index.shtml, 2018.

[35] S. Kölbl, M. M. Lauridsen, F. Mendel, and C. Rechberger, "Haraka v2–efficient short-input hashing for post-quantum applications," *IACR Transactions on Symmetric Cryptology*, pp. 1–29, 2016.

[36] A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe, "From 5-pass mq-based identification to mq-based signatures.," *IACR Cryptology ePrint Archive*, p. 708, 2016.

[37] N. Bindel, S. Akleylek, E. Alkim, P. Barreto, J. Buchmann, E. Eaton, G. Gutoski, L. P. Kramer, Juliane, J. Ricardini, and Z. Gustavo, "qtesla: Efficient and post-quantum secure lattice-based signature scheme." https://qtesla.org/, year=2019.

[38] E. Homsirikamol and K. Gaj, "Toward a new HLS-based methodology for FPGA benchmarking of candidates in cryptographic competitions: The CAESAR contest case study," in *IEEE International Conference on Field Programmable Technology*, pp. 120–127, 2017.

[39] K. Gaj, E. Homsirikamol, and M. Rogawski, "Comprehensive comparison of hardware performance of fourteen round 2 sha-3 candidates with 512-bit outputs using field programmable gate arrays," in *2nd SHA-3 Candidate Conference, Santa Barbara, August*, pp. 23–24, 2010.

APPENDIX

A. Two Performance Optimizations

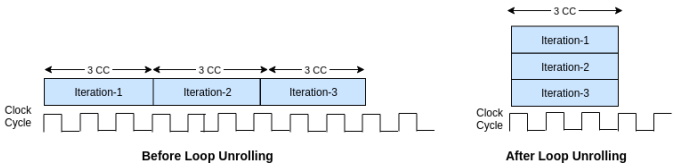


Fig. 19: Loop unrolling for a *for* loop of count 3.

1) Loop unrolling: The latency of a design depends upon the functions and loops. The loops are executed one iteration at a time (rolled). Thus, rolled loops increase latency. Fully unrolling a loop can minimize the latency, while using more hardware resources. Figure 19 explains loop unrolling. Vivado HLS provides the option to partially unroll the loop to balance the performance and area.

2) Loop Pipelining: Loop pipelining can be used to improve latency. Figure 20 explains loop pipelining. This optimizes both hardware and latency. We synthesize the algorithms by adding a directive to pipeline the longer loops.

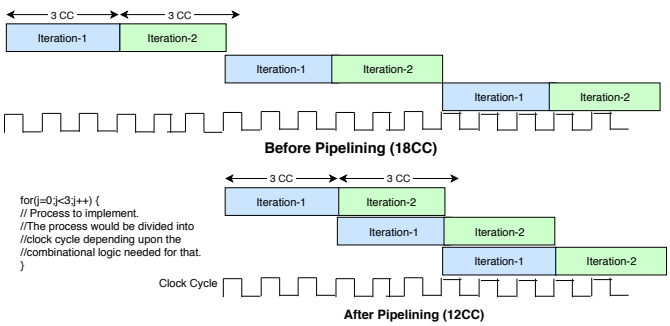


Fig. 20: Example of loop pipelining.

B. Design Space Exploration of CRYSTALS_KYBER

The HLS-based framework can aid in design exploration for each PQC algorithm. Let us consider CRYSTALS_KYBER as an example. An analysis of (`crypto_kem_enc()`) reveals that `indcpa_enc()` limits its latency and shows that `gen_matrix()` has the highest latency.

Table XV summarizes latencies of the functions in `crypto_kem_enc()` and `indcpa_enc()` and the loops in the function `gen_matrix()`. The function `gen_matrix()` has four loops of which Loop 1 is a simple loop. Loop 4 is nested within Loop 3, which is nested in Loop 2. Loop 4 calls an external function `Keccak_squeezeblocks()`, while Loop 3 calls two external functions – `keccak_absorb()` and `keccak_squeezeblocks()`. Loop 2 does not call any functions and just iterates over Loop 3 multiple times. Function `keccak_squeezeblocks()` has two loops, the first of which calls functions `KeccakF1600_StatePermute()` and the second `store64()`. The second loop of `Keccak_squeezeblocks()` is embedded in the first. `Keccak_absorb()` has 6 loops, of which loops 1, 4, 5 and 6 are single-line loops. Loop 3 calls function `load64()` and is embedded inside Loop 2. Loop 2 invokes function `KeccakF1600_StatePermute()`. The three functions `KeccakF1600_ StatePermute()`, `store64()` and `load64()` have single loops each. The call graph for `gen_matrix()` is shown in Figure 21.

```
void gen_matrix(polyvec *a, const unsigned
char *seed, int transposed) {
  unsigned int pos=0, ctr, nblocks=4,
  i, j;
  uint16_t val;
  uint8_t buf[SHAKE128_RATE*nblocks];
  uint64_t state[25]; // CSHAKE state
  unsigned char extseed[KYBER_SYMBYTES+2];
  for(i=0;i<KYBER_SYMBYTES;i++)
  /* Loop  1*/
    extseed[i] = seed[i];
  for(i=0;i<KYBER_K;i++) {  /*  Loop  2*/
    for(j=0;j<KYBER_K;j++) {  /* Loop 3 */
      ctr = pos = 0;
      if(transposed) {
        extseed[KYBER_SYMBYTES]   = i;
        extseed[KYBER_SYMBYTES+1] = j;
      } else {
        extseed[KYBER_SYMBYTES]   = j;
        extseed[KYBER_SYMBYTES+1] = i;
      }
      shake128_absorb(state,extseed,
      KYBER_SYMBYTES+2);
      shake128_squeezeblocks(buf,nblocks,
      state);
      while(ctr < KYBER_N) {  /* Loop 4 */
        val = (buf[pos] | ((uint16_t)
        buf[pos+1] << 8)) & 0x1fff;
        if(val < KYBER_Q)
            a[i].vec[j].coeffs[ctr++] = val;
        pos += 2;
        if(pos > SHAKE128_RATE*nblocks-2) {
         nblocks = 1;
         shake128_squeezeblocks(buf,
         nblocks,state);
         pos = 0;
        }
       }
      }
     }
    }
```

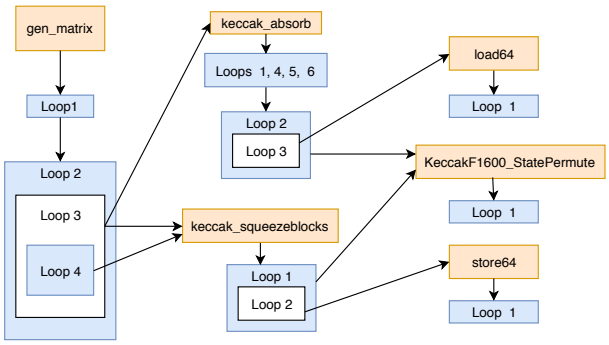


Fig. 21: Call graph of `gen_matrix()`.

1) Loop Unrolling: We unroll loops in the last-level functions, `store64()`, `load64()` and `KeccakF1600_StatePermute()`. We unroll innermost loops in the top functions, i.e., Loop 2 in `keccak_squeezeblocks()` and Loop 4 in `gen_matrix()`. The unrolling factor is set to 1 and inline function calls.

Function/Loop	Latency (clock cycles)
Top function: crypto_kem_enc	
indcpa_enc	40869
keccak_absorb	8056
randombytes	5895
Top function: indcpa_enc	
gen_matrix	10835
poly_getnoise	2595
polyvec_compress	2565
Top function: gen_matrix	
gen_matrix_loop 1	64
gen_matrix_loop 2	8419
gen_matrix_loop 3	2637
gen_matrix_loop 4	826

TABLE XV: Critical functions and loops that determine the latency of CRYSTALS-KYBER.

2) Loop Pipelining: We mark unrolled loops for pipelining. We set the target initiation interval for pipelining to 1, except for the loop in `KeccakF1600_StatePermute()`. Since `KeccakF1600_StatePermute()` is critical, does a lot of operations in a single loop, we created the fastest possible pipeline architecture. The Table XVI shows the results. The last column shows the speedup compared to the baseline. Reduction in latency provided by loop pipelining is comparable to loop unrolling. However, loop pipelining yields a design with 4% less area overhead.

Implementation	FF	LUT	MUX	Latency	Speedup
Baseline	6685	39357	606	10835	-
Loop Unrolling	5415	43955	454779	9513	1.23 ×
Loop Pipelining	5439	42206	435728	904	1.23 ×

TABLE XVI: Area and timing overhead for different implementations of `gen_matrix()`.