A High-performance Hardware Implementation of Saber Based on Karatsuba Algorithm

Yihong Zhu[1], Min Zhu[2], Bohan Yang[1], Wenping Zhu[1], Chenchen Deng[1], Chen Chen[1], Shaojun Wei[1] and Leibo Liu[1]

[1] Tsinghua University, China. zhuyihon18@mails.tsinghua.edu.cn
[2] Wuxi Micro Innovation Integrated Circuit Design Co.Ltd.

Abstract. Although large numbers of hardware and software implementations have been proposed to accelerate lattice-based cryptography, Saber, a module-LWR-based algorithm, which has advanced to second round of the NIST standardization process, has not been adequately supported by the current solutions. Based on these motivations, a high-performance crypto-processor is proposed based on an algorithm-hardware co-design in this paper. First, a hierarchical Karatsuba calculating framework, a hardware-efficient Karatsuba scheduling strategy and an optimized circuit structure are utilized to enable high-throughput polynomial multiplication. Furthermore, a task-level pipeline and truncated multipliers are proposed to enable algorithm-specific fine-grained processing. Enabled by all of the above optimizations, our processor takes 943, 1156, and 408 clock cycles for key generation, encryption, and decryption, respectively. Enabled by these optimizations, our processor takes 943, 1156 and 408 clock cycles for key generation, encryption, and decryption of Saber768, achieving 5.4×, 5.2× and 4.2× reductions compared with the state-of-the-art FPGA solutions, respectively. The post-layout simulation of our design is implemented with TSMC 40 nm CMOS process within 0.35 mm^2. The throughput for Saber768 is up to 346k encryption operations per second and the energy efficiency is 0.12 uJ/encryption while operating at 400 MHz, achieving nearly 52× improvement and 30× improvement, respectively compared with current PQC hardware solutions.

Keywords: lattice · Module-LWR · Saber · ASIC · high-throughput · Karatsuba · hierarchical calculation framework

1 Introduction

As the only module-learning with rounding (LWR)-based candidate, Saber has been optimized on the Intel Xeon processors [DKSRV18, Roy19], ARM Cortex-series [KBMSRV18] and FPGA platforms [Far19, RB20, MTK$^+$20]. For software implementations, a hybrid method combining Toom-Cook and Karatsuba algorithm is utilized for efficient implementations of polynomial multiplications [DKSRV18, Roy19, KBMSRV18]. For hardware implementations, simple but efficient schoolbook multiplication is utilized in [RB20, Far19]. A 4-way Toom-Cook method is utilized to reduce 43% the number of multiplication operations [MTK$^+$20]. However, the efficiency of implementing fast-multiplication algorithm to hardware of Saber can be further optimized.

An high-performance crypto-processor is proposed based on the algorithm-hardware co-design of module-LWR with multiple security levels. The main computational framework is an 8-level recursive splitting hierarchical Karatsuba framework, which reduces the degree-256 polynomial multiplication to the coefficient multiplication. Several optimization strategies, including optimized pre-/post-process structure, the task-rescheduling-based pipeline and truncated multiplier, are developed to further improve the energy efficiency. Our processor has been implemented on Xilinx Virtex UltraScale+ FPGA platform and post-layout simulation on TSMC 40nm process. The key contributions of this work are summarized as follows:

1. Hierarchical Karatsuba framework is utilized to accelerate the degree-256 polynomial multiplication in Saber. Compared with the schoolbook multiplication, more than 90% multiplication operations are saved through 8-level hierarchical Karatsuba framework.

2. Sequential hardware-efficient Karatsuba scheduling for post-processing and compact input pre-processing are proposed for the hierarchical Karatsuba framework. Optimization strategies of hierarchical Karatsuba framework, including post-processing and pre-processing circuits, reduce 90.5% registers and 96.0% adders compared with the existing partial-reusage solutions.

3. Task-rescheduling based pipelining strategy and truncated multipliers empower our design to achieve lower latency with small area. From implementation aspect, pipelining strategy and truncated multipliers empower our design a speed-up of nearly 75$\times$ with 42.0% less resource usage compared with [XHY$^+$20].

4. Multi-parameter components and core modules resuage enable our design configurability among three security levels to meet different security scenarios.

2 Karatsuba-based Polynomial Multiplication

2.1 Hierarchical Karatsuba Framework

The Karatsuba algorithm consists of three phases: Pre-Add, multiplication and Post-Add. Karatsuba systolic multiplier array [LHJL05, Meh08, XJHM12, XMM15] utilized three-stage additions and multiplications to support the Karatsuba algorithm in hardware. However, the calculation scale of polynomial multiplication in Saber is much larger than ECC or RSA. Applying Karatsuba algorithm in multiple dimensions is an efficient method to reuse a relatively small Karatsuba systolic multipliers array [WBW19]. A hierarchical Karatsuba framework is used in [WBW19] to accelerate the large-number multiplication, including the kernel hardware and the scheduling strategy. Figure 1 depicts a fully-unfolded hierarchical Karatsuba framework, which illustrates an example of executing degree-n polynomial multiplication ($M \times N$) using a kernel hardware and one level of scheduling strategy. The kernel hardware was a Karatsuba multiplier array that is able to execute degree-(n/2) polynomial multiplication at one call. The scheduling structure included pre-process and post-process circuits, before and after the kernel, corresponding to the Pre-Add and Post-Add phases in the scheduling layer, respectively. The scheduling strategy was a specific algorithm that follows a finite-state machine to schedule the kernel hardware as the pseudo codes in Figure 1. The pseudo codes in this Figure obeys to one level of Karatsuba algorithm and the scheduling strategy can be extended to obey to the Karatsuba algorithm with multiple levels. The work in [WBW19] utilized a 2-level hierarchical Karatsuba framework similar as fully-unfolded structure. Similarly, this work did not consider the module reusage of adders and registers in the input. Differently, there is one layer of registers reusage implemented in the output.

Hierarchical Karatsuba calculating framework is divided into two layers: kernel layer and scheduling layer. How to balance the weights of two layers is an important topic for the implementation of Saber. The main computational task in Saber is a degree-256 polynomial multiplication. 8 levels of Karatsuba algorithm are applied in our processor to convert the degree-256 polynomial multiplication to the coefficient multiplication, reducing from 65536 multiplication operations to 6521 operations. Up to 90% multiplication operations are saved. To achieve a better trade-off between latency and area, 4 levels of Karatsuba algorithm are arranged in kernel layer and another 4 levels are arranged in scheduling layer. In other words, 81 multipliers and additional adders form

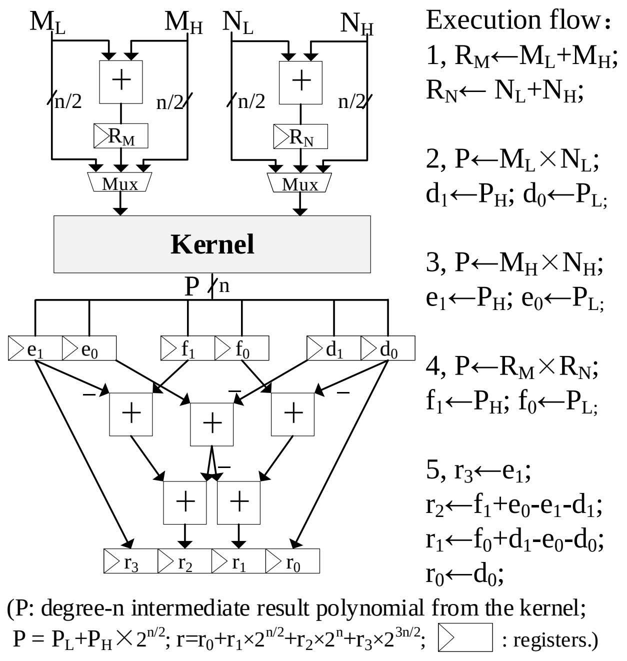

M_L M_H N_L N_H
n/2
R_M
R_N
Mux
Kernel
P n
e1 e0 f1 f0 d1 d0
r3 r2 r1 r0
Execution flow:
1, R_M←M_L+M_H; R_N← N_L+N_H;
2, P←M_L×N_L; d1←P_H; d0←P_L;
3, P←M_H×N_H; e1←P_H; e0←P_L;
4, P←R_M×R_N; f1←P_H; f0←P_L;
5, r3←e1; r2←f1+e0-e1-d1; r1←f0+d1-e0-d0; r0←d0;
(P: degree-n intermediate result polynomial from the kernel;
P = P_L+P_H×2^{n/2}; r=r0+r1×2^{n/2}+r2×2^n+r3×2^{3n/2}; : registers.)

Figure 1: Hierarchical Karatsuba hardware circuits calculating $r = M \times N$.

the kernel hardware. Kernel layer is able to process degree-16 polynomial multiplication at one call. Scheduling layer needs to transform from the required job, namely degree-256 polynomial multiplications, to the processing ability of the kernel hardware, namely degree-16 polynomial multiplications, in Karatsuba way.

2.2 Sequential Hardware-Efficient Karatsuba Scheduling

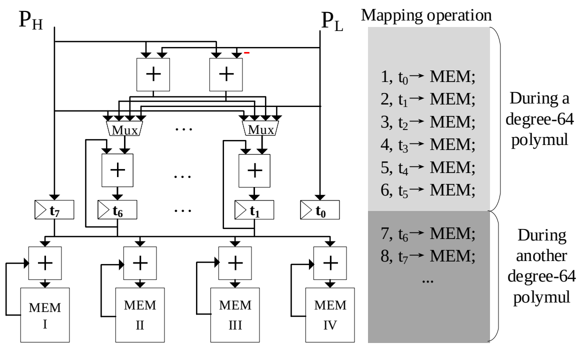
In the hierarchical Karatsuba framework, post-process structure is used to temporarily store the multiplication results to support Post-Add stage in the scheduling layer. Sequential hardware-efficient Karatsuba scheduling (SHEKS) strategy is proposed to optimize this overhead.

The main goal of SHEKS is to allow each multiplication in the Karatsuba algorithm to completely affects the final results without additional registers. Some final results are influenced by additional multiplications. This is due to the effect of Post-Add stage of Karatsuba algorithm in scheduling layer. If all the addresses in the results of each multiplication are already preassigned and allow each multiplication result to spread to all affected locations, then additional registers are no longer needed. The idea of SHEKS can be extended to more levels of Karatsuba algorithm in scheduling layer.

For the implementation of Saber, 4 levels of Karatsuba algorithm is utilized in scheduling layer and the number of adders in a direct application of SHEKS is a little higher. Moreover, a subtraction polynomial operation on the final results is needed because there is a modular polynomial $x^n + 1$, more adders are needed. So a new layer of registers is inserted in our processor to temporarily store the degree-64 subpolynomial multiplication results and the values are then mapped to the final memory one by one. The structure is shown in Figure 2.

2.3 Compact Input Pre-Processing

A compact input pre-processing technique is utilized in our processor to reduce the number of registers and adders required in pre-processing of scheduling layer. The optimization made in this



(t: temporal registers set storing the degree-128 polynomial results of degree-64 polymul. t_i: degree-16 subpolynomial; $t = t_0+t_1\times 2^d+t_2\times 2^{2d}+\ldots+t_7\times 2^{7d}$)

Figure 2: The output side of scheduling layer of Saber with 4-level Karatsuba algorithm.

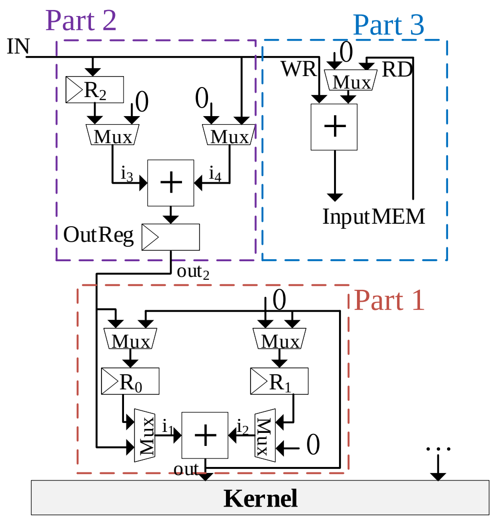


Figure 3: Optimized pre-processing design of Saber with 4-level Karatsuba algorithm.

paper is to reuse the registers and adders. Based on these observations, storing some inputs is enough to reduce the memory accesses and eliminate additional latency of the reading operations. Moreover, the execution order of the multiplications is reorganized to maximize the reusability of data stored in the input registers. The pre-processing for the two polynomial multiplication operands is identical.

For the implementation of Saber, the task of the pre-processing structure is to convert the multiplication operations with degree-256 polynomials to the operations with degree-16 polynomials that the kernel hardware is able to process. The 4-level pre-processing structure is shown in Figure 3. Hardware of Part 2 and Part 3 are used to convert polynomial operations in input side from degree-128 to degree-64 and from degree-256 to degree-128 in Karatsuba way, respectively. Hardware of Part 3 executes the summation operation of the first degree-128 polynomial and second degree-128 polynomial during the execution of Part 1 and Part 2. Then Part 3 writes the sum polynomial calculated to an additional input memory.

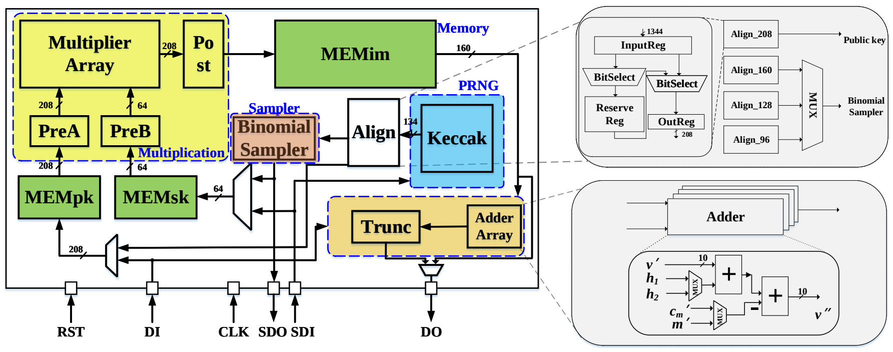


Figure 4: The system architecture of our design crypto-processor.

3 Hardware Architecture

3.1 System Architecture

Figure 4 shows the system architecture of our design. Public-key-related data pass through the DI and DO ports. Seed and secret-key-related data pass through the SDI port and SDO ports. Public matrix is generated from Keccak module marked in blue and imported into memory marked in green after alignment. Secret vector is generated from sampler marked in orange and imported into input memory. Polynomial multiplications are executed in multiplication part marked in yellow and the results are exported to output memory marked in green. Before output, the results needs addition operations in adder array and bits truncation operations in Trunc part.

To achieve a configurable design among the different versions and different stages of Saber, many components in Figure 4 adopt the idea of multi-parameter design. The data alignment modules are illustrated in the upper right corner of Figure 4. The four data aligning modules execute different types of data aligning jobs. At each cycle, BitSelect part chooses the corresponding bits from InputReg and ReserveReg part to write to ReserveReg and OutReg. The truncation modules adopt the same idea of multi-parameter. Multiple truncation modules execute different truncation jobs to output different numbers of bits.

3.2 Task-rescheduling-based pipeline design

Some pipeline tricks are added to reduce the time overheads of data importing before polynomial multiplication and data exporting after polynomial multiplication as much as possible. Figure 5 shows the circuit design and execution flow of our design. The loading and multiplication order in our design is similar as [RB20], but the generation order and the corresponding hardware components are different. The number after MEM in Figure 5 denotes the bank index of the MEM. Martix-vector and vector-vector polynomial multiplications are both involved in encryption stage. Vector-vector multiplication is scheduled before matrix-vector multiplication in our design to avoid the additional timing overhead of loading the vector of the secret key. As shown at the right corner of Figure 5, the multiplication $A_1 \times B_1$ during vector-vector polynomial multiplication is started as long as parts of the operands A_1 and B_1 have been loaded into the MEM. For key generation, public matrix is generated in column-major order and it is inconsistent with the matrix-vector multiplication order. So all four result polynomials of FireSaber needs to be stored in MEMim temporarily during matrix-vector multiplication in key generation.

While one polynomial of the public key is used in polynomial multiplication, the next polynomial is imported to another bank of MEMpk. This reduces the data importing time of multiple polynomials and the multiplication hardware keeps running once activated as shown at the bottom of Figure 5. The same holds for the reduction in data exporting time.

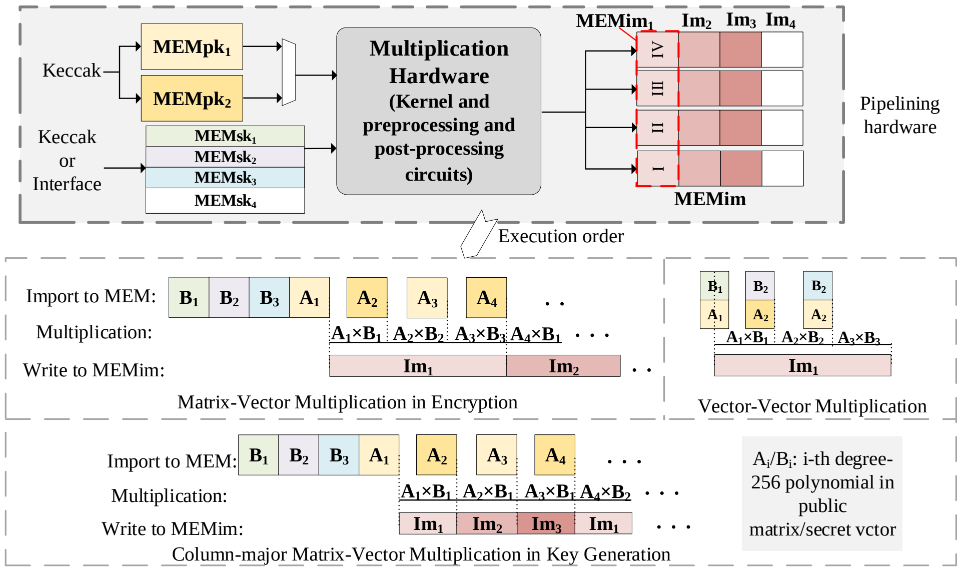

Keccak
MEMpk1
MEMpk2
Keccak or Interface
MEMsk1
MEMsk2
MEMsk3
MEMsk4
Multiplication Hardware (Kernel and preprocessing and post-processing circuits)
MEMim1 Im2 Im3 Im4
IV
III
II
I
MEMim
Pipelining hardware
Execution order
Import to MEM: B1 B2 B3 A1 A2 A3 A4
Multiplication: A1×B1 A2×B2 A3×B3 A4×B1
Write to MEMim: Im1 Im2
Matrix-Vector Multiplication in Encryption
B1 B2 B2 A1 A2 A2
A1×B1 A2×B2 A3×B3
Im1
Vector-Vector Multiplication
Import to MEM: B1 B2 B3 A1 A2 A3 A4
Multiplication: A1×B1 A2×B1 A3×B1 A4×B2
Write to MEMim: Im1 Im2 Im3 Im1
Column-major Matrix-Vector Multiplication in Key Generation
Ai/Bi: i-th degree-256 polynomial in public matrix/secret vctor

Figure 5: Task-rescheduling pipeline hardware and execution flow of Saber768.

3.3 Truncated Multiplier

The work in [RB20] utilized addition operations and look-up table to perform multiplication operations for Saber. However, this method is only suitable for schoolook multiplication. For Karatsuba multiplications, truncated multipliers for Saber are adopted in our design utilizing the properties of binomial sampling and LWR algorithms.

Thus, multiplications with secret key can be replaced by signed operations with signed values after reverse operations. It is noticed that only 4 bits out of all 13 bits are useful in the calculation, which means that the width of one operand for the multiplier can be reduced to 4 bits. The storage and transmission of private keys are also benefited from the reduction in effective bits. Instead of the modular operation of LWE, the round operation of LWR allows us to trim unnecessary operations. Considering that the parameter q of Saber is 8192, i.e., only the lowest 13 bits in the result are kept, all multiplication operations unrelated to generating the lowest 13 bits in the result can be avoided. However, the effectiveness of this technique is limited by the Pre-Add phase in the kernel hardware and the scheduling layer. This limitation is acceptable because at least 90% multiplication operations are saved through 8-level Karatsuba algorithm.

4 Implementation and Comparison

4.1 FPGA Implementation

Table 1: Performance comparisons for Saber768 on FPGA.

-	Platform	Frequency (MHz)	Time Encaps/Decaps(us)	DSPs	LUTs	Flip-flops	36kb BRAMs
[Far19][a]	UltraScale	322	49/48	256	12566	11619	3.5
[BSNK19][b]	Artix-7	66.7	3550/5472	-	234171	40824	-
[MTK+20]	Artix-7	125	4147/3844	28	7400	7331	2
[RB20]	UltraScale+	250	26.5/32.1	-	25079	10750	-
our design	UltraScale+	100	14.0[c]/16.8[c]	85	34886	9858	6

[a] Only the latency of hardware components is listed.
[b] Only the costs of multiplication hardware and data RAM implemented on FPGA are listed.
[c] The results are estimated through the existing PKE results and additional hash functions.

Table 2: Performance of different stages in LightSaber, Saber768 and FireSaber.

	Key Generation		Encryption		Decryption	
	Cyc-les	Power (mW)	Cyc-les	Power (mW)	Cyc-les	Power (mW)
Light Saber	519	36.3	664	39.2	326	25.0
Saber 768	943	42.7	1156	42.0	408	29.2
Fire Saber	1531	45.7	1811	47.0	490	32.4

The proposed our design crypto-processor is firstly implemented on Xilinx Virtex UltraScale+ FPGA, with its operating frequency of 100 MHz. In terms of resource consumption, 85 DSPs, 34886 LUTs, 9858 Flip-Flops and 6 36-kb-BRAMs are utilized. Among them, the components calculating the degree-256 polynomial multiplication only includes 85 DSPs, 13735 LUTs and 4486 Flip-Flops. In some studies on Saber, only the numbers of cycles for encapsulation and decapsulation of one version, Saber768, are provided, while the crypto-processor proposed in this paper fully supports the PKE scheme of Saber with all three versions. Two additional hash functions, namely, SHA3-256 and SHA3-512, are needed to support the key encapsulation mechanism (KEM). Performance of the KEM scheme of Saber in our design is estimated to participate in the comparison supposing that the Keccak core of the processor is reused.

It is noted that only our design supports all three versions of Saber and other FPGA implementation works only support Saber768. The comparisons of the resource consumption are listed in Table 1. Compared with the cycles count of encapsulation without SHA3-256 and SHA3-512 operations [RB20], our design achieves 5.4×, 5.2× and 4.2× reductions in cycles during key generation, encryption and decryption, respectively. our design on FPGA is 2.1× faster than [RB20] at the encryption stage. The works [Far19, MTK+20] are software-hardware co-design implementations and only the hardware part is included in the comparisons. Compared with the results presented in [Far19], our design consumes only the 29% and 35% of the latency in the encapsulation and decapsulation, respectively. However, the utilizations of LUTs and BRAMs are more than those of [Far19, RB20]. This occurs because our design is mainly designed for the ASIC and there is still room for optimization in FPGA resource consumption.

4.2 Post-layout implementation

The post-layout implementation of our design is achieved based on TSMC 40nm process in the worst process corner. The processor occupies 0.35 mm^2 after placing and routing, where Pre and Post denote pre-process and post-process hardware in scheduling layer. Pre-process and post-process structures consume 20% of area, which supports 3.2× speed-up for degree-256 polynomial multiplication. The number of equivalent gates of our design is 411.5k, which includes the hardware components executing logic operations and memory. The maximum operating frequency is up to 400 MHz with an average power consumption of 40 mW. The detailed implementation results are listed in Table 2.

In Table 3, the results are compared with the works implemented on a mainstream desktop Intel CPU with the optimization of AVX2 and an embedding CPU. It is observed that our design is 11 - 15× faster than the implementation on Intel Core i7 in [DKSRV18]. Moreover, our design is approximately 2100 - 2600× faster than the work on Cortex-M4 CPU in [KBMSRV18]. Compared with the state-of-the-art FPGA implementation [RB20], our design in post-layout platform achieves 8.4× speed improvement.

Table 4 shows the comparison between our design and the state-of-the-art ASIC implementations of other PQC algorithms.The results show that hardware implementation performs better

Table 3: Comparisons with other software implementations of Saber768.

-	Platform	Cycles	Frequency (Hz)	Time(us)	Ratio
Keygen/Encap/Decap [DKSRV18]	Core i7	101k/125k/129k	2.6G	38.90/48.23/49.67	15.0/13.8/11.8
Keygen/Encap/Decap [KBMSRV18]	Cortex M4	1147k/1444k/1543k	168M	6827.4/8595.2/9184.5	2628.4/2462.8/2181.6
our design Keygen/Encrypt/Decrypt	ASIC	943/1156/408	400M	2.36/2.89/1.02	0.91/0.83/0.24
our design Keypair/Encap/Decap	ASIC	1039[a]/1396[a]/1684[a]	400M	2.60[a]/3.49[a]/4.21[a]	1/1/1

[a] The results are estimated through the existing PKE results and additional hash functions.

Table 4: Comparisons with the hardware implementations of other algorithms.

Algorithm	Function	Process (nm)	Frequency (MHz)	Area (mm^2)	Cycles	Energy efficiency (uJ/op)	Post-Quantum Security(bits)
Newhope1024 [BPC19]	encryption	40	72	0.28	106611	12	235
Kyber768 [BPC19]	encryption	40	72	0.28	94440	10.31	161
Newhope1024 [XHY+20]	encapsulation	28	300	0.51[b]	85871	7.02	235
Kyber1024 [XHY+20]	encapsulation	28	300	0.51[b]	81569	7.94	218
NTT-512 [STCZ18]	NTT +DG(Binomial)	40	300	2.05	4196	1.346	-
NTT-1024 [FS19]	NTT	65	25	0.33	-	-	-
NTT-512 [NDBC18]	NTT	45	100	1.4	2854	1.016	-
NIST-P256-ECDSA [BJW+18]	sign	65	20	2	180000	14.58	-
our design LightSaber	encryption	40	400	0.35	664	0.065	115
our design Saber768	encryption	40	400	0.35	1156	0.121	180
our design FireSaber	encryption	40	400	0.35	1811	0.213	245

[a] The results are shown after the process normalization to TSMC 40nm.
[b] The area is estimated through equivalent gates and size of SRAM given.

with respect to both speed and energy efficiency. When encapsulation estimated of our design is compared with the state-of-the-art results [XHY+20] of algorithms with less post-quantum bits, FireSaber outperforms Newhope1024 and Kyber1024, by 53 and 52$\times$ speed, 28 and 32$\times$ in energy efficiency and 77 and 61$\times$ higher area efficiency in our design. For equivalent gates, the work [XHY+20] consumes 979kGE logic gates and 12 kB SRAM, while our design only consumes 411.5kGE. 42.0% equivalent gates are needed in our design when area of SRAM in [XHY+20] is not in the scope of comparison. Besides, the average power of our design is similar as [XHY+20] considering the process factor and 4$\times$ larger than [BPC19]. This is acceptable because the energy consumption of one encryption is still at least one order of magnitude less than that in their works.

5 Conclusions

In this manuscript, a high-performance crypto-processor for Saber based on hierarchical Karatsuba framework is proposed. To the best of our knowledge, we are first to implement Saber in hardware through 8-level Karatsuba algorithm. We hope that our method have some indications for non NTT-style calculating framework in hardware. And we hope our results provide some reference for NIST's third round evaluation.

References

[BJW+18] Utsav Banerjee, Chiraag Juvekar, Andrew Wright, Anantha P Chandrakasan, et al. An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 42–44. IEEE, 2018.

[BPC19] U. Banerjee, A. Pathak, and A. P. Chandrakasan. 2.3 an energy-efficient configurable lattice cryptography processor for the quantum-secure internet of things. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 46–48, Feb 2019.

[BSNK19] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. Nist post-quantum cryptography-a hardware evaluation study. *IACR Cryptology ePrint Archive*, 2019:47, 2019.

[DKSRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2018*, pages 282–305, Cham, 2018. Springer International Publishing.

[Far19] Farnoud Farahmand. Implementing and benchmarking seven round 2 lattice-based key encapsulation mechanisms using a software/hardware codesign approach. 2019.

[FS19] Tim Fritzmann and Johanna Sepúlveda. Efficient and flexible low-power ntt for lattice-based cryptography. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019.

[KBMSRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):243–266, Aug. 2018.

[LHJL05] Chiou-Yng Lee, Jenn-Shyong Horng, I-Chang Jou, and Erl-Huei Lu. Low-complexity bit-parallel systolic montgomery multipliers for special classes of GF(2^m). *IEEE Transactions on Computers*, 54(9):1061–1070, 2005.

[Meh08] Pramod Kumar Meher. Systolic and super-systolic multipliers for finite field GF(2^m) based on irreducible trinomials. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(4):1031–1040, 2008.

[MTK$^+$20] Jose Maria Bermudo Mera, Furkan Turan, Angshuman Karmakar, Sujoy Sinha Roy, and Ingrid Verbauwhede. Compact domain-specific co-processor for accelerating module lattice-based key encapsulation mechanism. Cryptology ePrint Archive, Report 2020/321, 2020. https://eprint.iacr.org/2020/321.

[NDBC18] Hamid Nejatollahi, Nikil D. Dutt, Indranil Banerjee, and Rosario Cammarota. Domain-specific accelerators for ideal lattice-based public key protocols. *IACR Cryptology ePrint Archive*, 2018:608, 2018.

[RB20] Sujoy Sinha Roy and Andrea Basso. High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware. Cryptology ePrint Archive, Report 2020/434, 2020. https://eprint.iacr.org/2020/434.

[Roy19] Sujoy Sinha Roy. SaberX4: High-throughput Software Implementationof Saber Key Encapsulation Mechanism. Cryptology ePrint Archive, Report 2019/1309, 2019. https://eprint.iacr.org/2019/1309.

[STCZ18] Shiming Song, Wei Tang, Thomas Chen, and Zhengya Zhang. Leia: A 2.05 mm 2 140mw lattice encryption instruction accelerator in 40nm cmos. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2018.

[WBW19] Y. Wu, G. Bai, and X. Wu. A karatsuba algorithm based accelerator for pairing computation. In *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 1–3, June 2019.

[XHY$^+$20] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng. Vpqc: A domain-specific vector processor for post-quantum cryptography based on risc-v architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–13, 2020.

[XJHM12] Jiafeng Xie, Jian Jun He, and Pramod Kumar Meher. Low latency systolic montgomery multiplier for finite field GF(2^m) based on pentanomials. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(2):385–389, 2012.

[XMM15] Jiafeng Xie, Pramod Kumar Meher, and Zhi-Hong Mao. Low-latency high-throughput systolic multipliers over GF(2^m) for NIST recommended pentanomials. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(3):881–890, 2015.