NIST Post-Quantum Cryptography- A Hardware Evaluation Study

Kanad Basu[1], Deepraj Soni[1], Mohammed Nabeel[2] and Ramesh Karri[1]

[1] New York University, kb150,dss545,rkarri@nyu.edu
[2] New York University, Abu Dhabi mohammed.nabeel@nyu.edu

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 comparison of the NIST PQC competition candidates. For this, we use a High-Level Synthesis (HLS) hardware design methodology to map high-level C specifications of selected PQC candidates into both FPGA and ASIC implementations.

Keywords: Post-quantum Cryptography, Hardware Evaluation

1 Introduction

We live in an age of Internet-of-things (IoT), where all electronic devices around us are connected to the internet, and hence, are vulnerable to security threats. Over the past few decades, public key cryptography has become fundamental security protocol for all forms of digital communication, wired or wireless. Public key cryptography is composed of three main cryptographic functions, namely (a) public key encryption, (b) digital signatures, and key exchange [1]. Current deployed schemes (RSA or Elliptic Curve Cryptography algorithms) provide security guarantees based on the difficulty of solving the integer factorization and discrete logarithm problems.

Peter Shor from Bell Labs was the first to show that quantum computers can factorize integers in linear time. This renders traditional public key cryptography algorithms ineffective [2]. To counter these challenges faced by current generation of public key cryptography algorithms on quantum computers, researchers are investigating other robust alternatives such as lattice and code cryptography algorithms. In this paper, we present a hardware assessment of post-quantum cryptographic (PQC) algorithms that were submitted to the NIST PQC assessment.

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 competitions have become a way of choosing cryptographic standards. NESSIE (2000-2002), eSTREAM (2004-2008), CRYPTREC (2000-2002), SHA-3 (2007-2012) and CAESAR (2013-) embraced this competition approach. In all these contests, security was the principal yardstick. Performance in software, performance in application specific integrated circuits (ASIC), and feasibility of implementation using limited resources (small microprocessors and low-power hardware) are the secondary criteria. One can pick competitors that offer sufficient security and superior performance. In the AES competition, Rijndael had the fastest ASIC implementation and the second fastest FPGA implementation relative to its adversaries with identical security guarantees.

The objective of this work is to prepare an extensive hardware comparison between the leading NIST PQC candidates. The main contributions of this study are:

1. Developed systematic FPGA and ASIC design flows for PQC evaluation starting from a C specification. For this, we transform the C PQC specifications to make them implementable in hardware.
2. Studied the performance vs area trade-offs for 13 PQC encryption and decryption algorithms, including KEM- and Signature algorithms that are lattice, code, hash, and multivariate.
3. Optimized PQC implementations to increase latency using loop unrolling and loop pipelining.
4. Explored improvement of latency using algorithm → FPGA and ASIC hardware design flows.

The rest of the paper is organized as follows. Section 2 gives a background on Post-Quantum Cryptography. Section 3 describes the design flow and Section 4 presents experimental results. Section 5 presents two case studies and Section 6 presents key takeaways.

2 Post-Quantum Cryptography

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

- **Lattice cryptography** algorithms offer the best performance results, but are the least conservative among all (i.e., have been studied the least) [4]. The robustness of lattice cryptography builds on the hardness of the shortest vector problem (SVP). SVP entails approximating the minimal Euclidian length of a lattice vector. Even with a quantum computer, the problem is polynomial in n [1]. Several lattice cryptography algorithms are based on Short Integer Solutions (SIS), which is an average case problems. These problems are secure in the average case if the SVP is hard in the worst-case [5].
- **Code cryptography** uses error correcting codes. It offers the most conservative approach for public-key encryption/key encapsulation, as it is based on this well-studied problem that has been around for 40 years [6]. This class of algorithms use large keys and some attempts at reducing they key size have made these algorithms vulnerable to attacks [7]. However, recently, researchers proposed several techniques to reduce the key size, without compromising on the security strengths [8, 9].
- **Multivariate polynomial cryptography** relies on the difficulty of solving the multivariate polynomial algorithm over finite fields. Solutions of multivariate polynomial problems are NP-hard over any field and NP-complete even if all the equations are quadratic and the field is $GF(2)$ [10]. Multivariate schemes are preferred as signature schemes, since they offer the shortest signatures. Although multivariate schemes have been proposed, a few of them have been broken [11].
- **Hash digital signatures** resist quantum-computer attacks. These schemes are based on the security properties of the underlying cryptograhic hash functions (collision resistance or second pre-image resistance).
- **Other cryptographic methods:** include evaluating isogenies on super singular elliptic curves. Shor' algorithm is ineffective against evaluating isogenies [12].

2.1 Digital signature generation and key encapsulation methods

The NIST PQC standardization process is ongoing consolidating invulnerable candidates after each successive round. Each candidate in the NIST PQC contest realizes one of three

Table 1: NIST PQC round 1: A distribution of the digital signature and key encapsulation submissions and the underlying hard mathematical problem.

PQC	Hard math problem	Sign	KEM	Total
Lattice	Find shortest vector, closest vector	5	23	28
Code	Decode random linear code	3	17	20
Multivariate	Solve multivariate quadratic equations	8	2	10
Hash	second pre-image resistance of hash function	3	0	3
Isogeny	Find isogeny map btwn elliptic curves with same # of points	0	1	1
Other	-	2	5	7
Total	-	**21**	**48**	**69**

functions: public-key encryption, digital signatures, and key encapsulation mechanism (KEM). Based on mathematical complexity, PQC algorithms can be classified as: Lattice, Code, Multivariate, Hash and Isogeny. Table 1 shows the number of PQC round 1 submissions based on the functionality and mathematical complexity.

NIST has stated that they intend to standardize more than one algorithm in order to provide different trade-offs depending on the application (speed vs memory, etc). The direct comparison of security of the candidate PQC algorithms is challenging for lack of a standard quantum computing platform, differences in the mathematical functions that underlie the algorithms, and the sophisticated algorithm specifications compared to prior cryptographic contests. In this section, we will discuss PQC digital signatures and key encapsulation methods.

2.1.1 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 senders public key. The PQC digital signature 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 in this hardware assessment are: SPHINCS+ (Haraka), MQDSS, RLIZARD and Crystals-Dilithium.

2.1.2 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. Popular asymmetric cryptographic algorithms based on ECC and RSA work on this principle. However, these classical asymmetric cryptographic algorithms are vulnerable to quantum attacks. A solution 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 easily reconstruct a small M. Hence, the sender has to make M large. Second, if the attacker somehow derives M, he/she can easily obtain m. KEM schemes counter these two challenges.

In KEM, the sender generates a random number rn and then generates a symmetric key $M = KDF(rn)$, using a Key Derivation Function (KDF). A cryptographic hash is an example KDF and addresses the two challenges as follows: (i) KEM obviates padding and

this way reduces the key size.(ii) Since KDF is one-way, the attacker can not generate rn, even if she recovers M. KEM algorithms 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 encrypted key ck.
3. **crypto_kem_dec** takes in secret key sk and encrypted key ck and outputs key k.

We consider the following KEM PQC algorithms: BIG Quake, Newhope, Frodokem, Crystals-KYBER, NTSKEM, NTRU-HRSS, Classic McEliece, LIMA, Saber [1].

2.2 Security classification of PQC algorithms

NIST specified five security strength categories. **Security level 1** $\Longrightarrow$ equivalent to AES-128 key search. **Security level 2** $\Longrightarrow$ equivalent to SHA-256/SHA3-256 collision search. **Security level 3** $\Longrightarrow$ AES-192 key search. **Security level 4** $\Longrightarrow$ SHA-384/SHA3-384 collision search. **Security level 5** $\Longrightarrow$ AES-256 key search. For all these categories, the minimum complexity threshold is used for the complexity of all the attacks against a given PQC candidate variant. The security strengths of the algorithms are presented in Table 2.

3 High-Level Synthesis of PQC Hardware

Dedicated PQC hardware accelerators are vital for their use in practical applications. These accelerators can be either low area and low power crypto cores used in small IoT devices or low latency implementations to be used in servers [13]. There are very few hardware implementations of PQC algorithms. Existing cryptographic co-processors were re-purposed for KYBER KEM [14]. An FPGA implementation of Niederreiter Cryptosystem was reported [13]. A hardware-software co-design scheme for the hash PQC algorithm XMSS was developed [15]. In contrast, this paper is an early study of hardware implementations of 13 PQCs employing a uniform hardware implementation flow.

We use High-Level Synthesis (HLS) hardware generation and 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 rest of the design flow (i.e., logic synthesis and physical design). HLS-based design and 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 tool vendors like Mentor Graphics, and Cadence, and FPGA vendors like Xilinx [16] and Intel [17] offer HLS tools and designs flows.

HLS organizes components in a hierarchical organization, according to the sub-function organization of the C specification, to cut down the design complexity. Each hardware unit adopts the classical Finite State Machine with Data (FSMD) model [18] with two components: *controller* and *data path*. The controller orchestrate the operations in each clock cycle. The finite state machine (FSM) representing the control flow signals the data path resources based on a set of conditions. The data path includes the functional units

[1] Why these thirteen?: We concentrated on the round 1 PQC algorithms most expected to make it further through the competition. We surveyed a few PQC investigators and their comments informed this shortlist. Our surveys concerned both KEM and Signature algorithms to have a heterogeneous case study. We expected NIST to declare the round 2 algorithms on January 10, 2019 to hone this list. However, this notice is pending.

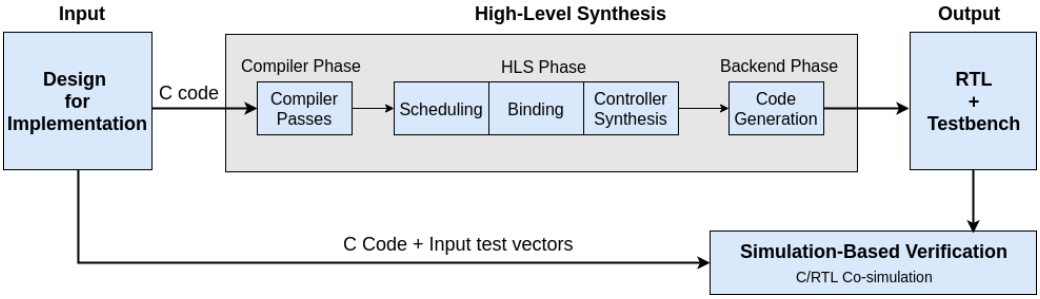

Figure 1: HLS design flow.

and registers to hold temporary values during the computation. Multiplexers drive the values based on the control flow.

HLS hardware design can be divided into three parts, as shown in Figure 1: The compiler phase (1) investigates the input C description and applies compiler-level transformations. The HLS phase (2) determines the microarchitecture. The back end (3) generates the Verilog/VHDL description and the RTL test benches. RTL simulation is performed on a set of pre-defined inputs to determine if the generated results match the golden ones obtained from the software execution.

3.1 C compiler optimizations

A typical HLS flow uses GCC or LLVM compilers to parse the input C code, applies compiler optimizations, and generates an intermediate representation (IR). HLS uses the Single Static Assignment (SSA) form so that the IR can be manipulated and turned into register transfer level (RTL) hardware by successive HLS steps. One can apply optimizations such as loop unrolling, constant propagation, and function inlining to permit downstream HLS optimizations such as extraction of instruction-level parallelism. The call graph describes the relationships between the functions and determines the components and the hierarchical interconnections between them.

3.2 HLS

Each function in the IR is transformed into an RTL hardware module. During the HLS *allocation* step, resources are selected. Next, 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 also 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 FSM. Based on the operations to execute on the microarchitecture, the FSM generates signals that route the data in the data path through the multiplexers in each clock cycle. We use Xilinx Vivado HLS as it supports C design [16]. HLS PQC hardware design flow can also use Mentor Catapult HLS [19], LegUp HLS [20], and Intel HLS [17].

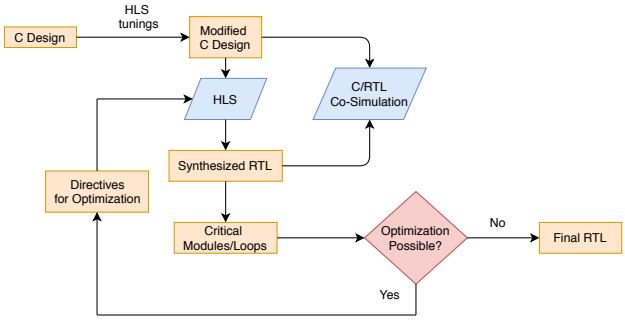

Figure 2: High-Level Synthesis design exploration of PQC algorithms.

3.3 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 is 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) [16] and the ASIC back-end flow (Synopsys Design Compiler [21]).

4 Experimental Evaluation

4.1 HLS experimental evaluation methodology

Figure 2 shows the HLS design exploration method for PQC algorithms. We modify the original C specification to make it HLS-suitable (e.g., change pointers to fixed dimension arrays, remove recursions). Next, we perform HLS on the synthesizable C code to generate RTL using Xilinx Vivado HLS. Vivado provides a detailed synthesis report indicating 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] .

4.2 PQC algorithms used in this study

We evaluated thirteen NIST PQC implementations. The PQC algorithms and their implementation and security characteristics are summarized in Table 2. In this paper, we focused on PQC encryption and decryption algorithms and synthesis of keypair generation as an important next step.

4.3 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

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

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, LUTs, and Multiplexers for FPGA implementations and chip area for ASIC implementations. There is a trade-off between speed and area. Reduction in latency often increases the total area. Hence, Latency-area product (LAP) is used to check the efficiency and resource utilization of the design. A lower LAP corresponds to a superior implementation.

4.4 Baseline hardware implementations

Tables 3 and 4 report the hardware and timing overhead for implementing the PQC encryption, decryption and keypair generation algorithms, respectively when synthesized without any additional constraints (latency). Figure 4 shows how the KEM and Signature encryption algorithms can be ordered when ranked on the basis of least latency.

Table 2: **PQC algorithms used in this study:** A high-level analysis of the 13 NIST PQC algorithm implementations. Of these, nine are KEM primitives and four are signature primitives. Of the KEM primitives, three 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. LIMA has two implementations of security level 3 and two of security level 5. The security levels of each algorithm used in this paper are shown in bold. Among these, nine are of security level 1, one is of security level 2, two are of security level 3 and two are of 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.

Algorithm	Basis hard problem	PQC Primitive	Supported Security Level (public key size in bytes)				
			1	2	3	4	5
Big Quake [22]	Code	KEM	1624896	-	**5384448**	-	9576000
Classic McEliece [23]	Code	KEM	-	-	-	-	**1047319** 1357824
NTS-KEM [24]	Code	KEM	**319488**	-	929760	-	1419704
LIMA [25]	Lattice	KEM	6109	-	6145 14577	10449	12289 **16497**
Saber [26]	Lattice	KEM	672	-	**992**	-	1312
Crystals-KYBER[27]	Lattice	KEM	**736**	-	1088	-	1440
NewHope [28]	Lattice	KEM	**928**	-	-	-	1824
FrodoKEM [29]	Lattice	KEM	**9616**	-	15632	-	-
NTRU-HRSS-KEM [30]	lattice	KEM	**9100**	-	-	-	-
RLIZARD [31]	Lattice	Signature	**4096**	-	4096 8192	-	8192
Crystals-Dilithim [32]	Lattice	Signature	**1184**	1472	1760	-	-
SPHINCS+ [33]	Hash	Signature	**32**	-	48	-	64
MQDSS [34]	Multivariate	Signature	-	**62**	-	88	-
qTESLA	Lattice	Signature	**4128**	-	8224	-	8224

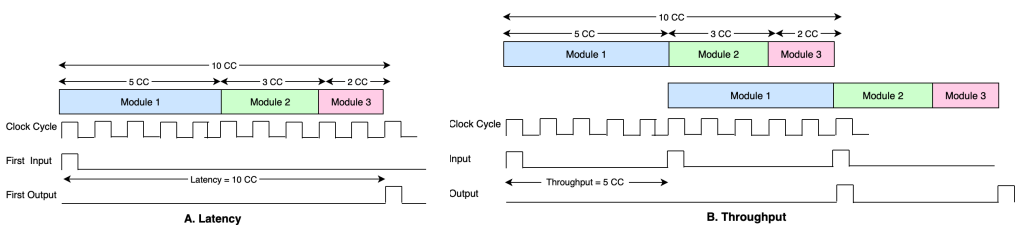

Figure 3: Illustration of Latency and Throughput metrics.

Table 3: **Description:** Security versus area versus the timing of PQC encryption algorithms, without optimizations (i.e., baseline). **Analysis:** Among the security level 1 KEM algorithms, NTRU-HRSS-KEM 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, RLIZARD and CRYSTALS-Dilithium have latencies fewer than a million cycles. **Takeaway:** MQDSS (for signature generation), and LIMA (for KEM) are good candidates for IoT devices. LIMA offers level 5 security and is the second fastest and the second smallest. MQDSS signature generation algorithm has security level 2 and is the third fastest and the third smallest.

Algorithm	Security	FF	LUT	MUX	Clock (ns)	Latency
KEM algorithms						
CRYSTALS-Kyber	1	40720	230540	1217	15	56345
Newhope	1	26257	135689	1538	15	681191
FrodoKEM	1	14516	82265	1014	10	469217
NTSKEM	1	68154	823172	15	10	3952
NTRU-HRSS-KEM	1	6633	33845	341	15	1496914
Big_Quake	3	6138	19932	909	10	8925800
Saber	3	38495	214764	1496	15	499812
Classic McEliece	5	60264	840384	898	10	5128978
LIMA	5	44678	58323	3	10	8252
Signature algorithms						
RLIZARD	1	1503	3757	2426	10	623730
CRYSTALS-Dilithium	1	25926	133461	3002	10	609828
SPHINCS+	1	8641	31147	897	10	628778326
qTELSA	1	62765	320842	5018	10	16571091
MQDSS	2	35263	193320	3853	15	49365597

Table 4: **Description:** Security versus area versus the timing of PQC decryption algorithms, without optimizations (i.e., baseline). **Analysis:** Among the KEM algorithms, NTRU-HRSS-KEM (security level 1), BIG_QUAKE (security level 3), LIMA and Classic McEliece (security level 5) have latency of more than 1 million cycles. NTSKEM is the fastest among those in security level 1. Among the Signature algorithms, RLIZARD and CRYSTALS-Dilithium have a latency less than a million cycles. CRYSTALS-Dilithium is the fastest among signature algorithms. **Takeaway:** RLIZARD (for Signature generation) and BIG_Quake (for KEM) are good candidates for IoT devices. None of the level 5 security decryption algorithms considered in this study have low latency and hence are not appropriate for servers. All the low latency algorithms only have level 1 security – NTSKEM (KEM) and CRYSTALS-Dilithium (Signature).

Algorithm	Security Level	FF	LUT	MUX	Clock (ns)	Latency
KEM- algorithms						
CRYSTALS-Kyber	1	33030	186244	953	15	53553
Newhope	1	19635	92250	1123	15	723027
FrodoKEM	1	14461	82307	1027	10	220344
NTSKEM	1	13904	83250	278	10	43425
NTRU-HRSS-KEM	1	5292	29532	440	15	1003222
Big_Quake	3	5221	10123	314	10	946887
Saber	3	33751	189597	1106	15	89392
Classic McEliece	5	70112	847949	914	10	146126996
LIMA	5	54064	66404	612	10	5376651
Signature algorithms						
RLIZARD	1	1728	5141	3041	10	621236
CRYSTALS-Dilithium	1	20865	108878	1737	10	5380
SPHINCS+	1	3335	11438	433	10	937975
qTELSA	1	39177	184931	3357	10	18555063
MQDSS	2	26423	147359	2568	15	25124450

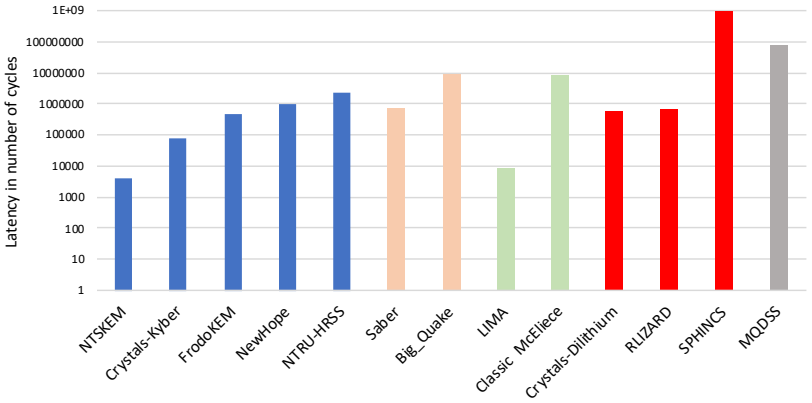

Figure 4: We sorted the PQC encryption algorithms by latency, normalized for a clock cycle of 10 ns for each security level. The KEM algorithms are in blue, orange and green, corresponding to security levels 1, 3, and 5. The Signature algorithms are in red and gray, corresponding to security levels 1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

4.5 Critical Functions

In this section, we will examine the critical functions as well as the loops in them that result in high latency for the PQC encryption algorithms. The results are obtained from the HLS reports and shown in Table 5. The critical functions are optimized using loop unrolling and loop pipelining as explained in Section 5.1.1 and Section 4.7, respectively.

4.6 Optimization 1: Loop unrolling

The latency of the 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 can minimize the latency. However, this uses more resources. Figure 5 explains loop unrolling.

Table 5: Critical functions of the PQC encryption algorithms.

Algorithm	Critical Functions	# Loops
KEM algorithms		
Big_Quake	`m2error`	1
NTRU-HRSS-KEM	`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
LIMA	`DecodePK`	1
NTSKEM	`hash`	1
Signature algorithms		
RLIZARD	`crypto_encrypt`	1
CRYSTALS-Dilithium	`expand_mat`	2
MQDSS	`crypto_sign`	2
SPHINCS+	`treehash`	5

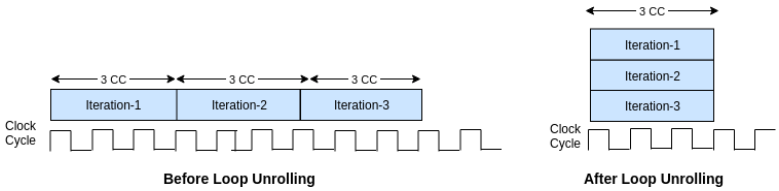

Figure 5: Loop unrolling for a *for* loop of count 3.

Vivado HLS provides the option to partially unroll the loop to balance the performance and area. We use loop unrolling for encryption and decryption and report the results in Table 6 and Table 7 respectively.

Table 6: **Description:** Security versus area versus the timing of PQC encryption algorithms, after loop unrolling. **Analysis:** Compared to Table 3, virtually all algorithms (except NTSKEM) 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. BIG_QUAKE has the most speedup in terms of latency (66× reduction in latency compared to baseline (in Table 3). **Takeaway:** Loop unrolling reduces the latency of all PQC encryption algorithms. However, it also results in an increase in area. LIMA is still an ideal candidate for application in servers, since it provides high security (level 5) and has low latency. Apart from LIMA, 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 (KEM), RLIZARD and SPHINCS+ (Signature) can be used for IoT devices, since the area overhead is low. NTSKEM (KEM , security level 1) can be used in servers owing to its low latency.

Algorithm	Security Level	FF	LUT	MUX	Clock (ns)	Latency
KEM algorithms						
CRYSTALS-Kyber	1	237182	2414748	1358	15	42823
Newhope	1	26257	135689	1538	15	680150
FrodoKEM	1	44284	136998	12422	10	366609
NTSKEM	1	68154	823172	15	10	3952
NTRU-HRSS-KEM	1	9035	65356	2263	15	22594
Big_Quake	3	249798	743560	913	10	286355
Saber	3	93234	376313	1233	15	236812
Classic McEliece	5	69795	934492	909	10	2373772
LIMA	5	79101	93217	59	10	5751
Signature algorithms						
RLIZARD	1	1503	3757	2426	10	268056
CRYSTALS-Dilithium	1	158313	584742	4880	10	18525
SPHINCS+	1	8931	42604	975	10	464961626
MQDSS	2	45135	230273	25958	15	49365597

The ranking of the encryption functions after loop unrolling is shown in Figure 7 (a). Among the high security algorithms, LIMA (KEM , security level 5) has the lowest latency. Among the low security algorithms, NTSKEM is the fastest among the KEM algorithms and Crystals-Dilithium is the fastest among the Signature algorithms.

4.7 Optimization 2: Loop Pipelining

Loop pipelining can be used to improve latency. Figure 6 explains loop pipelining. This optimizes both hardware and latency. We synthesize the algorithms by adding a directive to pipeline the longer loops. The results for encryption is presented in Table 8.

The ranking of the encryption functions after pipelining is shown in Figure 7 (b). Similar

Table 7: **Description:** Security versus area versus the timing of PQC decryption 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-KEM. **Takeaway:** Similar to encryption, loop unrolling reduces latency for PQC decryptions. 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 and LIMA (KEM) can be used, since the area overhead is low. For IoT devices where high level of security is not required, security level 1 algorithms with low area overhead like RLIZARD and 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.

Algorithm	Security Level	FF	LUT	MUX	Clock (ns)	Latency
KEM algorithms						
CRYSTALS-Kyber	1	194126	1977896	1028	15	43018
Newhope	1	28999	164937	1123	15	721986
FrodoKEM	1	97355	128031	741	10	117736
NTSKEM	1	99417	951249	15	10	49719
NTRU-HRSS-KEM	1	11514	97791	2439	15	21996
Big_Quake	3	247036	734805	995	10	237847
Saber	3	231549	2350000	998	15	365015
Classic McEliece	5	79962	870908	914	10	10659024
LIMA	5	46934	53176	3	10	76748
Signature algorithms						
RLIZARD	1	1728	5141	3041	10	267854
CRYSTALS-Dilithium	1	108154	388991	15309	10	5380
SPHINCS+	1	3335	11438	433	10	937975
MQDSS	2	93945	323734	48534	15	25084906

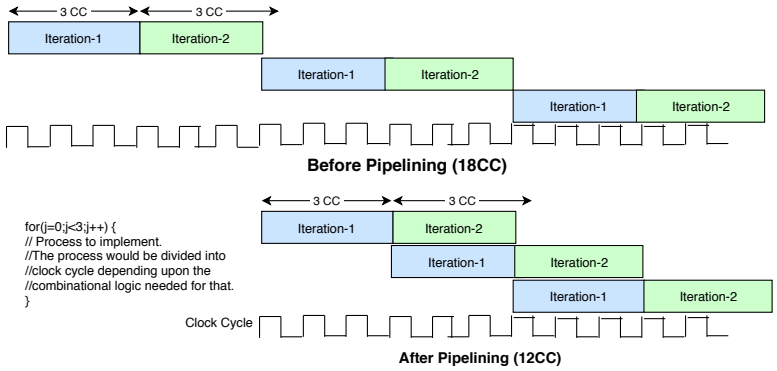

Figure 6: Example of loop pipelining.

to Figure 7 (a), LIMA provides the best latency among the high security algorithms. Among the algorithms with security level 1, NTSKEM is the fastest among the KEM algorithms and Crystals-Dilithium is the fastest among the Signature algorithms.

4.8 Latency-area product Comparisons

In this section, we compare the latency-area product (LAP) for the encryption algorithms for both the baseline and the optimization techniques. Similar to [35], we consider the area as the number of FPGA LUTs required to synthesize the design. The results are shown in Table 9. A lower LAP corresponds to better performance in terms of latency. As can be seen in Table 9, the number of algorithms for which the lowest LAP is obtained using unrolling is almost similar to that of pipelining. This is because of the many loop dependencies in some algorithms like CRYSTALS-Dilithium, which restricts the speedup

Table 8: **Description:** Security versus area versus the timing of PQC encryption algorithms, after loop pipelining. **Analysis:** Similar to loop unrolling, pipelining also reduces the overall latency for the PQC encryption algorithms. Among the KEM algorithms, only Classic McEliece has a latency of more than 1 million cycles. The major difference with Table 6 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 encryption 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 (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 three of the 10 algorithms. On the other hand, for signature algorithms, pipelining provides better latency for two of the four algorithms compared to unrolling.

Algorithm	Security Level	FF	LUT	MUX	Clock (ns)	Latency
KEM algorithms						
CRYSTALS-Kyber	1	11699	1307815	1076	15	31669
Newhope	1	25639	136457	1552	15	307847
FrodoKEM	1	105875	179290	1495	10	335891
NTSKEM	1	68154	823172	15	10	3952
NTRU-HRSS-KEM	1	12225	75141	341	15	100208
Big_Quake	3	78567	540165	290	10	42366
Saber	3	40824	234171	1394	15	367099
Classic McEliece	5	60270	840430	898	10	3787729
LIMA	5	29464	48016	59	10	8259
Signature algorithms						
RLIZARD	1	881	4253	2256	10	267386
CRYSTALS-Dilithium	1	146076	1327355	2274	10	155166
SPHINCS+	1	20628	66750	930	10	468789803
MQDSS	2	47441	270713	3882	15	25825918

due to pipelining. The PQC algorithms are ranked in terms of LAP in Figure 8.

4.9 Security level vs hardware tradeoffs

The PQC algorithms have different implementations depending on the intended security strength (i.e. the implementations vary in key sizes). In this section, we examine how the hardware overhead varies with security strength. We run experiments on a baseline implementation, i.e., with no optimizations. Figure 9 plots the hardware overhead of PQC algorithms in number of flip-flops and LUTs. Figure 10 reports the hardware overhead in terms of the number of flip-flops and LUTs, for the PQC decryption algorithms. The number of multiplexers don't change for the different implementations.

4.10 ASIC Implementation of PQC decryption algorithms.

In this section, we report the ASIC implementations of PQC decryption 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 10. 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.

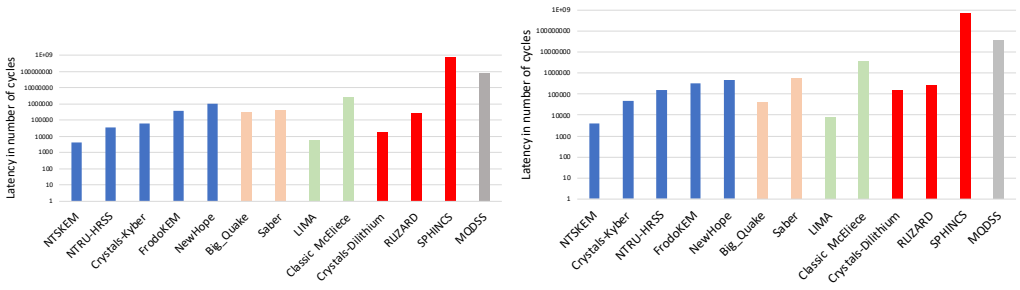

Figure 7: **Description:** PQC encryption algorithms sorted by latency, normalized for the clock cycle of 10 ns. The KEM algorithms are in blue, orange, and green, corresponding to security levels 1, 3, and 5. The signature algorithms are in red and gray, corresponding to security levels 1 and 2. **Analysis:** (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 has security level 1 and two have security level 3) and two KEM algorithms have latency over a million cycles (one has security level 1 and the other has security level 5). Two signature algorithms have latency fewer than a million cycles (both have security level 1) and two have latency higher than a million cycles (one has security level 1 and the other has security level 2). (b) Loop pipelining: Four KEMs have latency less than 100,000 cycles (three have security level 1, one has security level 3 and one has security level 5). Four KEMs have latency less than a million cycles (three have security level 1 and one has security 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 have security level 1) and two higher than a million cycles (one has security level 1 and the other has security level 2.)

Table 9: **Description:** Security vs. latency-area product (LAP) for various optimizations on the PQC encryption algorithms. The minimum values for each algorithm are written in bold. **Analysis:** For most KEM algorithms, pipelining produces the best latency-area product. Only for two (NTRU-HRSS-KEM 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:** LIMA (KEM) has a low LAP value along with high security (level 5). Among the low security alternatives (level 1), NTSKEM (KEM) and RLIZARD (Signature) have low LAPs.

Algorithm	Security Level	Baseline	Loop Unrolling	Loop Pipeline
KEM algorithms				
CRYSTALS-Kyber	1	12989776300	103406753604	**41417193235**
Newhope	1	92430125599	92288873350	**42007878079**
FrodoKEM	1	**38600136505**	50224699782	60221897390
NTSKEM	1	3253175744	3253175744	3253175744
NTRU-HRSS-KEM	1	50663054330	**1476653464**	7529729328
Big_Quake	3	177909045600	21291840600	**22884630390**
Saber	3	107341624368	89115434156	**85963939929**
Classic McEliece	5	10071094989153	**2218270943824**	3183321083470
LIMA	5	481281396	4081151648	**3965641443**
Signature algorithms				
RLIZARD	1	2343353610	1382786392	**1137192658**
CRYSTALS-Dilithium	1	81388254708	**1067154150**	205960365930
SPHINCS+	1	**19584558519922**	19809225114104	31291719350250
MQDSS	2	9543357212040	11367564117981	**6991411739534**

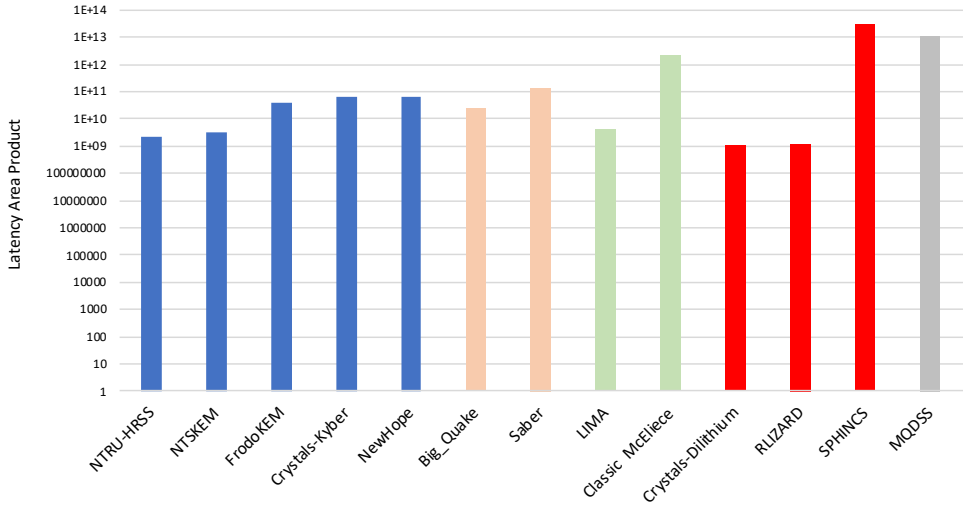

Figure 8: PQC encryption algorithms sorted by LAP, normalized for clock cycle of 10ns. The KEM algorithms are in blue, orange and green, corresponding to security levels 1, 3, and 5. Signature algorithms are in red and gray, corresponding to security levels 1 and 2. Among the KEM algorithms, three have a LAP of less than 10^{10} – three of security level 1 and one of security level 5. Five are in the range $10^{10} - -10^{12}$, three of security level 1 and two of security level 3. One security level 5 algorithm has a LAP of over 10^{12}. Among signature algorithms, two have LAP $< 10^{10}$ – both are of security level 1. Two others – one security level 1 and another of security level 2 have LAP $> 10^{12}$.

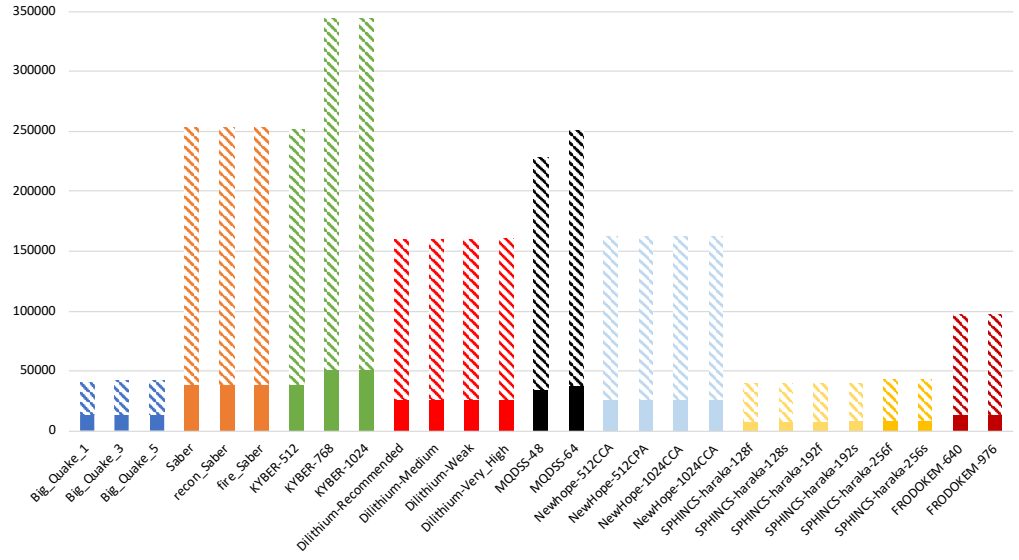

Figure 9: Hardware overhead in terms of number of LUTs and FFs for various implementations of PQC encryption algorithms. FFs are shown in solid colors and LUTs as hashed patterns. Implementations of the same algorithm are marked with the same color for ease of comparison. KYBER encryption is the largest and has the maximum variation in area relative to security levels. KYBER-768 requires 40% more hardware over KYBER-512.

5 Two PQC design exploration case studies

We will study two cases to explain the design space exploration. This entails analyzing functions which determine the minimum achievable latency used in PQC encryption algorithms. Then we will show that loop unrolling and pipelining can improve latency.

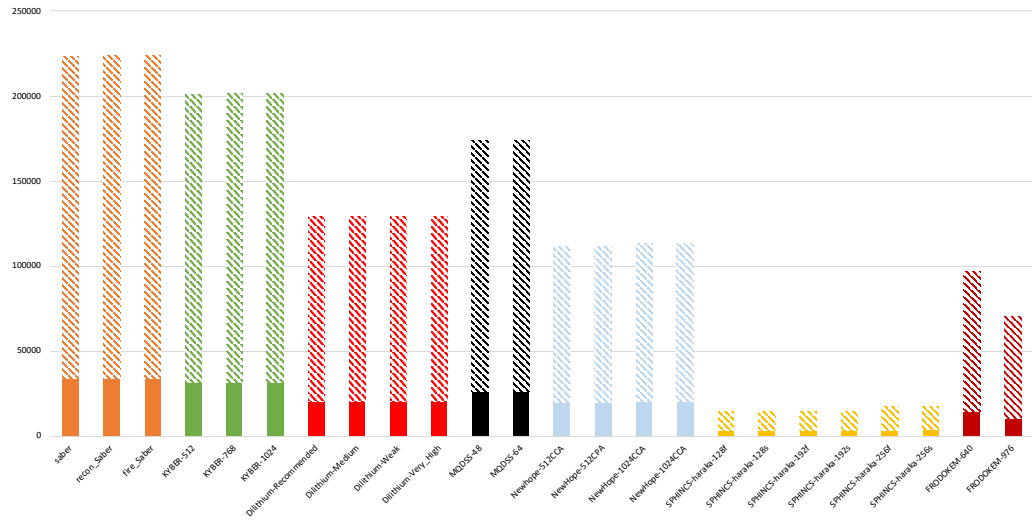

Figure 10: # of flip-flops (FF) and LUTs used by implementations of PQC decryption algorithms. FFs are shown in solid colors, while LUTs as hashed patterns. Implementations of the same algorithm have the same color. Except FRODO-KEM, none of the decryption algorithms have noticeable difference in FF/LUT count. The variation in LUT and FFs for the two security levels of Frodo-KEM is ~40%. Saber uses the most LUTs and FFs.

Table 10: **Description:** ASIC synthesis of some of the studied PQC decryption algorithms. **Analysis:** FrodoKEM (security level 1, KEM) and SPHINCS+ (security level 1, Signature) have small decryption 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	Security 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
NTS-KEM	1	156	3163206	1255	15.29
NTRU-HRSS-KEM	1	169.5	1246869	495	14.3
Big_Quake	3	200	40543	16	28.2
Saber	3	137.75	4774529	1895	54.49
LIMA	5	123.45	1474598	585.158	105.17
Signature algorithms					
RLIZARD	1	129.7	1701653	675.259	23.22
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

5.1 Case study 1: Design exploration of the BIG_QUAKE

Consider BIG_QUAKE code encryption algorithm. An analysis of BIG_QUAKE's encryption function `crypto_kem_enc()` reveals that the function `m2error()` (in file m2e.c) limits the latency as follows:

The latency for BIG_QUAKE encryption (`crypto_kem_enc()`) without optimization is 8925800 cycles of which `m2error()` contributed 8733488 cycles. Table 11 summarizes the latencies of the functions in `crypto_kem_enc()` and the important loops in `m2error()`. Function `m2error()` has three loops. Loop 1 and Loop 3 are simple, single-line loops that repeat a single operation without invoking any functions. Loop 2, on the other hand, invokes four functions, `uniform_m2e, swap_m2e, memcpy,` and `init_hash`. In turn, `uniform_m2e` calls `hash_trunc` and `ucharToInt`. `swap_m2e` has no loops and function invocations. `init_hash` calls `Keccak` hash function, which calls `KeccakF1600_StatePermute` which has eight independent loops that do not call other functions. Overall, the call graph for `m2error()` is shown in Figure 11 and the latency of each loop is shown in Table 11.

```
    int m2error(IN unsigned char *m, OUT int * error) {
    int i, j, s = 3, permutation[LENGTH];
    unsigned char * aux = malloc(HASH_SIZE*sizeof(unsigned char));
    for (i = 0; i < LENGTH; ++i)  /* Loop 1  */
        permutation[i] = i;
    init_hash(m);
    for(i = 0; i < NB_ERRORS; i++) {   /* Loop 2   */
        j = uniform_m2e(s, LENGTH - i-1);
        swap_m2e(permutation, i, i + j);
        memcpy(aux, buff, HASH_SIZE);
        init_hash(aux);
    }
    for (i = 0; i < NB_ERRORS; ++i) /* Loop 3  */
        error[i] = permutation[i];
    free(aux);
    return SUCCESS;
}
```

In order to apply loop unrolling, we modified the code to resolve the "if" conditions with the bottleneck in this case being `hash_trunc`. `buff_size` is a global variable set to `HASH_SIZE` (=32) by `init_hash`. Every call to `hash_trunc` reduces it by `s` (=3). Since

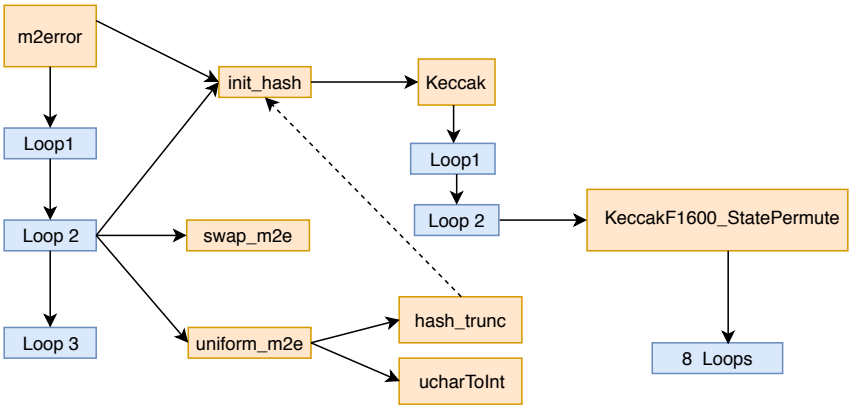

Figure 11: Call graph of `m2error()`.

`init_hash` is conditionally called by `hash_trunc`, the connection between the two is shown by a dotted line in the call graph. Therefore, in order to reduce the latency due to `m2error()`, it is necessary to optimize these loops.

```
unsigned char * hash_trunc(int s) {
unsigned char* aux = malloc(HASH_SIZE*sizeof(unsigned char));
buff_size -= s;
if (buff_size < 0) {
    memcpy(aux, buff, HASH_SIZE);
    init_hash(aux);
    buff_size -= s;
    return realloc(aux, s*sizeof(unsigned char));
}
memcpy(aux, buff + buff_size - 1, s*sizeof(unsigned char));
return aux;
}
```

5.1.1 Loop Unrolling

We unroll the three loops in `m2error()`, the two loops in `Keccak` and the eight loops in `KeccakF1600_StatePermute`. We set the unrolling factor to 1 to direct HLS to do all operations in a single cycle. We inline function calls and replace `memcpy` by loops[3].

5.1.2 Loop Pipelining

We pipeline the three loops of `m2error()` and set the target II as 1. We also partition the variable *state* in `KeccakF1600_StatePermute` to obtain minimum latency. The results for all experiments are shown in Table 12. We synthesized the function `m2error()`. The last column in Table 12 indicates the speedup obtained compared to the baseline implementation. Loop unrolling and loop pipelining yield significant speedups. Further, loop pipelining offers a 11× speedup and 27% less hardware compared to loop unrolling.

5.2 Case study 2: Design exploration of CRYSTALS_KYBER

In this section, we will perform design exploration of CRYSTALS_KYBER, an example lattice cryptography algorithm. An analysis of the encryption function (`crypto_kem_enc()`) reveals that the function `indcpa_enc()` limits the latency. An analysis of `indcpa_enc()` shows that `gen_matrix()` is the function with the highest latency.

```
void gen_matrix(polyvec *a, const unsigned char *seed, int transposed) {
  unsigned int pos=0, ctr, nblocks=4, i, j;
```

[3]Vivado HLS does not efficiently map `memcpy` into an equivalent hardware structure.

Table 11: Critical functions and loops that limit the BIG_QUAKE latency.

Function/Loop	Latency (clock cycles)
Top function: crypto_kem_enc	
randombytes_quake_en	39829
m2error	8733488
KeccakF1600_StatePer	28513
encrypt_nied	1294
Top function: m2error	
m2error_loop 1	7410
m2error_loop 2	8668864
m2error_loop 3	182

Table 12: Area and timing overhead for different implementations of `m2error()`.

Implementation	FF	LUT	MUX	Latency	Speedup
Baseline	1677	5141	329	8733488	-
Loop Unrolling	244875	732218	2192	93739	93.16 ×
Loop Pipelining	75845	531022	149	8135	1073.57 ×

```
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;
      }
     }
    }
   }
  }
```

Table 13 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, in turn, 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 other 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 overall function call graph for `gen_matrix()` is shown in Figure 12.

Table 13: Critical functions and loops that determine the latency of CRYSTALS-KYBER.

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

5.2.1 Loop Unrolling

We unroll the loops in the last level functions, `store64()`, `load64()` and `KeccakF1600_StatePermute()`. We unroll innermost loops in the top level functions, i.e., Loop 2 in `keccak_squeezeblocks()` and Loop 4 in `gen_matrix()`. The unrolling factor is set as 1 and added as a directive before synthesis to inline all function calls in the loops.

5.2.2 Loop Pipelining

We mark the unrolled loops for pipelining. We set the target II for pipelining as 1, except for the loop in `KeccakF1600_StatePermute()`. Since `KeccakF1600_StatePermute()` does a lot of operations in a single loop, we created the fastest possible pipeline architecture. The Table 14 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.

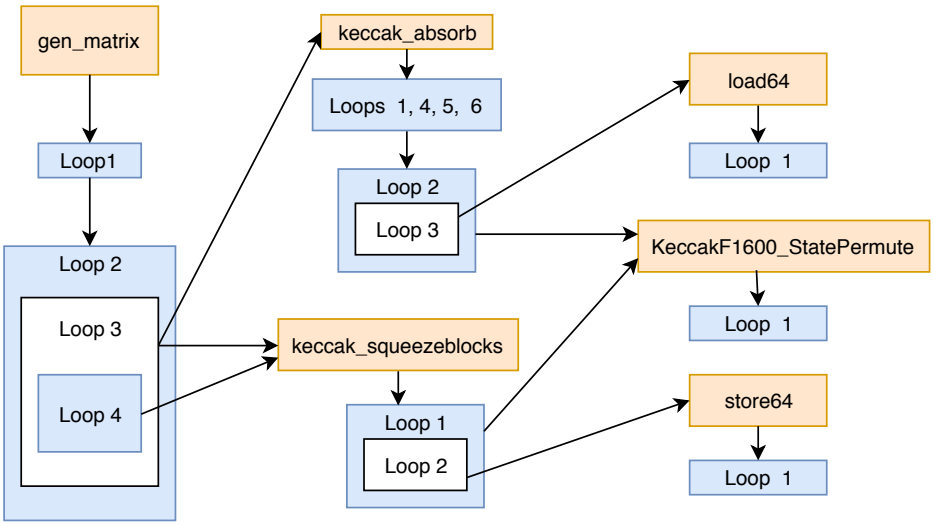

Figure 12: Call graph of `gen_matrix()`.

Table 14: Area and timing overhead for different implementations of `gen_matrix()`.

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 ×

6 Key takeaways from this study

In this paper, we have developed hardware implementations of several PQC algorithms. The key results from this preliminary study are:

1. Among the KEM algorithms of security level 1, NTSKEM is superior in terms of latency and LAP.
2. Among the Signature algorithms of security level 1, CRYSTALS-Dilithium is superior and 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. Notwithstanding this, LIMA is a good candidate for high (level 5) security, with low latency and low LAP and BIG_QUAKE 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, Big_Quake, NTRU-HRSS and RLIZARD FPGAs are ideal for IoT devices with security level 1. Low-latency NTS-KEM and Crystals-KYBER FPGA implementations are good for servers with security level 1.
5. FrodoKEM and SPHINCS+ ASICs have small decryption modules which spend the slightest power and hence helpful in small IoT devices, if a low security level of 1 suffices. NTRU-HRSS-KEM and NewHope (security level 1) are the fastest ASICs and thus are suitable in 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) KEM algorithms, latency rankings change when optimized.
8. PQC hardware implementations are not optimized for side-channel resistance. PQC researchers can use these implementations for further hardware security analysis such as the side-channel analysis.
9. A successful ongoing exercise involves an analysis of energy-area-performance trade-offs of more NIST PQC algorithms spanning all security levels.

7 Acknowledgements

- Profs. Gerardo Pelosi (Politecnico di Milano), Debdeep Mukhopadhyay (IIT Kharagpur) and Francesco Regazzoni (ALARI Switzerland) helped with picking the top PQC candidates for this hardware implementation study.
- Dr. Marc Manzano and Dr. Najwa Aaraj of Dark Matter inc. Abu Dhabi, UAE offered timely and insightful feedback (especially to explore the security-informed trade-offs) on the early drafts of the report.
- Profs. Alessandro Barenghi (Politecnico di Milano) and Xianhui Lu answered our questions while implementing LEDAcrypt and LAC, respectively.

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

[5] 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.

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] W. Wang, B. Jungk, J. Wälde, S. Deng, N. Gupta, J. Szefer, and R. Niederhagen, "Xmss and embedded systems - xmss hardware accelerators for risc-v." Cryptology ePrint Archive, Report 2018/1225, 2018. https://eprint.iacr.org/2018/1225.

[16] 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].

[17] 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.

[18] 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.

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

[20] 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.

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

[22] M. Bardet, E. Barelli, O. Blazy, R. C. Torres, A. Couvreur, P. Gaborit, A. Otmani, N. Sendrier, and T. Jean-Pierre, "BIG QUAKE binary goppa quasi–cyclic key encapsulation." https://hal.archives-ouvertes.fr/hal-01671866/document, 2017.

[23] 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.

[24] M. Albrecht, C. Cid, K. G. Paterson, C. J. Tjhai, and M. Tomlinson, "NTS-KEM." https://nts-kem.io/, 2018.

[25] N. Smart, M. Albrecht, Y. Lindell, E. Orsini, V. Osheter, K. Peterson, and G. Peer, "LIMA-1.1 : A PQC encryption scheme." https://lima-pq.github.io/, 2017.

[26] 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.

[27] 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.

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

[29] 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.

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

[31] T. Park, H. Seo, J. Kim, H. Park, and H. Kim, "Efficient parallel implementation of matrix multiplication for lattice-based cryptography on modern arm processor," *Security and Communication Networks*, 2018.

[32] 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.

[33] 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.

[34] 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.

[35] 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.