Design Space Exploration of SABER in 65nm ASIC

Malik Imran
Tallinn University of Technology
Tallinn, Estonia
malik.imran@taltech.ee

Felipe Almeida
Tallinn University of Technology
Tallinn, Estonia
felipe.almeida@taltech.ee

Jaan Raik
Tallinn University of Technology
Tallinn, Estonia
jaan.raik@taltech.ee

Andrea Basso
University of Birmingham
Birmingham, UK
a.basso@pgr.bham.ac.uk

Sujoy Sinha Roy
Graz University of Technology
Graz, Austria
sujoy.sinharoy@iaik.tugraz.at

Samuel Pagliarini
Tallinn University of Technology
Tallinn, Estonia
samuel.pagliarini@taltech.ee

ABSTRACT
This paper presents a design space exploration for SABER, one of the finalists in NIST’s quantum-resistant public-key cryptographic standardization effort. Our design space exploration targets a 65nm ASIC platform and has resulted in the evaluation of 6 different architectures. Our exploration is initiated by setting a baseline architecture which is ported from FPGA. In order to improve the clock frequency (the primary goal in our exploration), we have employed several optimizations: (i) use of compiled memories in a ‘smart synthesis’ fashion, (ii) pipelining, and (iii) logic sharing between SABER building blocks. The most optimized architecture utilizes four register files, achieves a remarkable clock frequency of 1GHz while only requiring an area of 0.314 mm². Moreover, physical synthesis is carried out for this architecture and a tapeout-ready layout is presented. The estimated dynamic power consumption of the high-frequency architecture is approximately 184mW for key generation and 187mW for encapsulation or decapsulation operations. These results strongly suggest that our optimized accelerator architecture is well suited for high-speed cryptographic applications.

CCS CONCEPTS
• Hardware → Application specific integrated circuits; • Security and privacy → Hardware security implementation; Cryptography.

KEYWORDS
SABER, Lattice cryptography, MLWR, Crypto core, ASIC

ACM Reference Format:
Malik Imran, Felipe Almeida, Jaan Raik, Andrea Basso, Sujoy Sinha Roy, and Samuel Pagliarini. 2021. Design Space Exploration of SABER in 65nm ASIC. In ASHES ’21: Workshop on Attacks and Solutions in Hardware Security, Nov. 19, 2021, Seoul, South Korea. ACM, New York, NY, USA, 6 pages. https://doi.org/xx.yyyy/zzzzzzzzzz

1 INTRODUCTION
Currently deployed public-key cryptographic schemes, i.e., Rivest Shamir Adleman (RSA) and Elliptic-curve Cryptography (ECC), have their security strength built on the hardness of solving hard mathematical problems such as prime factorization and discrete logarithms. While these crypto schemes have been standardized and, to a large extent, remain useful, the recent advances in the field of quantum computers now threat to break them [11]. Therefore, researchers are focusing on designing and investigating quantum-resistant public-key algorithms and protocols to keep future communications secure.

Recently, a competition has been started by the National Institute of Standards and Technology (NIST) for the standardization of post-quantum cryptographic (PQC) public-key protocols [9], i.e., protocols that would not be vulnerable to quantum computers. As the competition approaches its end, the majority of the remaining candidates are based on computationally infeasible lattice problems. One such candidate is a key encapsulation mechanism (KEM) named SABER [6], which is the central piece of this study.

Throughout the standardization/competition process, NIST has considered the security strength of PQC KEM protocols. C/C++ reference implementations of the finalist protocols are available from [9]. Naturally, as with ECC and RSA, having accelerators for PQC candidates is of interest as dedicated hardware can achieve significant speed-ups in performance. Examples of hardware accelerators for NIST PQC protocols are presented in [1, 2, 4, 5, 8, 10, 13] where both field programmable gate array (FPGA) and application specific integrated circuit (ASIC) platforms are targeted.

Comparatively, state-of-the-art hardware implementations of SABER [10, 13] provide significant performance improvements in terms of computational time for the key generation (KeyGen), encapsulation (Encaps) and decapsulation (Decaps) operations. The required computation time for these operations can be further reduced by employing different architectural and circuit-level solutions. Consequently, the focus of this work is to show the design space exploration for the NIST PQC finalist SABER with a focus on improving performance.

The design space exploration, in this work, determines the adaptation in various architectural elements (i.e., distinct memory configurations, pipelining, and logic sharing) with an emphasis on optimizing the design for a specific 65nm ASIC technology. Therefore, to initiate our design space exploration, we have selected an open source implementation of SABER[1]. The existing code targets an FPGA platform, whereas in our work we target an ASIC platform. Converting the code to ASIC is one of the contributions of our work, as well as the following:

The utilized SABER core is modelled as an instruction set coprocessor architecture. The code is written in Verilog at Register Transfer Level (RTL). It can be accessed directly at https://github.com/sujoyetc/SABER_HW.
• Exploration of different types, numbers, and sizes of compiled memories in a ‘smart synthesis’ fashion.
• Promoting logic sharing between SABER building blocks that require similar functionality.
• Pipelining of selected portions of the design, thus trading-off throughput for latency.
• Design of a tapeout-ready SABER core in a commercial 65nm CMOS technology, for which we provide a layout and power, area, and timing characteristics.
• Source codes for our many architectures.

The remainder of this paper is organized as follows: Section 2 provides the required mathematical background and discusses the baseline architecture for the SABER PQC KEM protocol. Our design space exploration is given in Section 3. Implementation results and a comparison to the state of the art is provided in Section 4. Finally, Section 5 concludes the paper.

2 PRELIMINARIES

This section presents the required mathematical background and a description of the chosen baseline architecture for SABER.

Symbols (or notations). The $p$ and $q$ are modulo powers of 2. Set of integers is presented with $\mathbb{Z}$. Then the ring of integers modulo $p$ and $q$ is $\mathbb{Z}_p$ and $\mathbb{Z}_q$, respectively. The ring of polynomials for an integer $N$ is presented with $R_p = \mathbb{Z}_q[x]/(x^N + 1)$ and $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ where $N$ is a fixed power of 2. Vectors are shown in bold and lower case font (e.g., $a$).

Security strength. The security strength relies on the hardness of module Learning With Rounding (Mod-LWR) problem. Therefore, a Mod-LWR sample is defined as follows:

$$\begin{align*}
(a, b) = \left( \frac{P}{q} (a^T s) \right) & \in \mathbb{Z}_q^{r \times 1} \times R_p
\end{align*}$$

(1)

In Eq. 1, $a$ is a vector of randomly generated polynomials in $R_q$, $s$ is a secret vector of polynomials in $R_q$ whose coefficients are sampled from binomial distribution, and the modulus $p < q$. The identification between Mod-LWR samples and uniformly random samples in $\mathbb{Z}_q^{r \times 1} \times R_p$ formulates the Mod-LWR problem. Therefore, this Mod-LWR problem is presumed to be computationally infeasible both on classical and quantum computers. Consequently, SABER is a good candidate for developing quantum-resistant cryptosystems.

PKE and KEM operations. SABER is a Chosen Ciphertext Attack, i.e., IND-CCA, secure KEM and Chosen Plaintext Attack, i.e., IND-CPA, secure public-key encryption (PKE) scheme. Therefore, the PKE crypto operations are the generation of pairs of public and private keys (PKE.KeyGen), encryption (PKE.Enc) and decryption (PKE.Dec). Similarly, the corresponding KEM operations are key generation (KEM.KeyGen), encapsulation (KEM.Encaps) and decapsulation (KEM.Decaps). These operations are described as follows:

Key Generation. PKE.KeyGen starts by randomly generating a seed that defines an $l \times l$ matrix $A$ containing $l^2$ polynomials in $R_q$. A function $gen$ (see Algorithm 1 of [10]) is used to generate the matrix from the seed based on SHAKE-128. A secret vector $s$ of polynomials is also generated. These polynomials are sampled from a centered binomial distribution. The generated public key contains a matrix seed and rounded product $A^T s$, while the secret key contains a secret vector $s$. KEM.KeyGen does not differ from PKE.KeyGen, except that it appends a secret key with a hash of the public key and a randomly generated string $z$.

Encryption and Encapsulation. The PKE.Enc operation consists of generating a new secret $s'$ and adding message to the inner product between the public key and the new secret $s'$. This forms the first part of the ciphertext while the second part contains the rounded product $As'$. The KEM.Encaps operation starts by randomly generating a message $m$ and obtaining from that the public key. The ciphertext $c$ contains the encrypted message and a value achieved from the message and public key.

Decryption and Decapsulation. PKE.Dec requires the secret key $s$ to extract original message from the inner product between the public and secret keys. It is the reverse to PKE.Enc. KEM.Decaps re-encrypts the obtained message with the randomness associated with it and checks whether the ciphertext corresponds to the one received.

Set of parameters. For a security level equivalent to AES-128, AES-192, and AES-256, SABER provides three variants that are termed LightSABER, SABER, and FireSABER, respectively. All three variants use polynomial degree $N = 256$ and moduli $q = 2^{13} \& p = 2^{10}$. They differ only in the module dimension, binomial distribution parameter ($\rho$), and the message space. For more details about security parameters, PKE and KEM operations, we refer readers to algorithms 1–6 of [10].

2.1 Baseline architectures

2.1.1 FPGA Coprocessor architecture of [10]. As introduced in Sec- tion 1, we have used an open source crypto core for which the target platform is FPGA. The coprocessor consists of: (i) a data memory (BRAM with a size of 1024x64); (ii) a program memory; (iii) a dedicated finite state machine based (FSM) controller for orchestrating the SABER operations; and (iv) individual SABER building blocks. The building blocks include: (i) polynomial Vector-Vector multiplier wrapper; (ii) variants of secure hashing algorithms, i.e., SHA3-256, SHA3-512, and SHAKE-128; (iii) a binomial sampler; (iv) AddPack; (v) AddRound; (vi) Verify; (vii) Constant-time Move (CMOV); (viii) Unpack; (ix) CopyWords; and (x) BS2POLVECP.

A BRAM-implemented memory is used to keep initial, intermediate, and final results for the computation of required cryptographic operations. A program memory is employed to enable the coprocessor flexibility and its instruction set architecture (ISA) that compre- hends a number of instructions required by (the variants of) SABER. For polynomial multiplication, inside the Vector-Vector multiplier, a centralized schoolbook multiplier architecture is utilized (described in [3]). A sampler is required to compute a sample from pseudo-random input string for all KeyGen, Encaps, and Decaps operations. The verify block is responsible for comparing two byte strings of the same length. Based on the output of the verify unit, CMOV is responsible to either copy the decrypted session key or a pseudo random string at a specified memory location. The AddPack block computes coefficient-wise addition with a constant followed by generated message. Moreover, it packs the resultant bits into a byte string. Similarly, the AddRound block performs coefficient-wise addition of a constant followed by coefficient-wise

---

3Available from [7].
rounding. The unpack unit converts a byte string into bit string. The BS2POLVECp block converts the byte string into a polynomial vector. A dedicated FSM is responsible for interpreting incoming instructions from the program memory and to communicate/activate the individual building blocks.

2.1.2 Our baseline architecture. To achieve our design premise, i.e., high performance, we have constructed a baseline ASIC architecture for evaluation on a commercial 65nm technology. The first key difference with respect to [10] is the replacement of the BRAM with an SRAM. The SRAM is generated by using a commercial memory compiler provided by a partner foundry. Initially, for the baseline architecture, the memory size is kept identical (1024×64). We will later show many variants where the number of memory instances and their sizes are optimized with the aim of improving the clock frequency.

It is important to note that our baseline architecture remains a coprocessor architecture and that the same ISA is utilized. We assume the program memory resides outside of the SABER accelerator core. The same building blocks utilized in [10] are kept in our work, but most of them are modified during our optimizations, which we detail in the next section.

3 DESIGN SPACE EXPLORATION PROCESS

To differentiate our generated architecture to one another, we have adopted a different name for each design as shown in Fig. 1. In order to provide a simple terminology for our studied architectures, we make use of the prefixes DP and SP, meaning that the architecture employs either a dual-port or a single-port memory. Similarly, the PIP prefix implies that the architecture in question is pipelined. Based on this terminology, the following architectures are considered:

- Baseline
  - DP_1(1024×64)
  - DP_2(1024×32)
  - DP_4(1024×16)

- Optimized
  - DP_8(512×16)
  - PIP_DP_4(1024×16)
  - PIP_SP_4(256×64)

Therefore, we have presented five optimized designs originating from our baseline architecture. The memory is structured as (\(m \times n\)), where \(i\) is the number of instances, \(m\) is the number of memory addresses, and \(n\) is the data width of each address.

In addition to the FSM controller and building blocks shown in Fig. 1, our design space exploration led to the creation of new units: (i) memory manager; (ii) pipeline register; and (iii) shared shift buffer. All these units are common to all of our studied architectures, except for the pipeline register that is employed only in our pipeline architectures, i.e., PIP_DP and PIP_SP. Furthermore, we have done modifications to many building blocks to synchronize their inputs/outputs with the memory timing requirements. The modified blocks are shown with dashed lines in Fig. 1.

3.1 Memory manager

A smart memory synthesis [12] approach is investigated and implemented in our Memory Manager unit. We clarify that the central concept of smart synthesis is the observation that having smaller and distributed memories can be advantageous in an ASIC design. Smaller memories require simpler address decoder units (which are faster). This, combined with the fact that part of the address decoding is now described as logic and can be co-optimized with the remainder of the design, leads to performance improvements with sometimes marginal increase in area. In this work, we explore a smart memory synthesis strategy within the limitations of a commercial memory compiler.

For KEM operations, when the security is equivalent to AES-192, SABER requires 992, 1344, and 1088 bytes for generating a single public-key, secret-key, and a cipher text [6]. Therefore, a relatively large memory (1024 × 64) is employed in [10]. We have used the same memory size in our baseline architecture. To initiate our design space exploration process, we have divided the data width (64 bit) of the employed memory into smaller chunks (32 and 16) and increased the number of memory instances accordingly. With this division, the memory structure becomes DP_2(1024×32) and DP_4(1024×16). This design choice results in an increase in clock frequency at the expense of area and power. Thereafter, from DP_4(1024×16) memory structure, we have constructed another architecture where we have reduced the required number of memory addresses from 1024 to 512. In this case, the memory structure becomes DP_8(512×16). Conversely, this design choice results in an increase in area and power with a marginal gain in clock frequency. Therefore, at this point, we deem that further diving the memories is no longer of interest.

In our first pipelined architecture, i.e., PIP邳, we have used the same 4(1024×16) memory structure as employed in DP_4(1024×16). Our second pipelined architecture, however, utilizes compiled RegFiles\(^3\). One of the limitations of the use of a RegFile is that the IP available to us is single-port, meaning that the design has to be modified such that all building blocks that benefit from concurrent read and write operations now execute them sequentially, one after the other. The consequence is that the overall number of clock cycles for a given cryptographic operation will increase. Later, we

\(^3\)RegFiles are not flip-flops. This is a vendor-specific terminology for a compiled 6T SRAM memory that is advantageous when bit density can be traded-off with performance. It is also termed a "high-speed" variant of SRAM by its vendor.
will show that this increase is beneficial since the improved clock frequency still reduces the overall latency for all SABER operations. The memory structure of the PIP_SP architecture is 4(256×64).

3.2 Pipelining

Initially, with the goal of improving clock frequency, we have employed different memory configurations until the improvements in clock frequency were exhausted. However, as the memory configurations change, the critical path of the design changes as well. In order to shorten the critical path and to further optimize the clock frequency, we have to explore other circuit level solutions, such as selective pipelining.

Based on the evaluation of the critical path of several architectures (details are given in section 4.1), it becomes evident that the memory is the performance bottleneck of the design. For this reason, we have placed pipeline registers at the memory output. This guarantees that the critical path is proportional to the memory access time (as opposed to being proportional to the memory and to the logic that follows it). Therefore, in our PIP_DP and PIP_SP architectures, the input to the pipeline register is from the memory while the output is connected to the binomial sampler (not shown in Fig. 1).

3.3 Shared shift buffer

For several building blocks of SABER, i.e., AddRound, AddPack, BS2POLVEC_p, and multiplier, a shift register is required to read from many memory addresses and accumulate (hundreds of) bits into local registers. For example, a 320-bit long register is required in AddPack and BS2POLVEC_p, while a 64 and 676 bit register are required in AddPack and Multiplier, respectively. It is important to mention that all the SABER building blocks produce outputs serially, so the shift buffer can be shared as there are no concerns with concurrent access. Therefore, we have efficiently employed a single 676-bit register that is shared by AddRound, AddPack, BS2POLVEC_p, and Multiplier. The use of a shared shift buffer results in a 10.3% decrease in the total area with no impact on performance. All results given in the next section consider the use of this shared buffer by all architectures.

4 RESULTS AND COMPARISONS

The synthesis results on a 65nm commercial technology for our baseline and optimized architectures are presented in Table 1. These results are obtained after logic synthesis in Cadence Genus. The initial power estimates are obtained by assuming constant switching probabilities (i.e., while considering a synthetic workload).

4.1 Critical path analysis

The critical paths of our baseline and optimized architectures are shown in Fig. 2. Our analysis reveals that the memories containing longer access time result in longer critical paths for most architectures (i.e., the memory presents itself as the bottleneck) while the use of faster RegFiles result in a shorter critical path. In other words, as shown in Fig. 2, the critical path of our baseline architectures depend on the memory and some amount of combinational logic (to a lesser degree). However, this is not the case for our optimized PIP_SP architecture where the critical path is mostly combinational logic (and the setup time of the destination flip-flop). This result implies that our optimized architecture is saturating the memory bandwidth thanks to our optimization strategies at architecture and circuit levels.

4.2 Physical layout for PIP_SP

The layout of CCA-secure KEM SABER accelerator, as shown in Fig. 3, is obtained from Cadence Innovus. The accelerator circuit was implemented with a nominal voltage of 1.2V in a 65nm CMOS technology.
### Table 1: Logic Synthesis results for CCA-secure KEM SABER

| Design          | Area Information | Timing Information | Power Information (in mW) |
|-----------------|------------------|--------------------|---------------------------|
|                 | Area (mm²) | Gates | Clk. P (ns) | Freq. (MHz) | Crypto core | Combinational logic | Memory |
|                 |           |       |             |            | Lkg | Dyn | Lkg | Dyn | Lkg | Dyn |
| DP_1(1024×64)   | 0.299     | 43336  | 2.000       | 500        | 0.090 | 86.844 | 0.059 | 16.235 | (19%) | 0.003 | 38.001 | (44%) |
| DP_2(1024×32)   | 0.308     | 45319  | 1.718       | 582        | 0.091 | 104.835 | 0.059 | 18.499 | (18%) | 0.010 | 157.490 | (71%) |
| DP_4(1024×16)   | 0.340     | 39981  | 1.638       | 610        | 0.082 | 135.342 | 0.051 | 18.762 | (14%) | 0.006 | 81.368 | (60%) |
| DP_8(512×16)    | 0.478     | 45979  | 1.624       | 615        | 0.099 | 220.410 | 0.062 | 21.691 | (10%) | 0.010 | 157.490 | (71%) |
| PIP_DP_4(1024×16) | 0.365     | 46217  | 1.508       | 663        | 0.097 | 233.361 | 0.063 | 20.890 | (10%) | 0.006 | 168.476 | (72%) |
| PIP_SP_4(256×64) | 0.314     | 64230  | 0.998       | 1002       | 0.111 | 142.413 | 0.074 | 32.925 | (23%) | 0.006 | 39.060 | (27%) |

| Table 2: CCs information for SABER building blocks |
|---------------------------------------------------|
| building blocks | Clock cycles | Reason |
|-----------------|--------------|--------|
| Binomial Sampler | 145 | 246 | Pipelining |
| Multiplier      | 894 | 970 | Memory sync. |
| Unpack          | 167 | 295 | Memory sync. |
| CopyWords       | 60  | 211 | Single-port RegFile |
| Others          | -  | -  | No change |

| Table 3: Total CCs and latency for CCA-secure KEM SABER on a 65nm commercial technology |
|---------------------------------------------|
| Designs | Total clock cycles | Latency (µs) |
|        | KeyGen Encaps Decaps | KeyGen Encaps Decaps |
| DP_1   | 5644 6990 8664 | 11.2 13.9 17.3 |
| DP_2   | 5644 6990 8664 | 9.6 12.0 14.8 |
| DP_4   | 5644 6990 8664 | 9.2 11.4 14.2 |
| DP_8   | 5644 6990 8664 | 9.1 11.3 14.0 |
| PIP_DP_4 | 5741 7087 8761 | 8.6 10.6 13.12 |
| PIP_SP_4 | 7154 7136 9359 | 7.1 7.1 9.3 |

### Figure 2: Critical path analysis for our studied architectures.

The results achieved after physical synthesis for different corners are given in Table 4. These results were obtained with the aid of value change dump (VCD) files, i.e., files that capture the activity of the design based on representative simulation loads. Thus, the power values reported here are more realistic. Three different corners were used for characterization: slow-slow (SS), typical-typical (TT), and fast-fast (FF). These corners have operating conditions for different voltages and temperatures. The results reveal, as expected, that FF consumes more power than TT. Similarly, TT consumes more power than SS.

The comparison to existing SABER implementations is given in Table 5. Column one provides the reference implementation while the targeted platform is given in column two. The latency in µs for KEM.KeyGen, KEM.Encaps and KEM.Decaps is given in column three. Column four provides the clock frequency (MHz). Finally, the last column provides the area for FPGA (in terms of look-up-tables and flip-flops) and ASIC (in mm²) platforms. We have placed a ‘–’ where required information is not available.
Comparison to FPGA implementations [1, 5, 8, 10]. In terms of computation time (shown in Table 5), the most efficient implementation of SABER on FPGA is described in [10]. It takes 5453, 6618 and 8034 CCs for the computation of one KEM.KeyGen, KEM.Encaps and KEM.Decaps which are comparatively 24%, 8% and 15% lower than our PIP_SP architecture. Moreover, our PIP_SP architecture requires 3.07, 3.73 and 3.45 times lower latency. For same operations, the proposed PIP_SP architecture takes 450.7, 577.4 and 408.6 times lower latency as compared to [8]. Additionally, our PIP_SP architecture achieves 8 and 4 times higher clock frequency as compared to [8] and [10], respectively.

On Xilinx Zynq Ultrascale+ MPSoC, a software/hardware co-design processor architecture is presented in [5]. For KEM.Encaps and KEM.Decaps, our PIP_SP architecture is 8.45 and 6.98 times faster (in terms of latency). As compared to lightweight implementation of SABER, described in [1], our PIP_SP architecture requires 65.78 and 56.73 times lower latency for KEM.Encaps and KEM.Decaps, respectively. Moreover, our PIP_SP architecture results 10 and 3.10 times higher clock frequency as compared to [1] and [5]. Noted that the area comparison to [1, 5, 8, 10] is not possible due to distinct implementation platforms (as we have provided synthesis on ASIC while [1, 5, 8, 10] utilizes FPGA).

Comparison to ASIC accelerator [15]. As shown in Table 5, our optimized PIP_SP architecture has higher latency. On the other hand, we are utilizing 1.21 times lower hardware resources on a 65nm technology while the referenced work utilized 40nm. It is therefore likely that our design would be a fraction of the size in the same technology. Moreover, we are achieving 2.5 times higher clock frequency. For multiplication of two 256-degree polynomials in SABER, we have employed a centralized schoolbook multiplier architecture of [3]. It takes 256 CCs to compute one polynomial multiplication. On the other hand, in [15], the use of an 8-level Karatsuba multiplier for the same polynomial length requires 81 CCs instead of 256.

Furthermore, a high-speed Keccak module containing two parallel sponge functions (Keccak-f) is used in [15]. It computes two Keccak-f[1600] computations in each clock cycle and each round of Keccak is performed every 12 CCs. In our architectures, a single sponge function in a serial fashion is incorporated which results in 28 CCs to generate 1,344 bits of a pseudo-random string. In addition to aforesaid differences in performance, our implementation follows a coprocessor architecture while a fully parallelized architecture is described in [13]. Consequently, the decrease in clock cycles in [13] ultimately shows decrease in computation time.