HaMAYO: A Reconfigurable Hardware Implementation of the Post-Quantum Signature Scheme MAYO

Oussama Sayari[1], Soundes Marzougui[1,2], Thomas Aulbach[3], Juliane Krämer[3], and Jean-Pierre Seifert[1,4]

[1] Technical University of Berlin, Berlin, Germany
[2] STMicroelectronics, Diegem, Belgium
[3] Regensburg University, Regensburg, Germany
[4] Fraunhofer Institute SIT, Darmstadt, Germany

Abstract. MAYO is a topical modification of the established multivariate signature scheme Unbalanced Oil and Vinegar (UOV), with a significantly reduced public key size while maintaining the appealing properties of UOV, like short signatures and fast verification. Therefore, MAYO is considered an attractive candidate in the NIST standardization process for additional post-quantum signatures and an adequate solution for real-world deployment in resource-constrained devices.
This paper presents the first hardware implementation of the signature scheme MAYO. Our implementation can be easily integrated with different FPGA architectures. Additionally, it includes an agile instantiation with respect to the NIST-defined security levels for long-term security and encompasses modules' optimizations such as the vector-matrix multiplication and the Gaussian elimination method employed during the signing process. Our implementation is tested on the Zynq ZedBoard with the Zynq-7020 SoC and its performance is evaluated and compared to its counterpart multivariate scheme UOV.

Keywords: MAYO · Multivariate Cryptography · Post-Quantum Cryptography · Digital Signature · Hardware Implementation

1 Introduction

As quantum computing continues to advance, it is anticipated that quantum attacks can break many of the computational problems that classical cryptography relies on, such as factorization and discrete logarithms used in RSA and ECDSA, respectively. To address this, researchers have proposed new mathematical assumptions and computational problems that are difficult to solve with quantum computers, resulting in the field of post-quantum cryptography. These new assumptions are grouped into different families, such as lattice-based, code-based, hash-based, and multivariate cryptography.

Multivariate schemes mainly rely on the difficulty of solving large systems of multivariate quadratic equations, known as the MQ Problem. As such, the signature scheme Rainbow [DS05] was a finalist in the third round of the NIST post-quantum cryptography (PQC) Standardization Process. Rainbow is a two-layered version of the UOV signature scheme [KPG99]. Hence, multivariate signature schemes based on the oil and vinegar principle received a lot of attention. They offer very short signatures and efficient verification, since the signature is mainly the solution to a system of multivariate quadratic equations, and verifying boils down to evaluating the polynomials at the presumed solution. Still, during the third round, Beullens developed an algebraic attack on Rainbow [Beu22a], targeting the layer structure that differentiates Rainbow from UOV. This led to the elimination of Rainbow from the ongoing process since it lost all its alleged advantages over the base scheme UOV.

Meanwhile, NIST has completed the third round of the PQC standardization process, which resulted in the selection of a total of four candidate algorithms for standardization, i.e., two lattice-based signature schemes, one hash-based signature scheme, and one lattice-based Key Encapsulation Mechanism (KEM). An upcoming round of post-quantum digital signature standardization will be introduced, where NIST aims at enhancing the variety of the signature schemes by prioritizing signature schemes that are not reliant on structured lattices, have small keys, and offer concise and efficient verification processes.

In [Beu22b], Beullens introduced a novel multivariate signature scheme called MAYO, which builds upon UOV. It uses the same trapdoor - a secret oil space that is annihilated by the public key map - but is developed such that the signer and the verifier locally enlarge the public key matrices. Therefore, the dimension of the oil space can be reduced. That allows also to reduce other parameters like the number of variables in the quadratic equations since certain algebraic attacks get harder with a smaller oil space [KS06]. In total, this leads to significantly smaller public keys in MAYO, while keeping good performance numbers and signature sizes. For instance, with parameters targeting the first security level of the NIST process, the public key size of MAYO is 1,168 bytes, the secret key is 24 bytes, and the signature size is 321 bytes [BCC+23]. These results make the MAYO signature scheme more compact than other state-of-the-art lattice-based signature schemes such as Falcon and Dilithium [PQD23].

Contribution In this paper, we present the first pure hardware implementation of the multivariate signature scheme MAYO. Our contribution is summarized as follows:

- We manually settle a pure hardware implementation of MAYO. Our implementation is reconfigurable and can be easily integrated with different FPGA architectures and for different security levels.
- We optimized certain functionalities used within key generation and signing.
- We present a new approach for the Gaussian solver and compare it to the well-known GSMITH approach of Rupp et al. in [REBG11].

The source code of our implementation is publicly available [5].

[5] https://anonymous.4open.science/r/MAYO-0A15/

Related work At the time of writing this paper, there is a scarcity of complete hardware designs for post-quantum cryptographic schemes [ZZW$^+$21, XL21, FG18, HZ18]. However, given that the NIST PQC reached the fourth round, it is expected that more dedicated hardware designs will emerge. These designs would be instrumental in showcasing the strength and inherent properties of specific protocols [NIS23a].

Multivariate schemes necessitate the development of comprehensive and extensive implementation designs to address the challenging gaps due to the schemes' large key sizes [DS05, KPG99]. These key sizes often pose challenges for devices with limited resources, as they may struggle to accommodate the storage requirements of these schemes. Moreover, multivariate schemes commonly involve memory and time-consuming blocks, with the Gaussian solver being a well-known performance bottleneck [REBG11]. Despite the above-mentioned challenges, there have been a few published hardware implementations that have reported results for multivariate schemes [TYD$^+$11, HZ18, FG18].

In [FG18], Ferozpuri and Gaj present a high-speed FPGA implementation of Rainbow. Their hardware implementation uses a parameterized system solver where the execution time is proportional to the system dimension, i.e., it can solve an n-by-n system in n clock cycles. Moreover, their work reduces the number of required multipliers by almost half, speeds up execution as compared to the previous state-of-the-art work, and implements Rainbow for higher security levels.

In [TYD$^+$11], Tang et al. present another high-speed hardware implementation of Rainbow. The authors targeted similar functionalities for optimization as in [FG18], i.e., the Gaussian solver and the multipliers. They developed a new parallel hardware design for the Gaussian elimination and designed a novel multiplier to speed up the multiplication of three elements over a finite field.

With Rainbow being broken [Beu22a], all its previously published software and hardware implementations are rendered obsolete for practical use. To address this issue, MAYO is seen as a viable alternative, showcasing improved performance results. Nevertheless, there remains a significant gap in practical implementations of the MAYO scheme, hindering its real-world deployment. At the moment of writing this paper, there is no hardware implementation of the MAYO post-quantum signature scheme.

2 Preliminaries

Notation. We write $\mathbb{F}_q$ for a finite field with q elements. The set of matrices over $\mathbb{F}_q$ with m rows and n columns is given by $\mathbb{F}_q^{m\times n}$. By $\mathbf{I}_{n\times n}$ we denote the identity matrix of size n and $\mathbf{0}_{m\times n}$ is the $m\times n$ zero matrix. We represent a vector $\mathbf{x} \in \mathbb{F}_q^n$ in bold letters unless specified otherwise. Thus, $\mathbf{0}_n$ is the zero vector in $\mathbb{F}_q^n$. We denote by $\mathbf{x}[i]$ or x_i the i-th entry of $\mathbf{x}$, i.e., $\mathbf{x} = \{x[i]\}_{i\in[n]} = \{x_i\}_{i\in[n]}$. For $0 \leq i < j \leq n$, we mean by $x[i:j] \in F_q^{j-i}$ the vector whose $j-i$ elements are $x_i, \ldots, x_{j-1}$. We define the component-wise sum as $\mathbf{x}+\mathbf{y} := \{x_i + y_i\}_{i\in[n]}$, and the scalar multiplication as $a\mathbf{x} := \{ax_i\}_{i\in[n]}$. For the concatenation of the two vectors $\mathbf{x} \in \mathbb{F}_q^n$, $\mathbf{y} \in \mathbb{F}_q^m$, we write $(\mathbf{x}, \mathbf{y})$ as the corresponding vector in $\mathbb{F}_{n+m}^q$. For a finite set X, we write $x \leftarrow X$ to indicate the sampling of the element x from the uniform distribution over X. For $a, b \in \mathbb{N}$, we denote by $[a, b]$ the set $\{a, \ldots, b\}$ and similarly we denote by $[b]$ the set $\{1, \ldots, b\}$.

2.1 Multivariate Quadratic Maps

The MAYO signature scheme [Beu22b] is a special modification of the UOV signature scheme [KPG99] and belongs to the field of multivariate cryptography. Herein, the main object is the multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ with m components and n variables. In more detail, it is a sequence $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$ of m quadratic polynomials in n variables $\mathbf{x} = (x_1, \ldots, x_n)$, with coefficients in a finite field $\mathbb{F}_q$. If we want to evaluate the map $\mathcal{P}$ at a given vector $\mathbf{a} \in \mathbb{F}_q^n$, we can simply evaluate each of its component polynomials in $\mathbf{a}$ to get a vector $\mathcal{P}(\mathbf{a}) = \mathbf{b} \in \mathbb{F}_q^m$, where the i-th entry of $\mathbf{b}$ is given by $b_i = p_i(\mathbf{a})$. Very abbreviated, multivariate cryptography is based on the hardness of finding a preimage $\mathbf{s} \in \mathbb{F}_q^n$ of a target vector $\mathbf{t} \in \mathbb{F}_q^m$ under a given multivariate quadratic map $\mathcal{P}$, i.e., solving a multivariate system of quadratic equations. This task is often referred to as the MQ problem.

To solve this problem for parameters that are relevant in cryptography, various algorithms have been developed such as F4/F5 or XL that use a Gröbner-basis-like approach [Fau99, CKPS00]. Since their efficiency depends on the given instance of the problem and the parameter sizes, the preferable algorithm might vary in different settings. Recently, the most important classical algorithms have been collected in [BMSV22], where an overview of the estimated computational complexities of the respective algorithms is presented.

There are mainly two different approaches of constructing signature schemes from the MQ Problem. The first is to employ a zero-knowledge identification scheme and turn it into a non-interactive signature scheme with the Fiat-Shamir transform [FS87]. In the multivariate case, this implies the proof of knowledge of a certain secret $\mathbf{s}$, such that $\mathcal{P}(\mathbf{s}) = \mathbf{v}$ for a random multivariate quadratic map $\mathcal{P}$. Mudfish [Beu20] and MQDSS [HRSS16] are based on such protocols, but they come with the drawback of rather huge signature sizes since a lot of parallel rounds are required to obtain a secure scheme. Second, one can follow the hash-and-sign approach, where the message is hashed to a target vector in the right domain. This requires including a trapdoor into the public key map $\mathcal{P}$, such that the signer is able to invert this map and find a preimage $\boldsymbol{s}$ of $\mathbf{t}$. Both UOV and MAYO follow this idea and even share a similar trapdoor. We will introduce them in the following.

2.2 The Trapdoor in UOV

In UOV, the trapdoor information is a basis of a secret linear subspace $\mathcal{O} \subset \mathbb{F}_q^n$ of dimension $\dim(\mathcal{O}) = m$, the so-called oil space [Beu21]. The multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is then chosen in a way that it vanishes on this oil space, i.e., $\mathcal{P}(\mathbf{o}) = \mathbf{0}_n$ for all $\mathbf{o} \in \mathcal{O}$. For the multivariate quadratic polynomials

$p_i(\mathbf{x})$, which constitute the map $\mathcal{P}$ via $\mathcal{P}(\mathbf{x}) = p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$, one can define their *polar form* or *differential* as

$$p_i'(\mathbf{x}, \mathbf{y}) \coloneqq p_i(\mathbf{x} + \mathbf{y}) - p_i(\mathbf{x}) - p_i(\mathbf{y}) + p_i(\mathbf{0}).$$

Since we commonly work with homogeneous polynomials, the term $p_i(\mathbf{0})$ will be omitted in the following. Similarly, we can define the polar form of $\mathcal{P}$ as

$$\mathcal{P}'(\mathbf{x}, \mathbf{y}) = p_1'(\mathbf{x}, \mathbf{y}), \ldots, p_m'(\mathbf{x}, \mathbf{y}).$$

As shown in [Beu21, Theorem 1], the map $\mathcal{P}' : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a symmetric and bilinear map. Furthermore, if one has knowledge of the secret oil space, it can be used to efficiently find preimages $\mathbf{x} \in \mathbb{F}_q^n$ of a given target $\mathbf{t} \in \mathbb{F}_q^m$ such that $\mathcal{P}(\mathbf{x}) = \mathbf{t}$. To do so, one can randomly pick a vinegar vector $\mathbf{v} \in \mathbb{F}_q^n$ and solve the system $P(\mathbf{v} + \mathbf{o}) = \mathbf{t}$ for $\mathbf{o} \in \mathcal{O}$. This is possible since in

$$\mathbf{t} = \mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathcal{P}(\mathbf{v}) + \mathcal{P}(\mathbf{o}) + \mathcal{P}'(\mathbf{v}, \mathbf{o}) \tag{1}$$

the term $\mathcal{P}(\mathbf{v})$ is constant and $\mathcal{P}(\mathbf{o})$ vanishes, so whenever the linear map $\mathcal{P}'(\mathbf{v}, \cdot)$ is non-singular, the system has a unique solution $\mathbf{o} \in \mathcal{O}$, which can be computed efficiently. This happens with probability roughly $\frac{q-1}{q}$. If this is not the case, one can simply pick a new value for $\mathbf{v}$ and try again. Without a description of the oil space $\mathcal{O}$, the term $\mathcal{P}(\mathbf{o})$ implies that Equation 1 constitutes a system of quadratic equations, which remains hard to solve.

Remark 1. The given description of signatures using the oil and vinegar approach is different from the original one as in [Pat97] or [KPG99]. Nevertheless, it allows for a simpler representation and is closer to the way schemes like UOV and MAYO are implemented nowadays.

Building a signature scheme directly from this setting has one big disadvantage. The oil space needs to be as large as the image space of the multivariate quadratic map $\mathcal{P}$, i.e., $\dim \mathcal{O} = m$. To counter the Kipnis-Shamir attack [KS06], the parameter n needs to be sufficiently larger than m, with $n \approx 2,5m$ being used in all currently considered implementations. The parameter m itself needs to be of a certain size as well, to provide security against direct attacks or the intersection attack [Beu21]. This leads to key pairs of enormous size, which is considered the main drawback of multivariate signatures. Recently, Beullens developed the signature scheme MAYO to tackle this problem.

2.3 Description of MAYO

The essential modification is the downsizing of the dimension of the oil space to $\dim \mathcal{O} = o < m$. Actually, this oil space is now too small to sample signatures, since the system $\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathbf{t}$ given in Equation 1 consists consequently of m linear equations in o variables and is unlikely to have any solutions. Thus, the approach taken in [Beu22b] is to stretch the public key map into a larger whipped map $\mathcal{P}^* : \mathbb{F}_q^{kn} \to \mathbb{F}_q^m$, such that it accepts k input vectors $\mathbf{x} \in \mathbb{F}_q^n$. This is realized by defining

$$\mathcal{P}^*(\mathbf{x}_1, ..., \mathbf{x}_k) \coloneqq \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{x}_i) + \sum_{1 \leq i < j \leq k} \mathbf{E}_{ij}(\mathcal{P}'(\mathbf{x}_i, \mathbf{x}_j)), \tag{2}$$

where the matrices $\mathbf{E}_{ij} \in \mathbb{F}_q^{m \times m}$ are fixed system parameters with the property that all their non-trivial linear combinations have rank m. In the MAYO specification it is currently proposed to choose $\mathbf{E}_{ij}$ that represent multiplications by $1, X, X^2, \ldots, X^{\binom{k}{2}-1}$ in $\mathbb{F}_q[X]/f(X)$, for some monic irreducible polynomial $f(X)$ of degree m. This is possible for parameter sets that satisfy $\binom{k}{2} < m$.

It is easy to see that $\mathcal{P}^*$ vanishes on the subspace $\mathcal{O}^k = \{(\mathbf{o}_1, \ldots, \mathbf{o}_k) | \text{ with } \mathbf{o}_i \in \mathcal{O} \text{ for all } i \in [k]\}$ of dimension ko. By choosing the parameters such that $ko > m$, the k copies of the oil space are large enough to construct preimages of a target vector $\mathbf{t} \in \mathbb{F}_q^m$ under the whipped map $\mathcal{P}^*$. In more detail, the signer randomly samples $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathbb{F}_q^{kn}$, and then solves

$$\mathcal{P}^*(\mathbf{v}_1 + \mathbf{o}_1, ..., \mathbf{v}_k + \mathbf{o}_k) = \mathbf{t} \tag{3}$$

for $(\mathbf{o}_1, ...\mathbf{o}_k) \in \mathcal{O}^k$. Observe from Equation 2 that this system remains linear in the presence of the linear emulsifier maps $\mathbf{E}_{ij} \in \mathbb{F}_q^{m \times m}$. Thus, the signer can efficiently compute a preimage $\{\mathbf{s}_i = \mathbf{v}_i + \mathbf{o}_i\}_{i \in [k]}$ of $\mathbf{t}$. Similar to UOV, the verifier just needs to check if the given $\{\mathbf{s}_i\}_{i \in [k]}$ satisfy Equation 3.

Remark 2. Please note that both, the signer and the verifier, only locally whip up the public key map $\mathcal{P}$ to $\mathcal{P}^*$, so this modification comes with no additional cost in terms of key sizes. However, it entails additional computations during signing and verification. Furthermore, it increases signature size, since now a k-tuple of vectors in $\mathbb{F}_q^n$ constitute the signature. These negative effects are cushioned by the ability to reduce parameter sizes while maintaining the security level.

3 Hardware Design

In this section, we discuss the different aspects that led to the reasoning behind our hardware design. Therefore, we introduce a general description of our design and detail on the optimized modules. Our primary goal is to provide an optimized and reconfigurable hardware code that can be easily integrated with different FPGA architectures and for different security levels.

3.1 Design Rationale

The selection of the FPGA chip is crucial and must be taken into consideration, as most low-tier options may not offer sufficient memory and resources to house the design and the intermediate calculations required by MAYO.

Although MAYO's keys and signature stream have reduced fingerprint compared to other multivariate alternatives, it still internally necessities approximately 140KB for key-generation process and 250KB of memory to execute the signing phase, for the first security level defined by NIST [Beu22b]. For implementation and testing of our hardware design, we opted for the target board Zynq ZedBoard with the Zynq-7020 SoC [Xil23b], which has 85K Logic Cells and 4.9MB Block RAM, deemed to be the main storage space of the overall hardware design. The board is also equipped with 512MB DDR3 RAM and 256MB QSPI Flash as external memory. The Zynq ZedBoard offers also an ARM Cortex-A9 hardcore that will be employed later in our design.

The majority of the system architecture of our hardware design is described in VHDL, while a few modules are implemented using Verilog. AMD's Vivado SDK [Xil23a] tool provides, in fact, such source language mixture during the design's synthesis and validation to ensure flawless compatibility.

It is essential for the architecture to be encapsulated as an Intellectual Property (IP), to ensure design reuse. Therefore, two main IPs, namely Keygen, and Sign, were developed to describe the hardware implementation of MAYO. It is possible to utilize one of the IPs on the target chip. Both cores are independent and capable of coexisting on the Programmable Logic operating at respectable frequencies.

The CPU-Peripheral communications between the built IPs are handled through AXI4-FULL, AXI-Lite, and interrupts. The provided firmware takes care of the AXI transactions, thanks to the Zynq hybrid architecture. Incidentally, the design focuses on maintaining high transfer bit-rates by extensively leveraging the CPU's 32-bit architecture. Frequencies and reset signals are also controlled by the hardcore and are propagated throughout the design.

Based on the proposed MAYO pseudo-code in [BCC+23], the scheme incorporates multiple helper functions that are implemented as sub-modules and arithmetic units within the hardware IPs. This approach fulfills another significant design requirement by minimizing unused modules and maximizing the utilization of Flip-Flops (FFs) and Lookup Tables (LUTs). By avoiding code duplication in hardware and organizing the design into smaller, specialized modules, each capable of performing a single functionality, the overall efficiency and modularity of the design are improved.

All the previously explained design decisions and optimizations contribute to a more understandable, efficient, and integrated system, which will be discussed in depth in the following section.

3.2 General Description

Since the targeted board is of AMD's Zynq Chip Family, the external data-flow including the public key, the message input, and resulting signature extraction are provided by the Zynq dual Cortex ARM Core. The design incorporates both an AXI4 interface for data stream and an AXI-Lite interface for configuration and control. Therefore, to use the proposed implementation, it is encouraged to utilize the provided C firmware discussed later in Section 3.8.

The design aligns itself with the 32-bit ARM multi-core processor architecture and uses a 32-bit data bus width. This approach simplifies data processing within each sub-module. In the case of MAYO, the values are usually stored in a 5 bits-wide reduced space. For the NIST security level 1, the scheme operates on values that are eventually reduced to $\mathbb{F}_q$ [6], meaning that the results must be less or equal to $q = 31$. To store such numbers, $5 = \lceil \log_2(31) \rceil$ bits are mandatory. As a result, the design allocates 8 bits of memory (i.e., unsigned char) for each numerical unit. We, then, exploited the 32-bit architecture in various pipeline techniques by processing simultaneously four 8-bit values.

MAYO core's structure is generalized as two primal IPs consisting of combinatorial Finite State Machines (FSMs) that govern the multivariate scheme procedure. These FSMs are interconnected with various multi-purpose sub-modules through an extended wire network, that determines the destination of the exchanged data on the bus laying in between. As depicted in Figure 2, although both FSMs coexist, they typically share the same bus to any specific arithmetic core. Each core, though designed for an exclusive functional purpose (such as subtracting two vectors) is enabled multiple times throughout the design, with different source and destination addresses to the system's memory.

Likewise, the sub-modules themselves are smaller state machines designed with a focus on parallelism and reaching maximum throughput. The time-consuming modules are pipelined and feature a familiar bundle of ports, as illustrated in Figure 1. The IPs' state machines define these ports' content and await the port's signal indicating the completion of the operation in hand. The process then moves on to the subsequent step of the MAYO scheme.

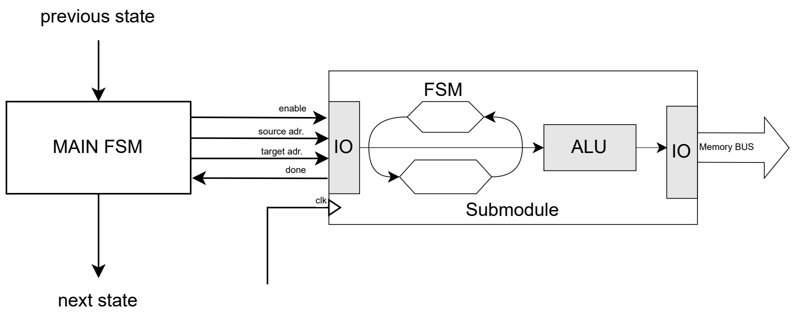

Figure 1: IP-Submodule interconnect block diagram

The MAYO public key map is expanded throughout certain steps of the algorithm (Equation 2, Section 2.3). To address this, MAYO's polynomial mappings $\mathcal{P}$ and $\mathcal{P}^*$ (in Section 2.3), and the vectors that represent solutions to the various systems of equations required by the algorithm are stored within three separate Block Random Access Memories (BRAMs). These storage units refer to

[6] The chosen MAYO security parameters (n, m, o, k, q) are $(62, 60, 6, 10, 31)$ as in [BCC$^+$23]

dedicated configurable memory blocks within the FPGA, providing fast on-chip access to multiple storage cells organized into an array structure. For our design, Vivado's BRAM generator is employed, allowing the SDK to automatically organize the BRAM layout based on specified depths. This technology can be easily replaced on other FPGA alternatives such as Intel's QDR SRAM Controller in Quartus Prime, or by simply re-describing the memory as RTL arrays that can be understood by the compiler of choice [Int].

Other than registers and temporary value holders inside the sub-modules, BRAMs serve as the main storage unit for the MAYO hardware system. The memory is divided into *three* True Dual Port BRAMs, statically partitioned into 2×256KB BRAMs to store big matrices and large vectors like the $\mathcal{P}$ system and $\mathcal{O}^k$ subspaces, and 1×8KB BRAM designated for small scratch buffers and sensitive information such as the seed, signature, and secret key. Among these BRAMs, only one of the big BRAMs is exposed to CPU through the AXI bus. Detailed memory management and utilization is deliberated later in Section 3.6. As shown in Figure 2, most modules are connected to the BRAMs accordingly.

To handle the increasing number of memory-to-module interconnections, we needed a multiplexer or a bus controller to merge the modules' I/Os into the memory's interfaces. As a result, only one core is allowed to hold a control signal up. The bus manager then gives this core permission to access the memory space. Furthermore, output registers are placed on the memory's ports to ensure better hold timings for Place and Route (P&R).

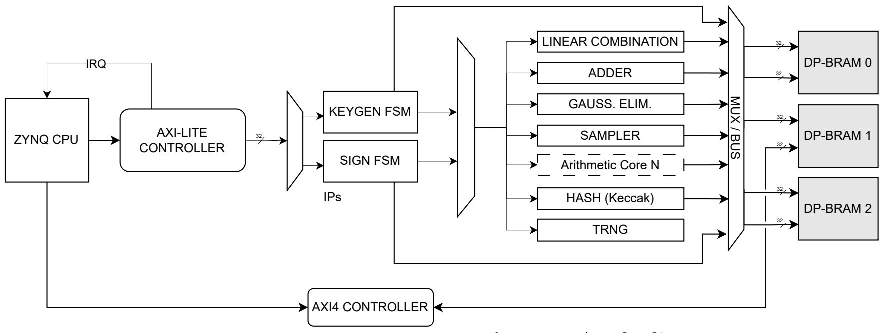

Figure 2: Block Diagram of the MAYO Core

3.3 Hash Function

Our design employs the Keccak core [BDH+22] to generate seeds and expand the message as a first step of the signing process. For the first security level, SHAKE128 was used as an extendable-output function (XOF) based on the FIPS 202 standard [NIS23c]. We note that for higher security levels, it is necessary to adjust the parameters within the Keccak core accordingly. Nonetheless, the fundamental design of the hash sub-module remains applicable and does not require significant changes.

The Keccak implementation in [BDH+22] streams data utilizing a different format compared to the proposed MAYO hardware 32-bit format. To address this discrepancy, we developed a wrapper around the core. The reasoning behind this is that MAYO algorithm requires a hash of approximately 120KB for the key generation. The hash is eventually stored in the inner 32-bit-wide block memory.

The proposed architecture, as shown in Figure 3, stores the input seed and output message in separate descriptor-like registers. These intermediate registers are simultaneously accessed by the hash core and BRAM. Keccak can, in fact, access 64 bits, meaning a full descriptor at a time, while BRAM treats the descriptors as lower and upper halves and needs 2 read/write operations to fully manage a descriptor. This can create a synchronization problem between the hash core and the BRAM. To solve this issue, the descriptors are arranged to form a ring-shaped buffer, which can only be modified through indexing pointers. By doing so, we ensure a continuous flow of information within the BRAM, even while the hash module is still generating additional bytes.

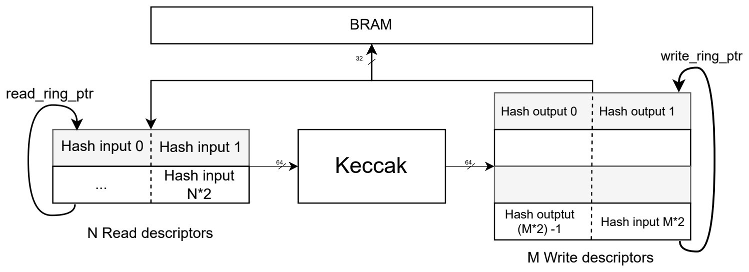

Figure 3: Block Diagram of the hash core; the descriptors are arranged to form a ring-shaped buffer which is simultaneously accessed by the hash core and BRAM

3.4 True Random Number Generator

For random number generation, we employed the neoTRNG [FI]. It uses ring-oscillators as a source of entropy. Although the TRNG is compact running on its' default configuration, a total number of 16 clock cycles is still required to generate *one* raw random byte. As a result, we developed a wrapper to reduce this output delay. The TRNG maintains a nearby First-In-First-Out (FIFO) buffer full. The MAYO Core reads exclusively from the FIFO, thus popping bytes that are once more replaced in the FIFO by the random number generator. This process helps eventually reduce the power usage, as the TRNG is not active at all times but only when the FIFO is *not* full anymore and is able to capture more random bytes.

3.5 Vector-Matrix Multiplication

The MAYO hardware implementation contains various arithmetic cores that are called multiple times by the main state machines of both IPs as shown in Figure 2. Upon analyzing the proposed MAYO pseudo code [BCC+23, Section 2], it becomes evident that matrix-vector multiplication proceeded by a $\mathbb{F}_q$ space reduction, is frequently utilized operations throughout the algorithm. Hence, its optimization will improve the performance of our design.

Compared to the initial MAYO Software C implementation[7], the vector-matrix multiplication iterates through a matrix stored in a row-wise manner, as seen in the left side of Figure 4, multiplying (using MULT operation) the content with a given series of coefficients and accumulating the results. Once this nested row/column loop concludes, another loop starts reducing the accumulated result through MOD operation. For instance, on an ARM Cortex-M3 with ARMv7-M instruction set, a single MULT operation with 8-bit operands takes around 2 to 3 clock cycles [ARM]. The reduction is done using the MOD operation that is usually translated to MULT and UDIV as Cortex-M3 lacks native modulo calculation. Consequently, the vector-matrix multiplication function could consume up to 6500 clock cycles, excluding the memory load and store operations.

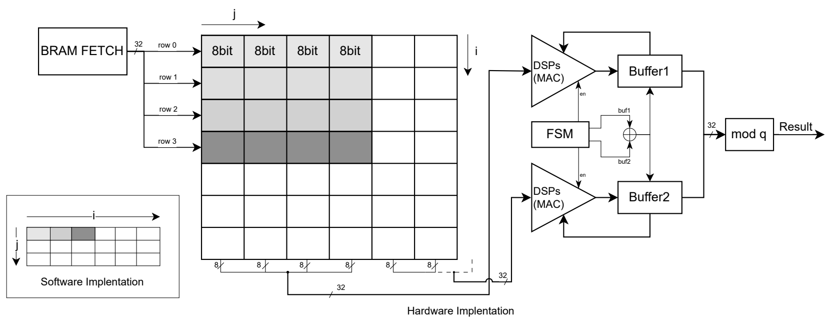

Figure 4: Matrix-Vector multiplication architecture; on the left side the vector-matrix multiplication iterates through a matrix stored in a row-wise manner as in the software implementation. In hardware design, we reversed the indexing order, and input four bytes to each DSP which executes 4 multiplications simultaneously.

In this paper, we process the multiplications differently. Firstly, our design offers four values on each memory read operation thanks to its 32-bit wide bus and executes 4 MULT operations from one row simultaneously. Secondly, we reversed the indexing of the input matrix, as shown in Figure 4.

[7] Note here that we refer to the first implementation of MAYO scheme by Ward Beullens in [Beu22b]

As matrices are stored row-wise, each memory access returns $four$ sequential cells from *one* row. Note that the matrix is stored in BRAMs and not in an FF-layered structure as might be understood from the figure below.

Furthermore, the input of both Digital Signal Processors (DSPs) is composed of 4 bytes. This architecture helps increase the throughput and enables the parallelization of both MULT and MOD operations.

Once the accumulated data of a block of four columns begins the final MOD operation, the subsequent block is fetched and starts with MULT operation, as shown in the timing diagram in Figure 5. The first row of the Matrix $\mathbf{M}$ and the first coefficient of the Vector $\mathbf{V}$ are fetched from the BRAMs. The read port then keeps feeding the system with blocks from each consequent row noted as $\mathbf{M}[rowIndex, columnBlock]$, until the accumulated result is ready to be stored through a different write-only port (WriteRES).

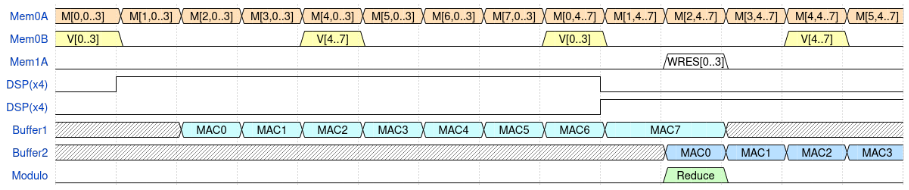

Figure 5: A simplified example showing the Matrix-Vector multiplication scheduling of $\mathbf{M}_{8\times 8} \times \mathbf{V} = \mathbf{RES}$, where $\mathbf{M}$ is a matrix of size 8×8, and $\mathbf{V}$ and $\mathbf{RES}$ are vectors of size 8. $\mathbf{M}$, $\mathbf{V}$, and $\mathbf{RES}$ are stored in Mem0A, Mem0B, and Mem1A, respecitively. $\mathbf{M}[rowIndex, columnBlock]$ corresponds to the row of index $rowIndex$ of the block having the index $columnBlock$. We mean by MAC the Multiply And Accumulate function.

The design encloses a number of DSPs to enhance the Multiply And Accumulate functionality (MAC) of the sub-module, therefore, achieving better timing results during synthesis. Finally, instead of using a full $M \times 32-$bit accumulate vector, we incorporate double-buffered temporary registers that are flushed alternatively at the end of each sequence. Similarly, this reduces the usage of FFs within the sub-module. Since the accumulated values are stored in BRAM once ready, its' FFs can accumulate the proceeding series instead of allocating extra registers.

3.6 Memory Organization

The hardware implementation of MAYO mainly relies on BRAMs to store its vectors and matrices. To ensure that both cores, namely the KeyGen and Sign, have sufficient stack-like memory, 98% of the available on-chip BRAM is allocated for the implementation. Thereby we provide enough headroom for potential parameter modification of MAYO that might increase memory usage, e.g., when changing the security level from 1 to 5, the expanded secret key size increases from 70KB to 557KB [BCC+23]. It is important to note, that not *all* the dedicated memory is utilized for the first security level. In fact, only roughly 60% of the 4.9MB Block Memory of the Zynq device is allocated with data.

The content of the BRAM cells is pre-allocated and statically organized since the sizes of most elements are pre-defined. In other words, all vectors and matrices' addresses are provided in a VHDL file to create a mapping. This file is then included in all sub-modules for better consistency.

The memory is partitioned into *three* dual port BRAMs, offering enhanced performance and flexibility. This configuration allows, for instance, efficient reading from one port while dedicating the other port for writing. Some sub-modules, such as vector-matrix multiplication, or vector addition, tend to utilize three ports for dual read and final write operations, therefore allowing better opportunities for parallelism within the sub-module. Moreover, the BRAMs offer the "write enable" signal to address bytes of a memory row separately.

A simplified example of the memory data space allocation is presented in Figure 6. Small buffers and vectors that are not meant to be accessed exclusively by programmable hardware are found in the smaller BRAM. The big BRAMs are indeed also shared with the CPU through AXI bus to stream input information such as the message and secret key to the MAYO core itself. Furthermore, since the key generation and the signing are not designed to operate synchronously but rather consecutively, multiple arrays and vector spaces overlap if one of their lifetimes expires. This approach helps the system avoid unnecessary increases in memory fingerprints.

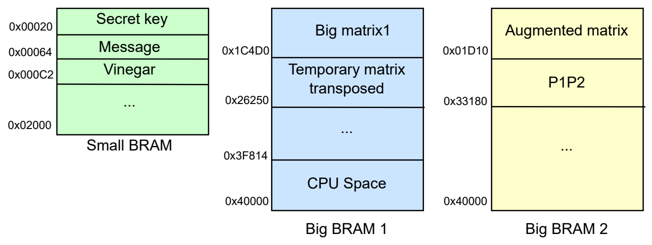

Figure 6: MAYO Block memory mapping presenting three True Dual Port BRAMs statically partitioned into 2 big BRAM (each is 256KB) for large matrices and vector storage, and small BRAM (of 8KB) for small scratch buffers and other sensitive information.

3.7 Gaussian Elimination

Solving a System of Linear Equations (SLE) is evidently one of the primordial computations for the MAYO algorithm to generate a valid message signature as explained in Section 2.3. Internally, the SLE is represented as a matrix $\mathbf{A}$ of coefficients that resides in one of the large BRAMs. After defining the rightmost $\mathbf{b}$ vector, the Gaussian elimination method is applied to determine the x_i unknowns of the vector $\mathbf{x}$ within the $\mathbf{Ax} = \mathbf{b}$ system. In this section, we present a hardware implementation specifically designed for this task.

The Gaussian elimination can be summarized as transforming the system into a similar triangular-shaped matrix to easily identify the solution. The initial step is to perform forward elimination to create the desired triangle shape. Starting from the first row, each following row is scaled and subtracted from the rows below it to eliminate the coefficients below the main diagonal. Rows can be interchanged to ensure that the largest coefficient a_{ij} is placed in the pivot position. Once the upper triangular matrix is obtained, back substitution is performed. Starting from the last row, the unknown variables are unraveled one by one.

Several publications deal with hardware implementation of Gaussian elimination for various cryptographic applications, primarily focusing on $\mathbb{F}_2$. Among them, GSMITH [REBG11] has been widely recognized for efficiently handling $\mathbb{F}_{2^k}$ equations. Unfortunately, GSMITH's architecture only conforms with small and medium-sized matrices, whereas MAYO's SLE $m \times m$ shaped matrix is larger. This quadratic shape depends on the NIST security level. Not only would the proposed GSMITH architecture utilize costly resources, but also hinder the overall architecture's performance and increase the needed Look-up Tables (LUTs) when targeting $\mathbb{F}_{31}$. GSMITH describes, in fact, a systolic network composed of various types of tiny processors capable of specific Gaussian steps and propagating its values. Yet, since the source code was not open-sourced, we had to redesign GSMITH. The final architecture, however, fails to meet our resource requirements, depleting the Zynq's FFs and LUTs, due to the internal registers required in each GSMITH processor and its' interconnection with the proposed BRAM. When considering the other needed arithmetic cores, we concluded it was unfeasible to fit GSMITH for the first security level.

To overcome this issue, we developed a state machine that fetches values directly from BRAM as the matrix is stored externally rather than within the core's FFs. Additionally, it was mandatory to allocate sufficient memory to accumulate every cell in the matrix. In other words, during the first step of the Gaussian elimination, multiplying rows with scalars may surpass the existing 8-bit limit. Hence, the targeted matrix is initially unpacked into 16-bit wide values with added padding, meaning that every row in the BRAM now contains two instead of four values.

Moreover, to speed up the mod-inverse, which calculates the needed value to transform the pivot element into 1 throughout the first scale step, prefilled Read-Only Memory (ROM) with end results of this operation is utilized instead of performing the actual calculations on run-time. These optimizations contribute to the overall effectiveness of the MAYO core in solving an SLE. Although GSMITH might offer superior performance, this core certainly consumes less resources and power. Our architecture is theoretically compatible with other configuration sets, with a marginal difference in resource utilization. For instance, n, m, o control the SLE size which should affect BRAM consumption, while q modifies the LUT consumption, cell width, and the unpacking operation. The solver should support up to $\mathbb{F}_{2^8}$ and for smaller q values, unpacking the matrix might become unnecessary, as the result could still fit inside the original 8-bit vector.

3.8 Optimizations and Firmware

Many sub-modules within our design share the same access to one of the BRAM ports. Nevertheless, the usage of each port, whether for reading, writing, or both differs. The core responsible for the vectors' addition, for example, features multiple modes depending on the location of the input vectors in different BRAMs. It manages to efficiently utilize all available ports to leverage data throughput and synchronize the addition process accordingly.

Another notable design optimization lies in the polynomial reduction sub-module where multiple arrays of scratch buffers are used to minimize memory interactions. Hence, the core is provided only with new values which are stored as final results.

	Resource Utilization		
Submodules	LUTs	FFs	DSP
Keccak (Hash)	10580	18448	0
TRNG	460	218	0
Vector-Matrix multiplication	1035	528	8
Oil Space Sampling	176	289	0
Gaussian Elimination	1822	413	3
Vector Addition	485	300	0
Vector Negation	176	93	0
Vinegar Sampling	245	277	0
BRAM Port management	448	9	0
FSM Signing	2874	1068	0
Combined Architectures	23356	24645	11

Table 1: Resource utilization of our hardware design on Zynq 7020 at a frequency of 100 Mhz

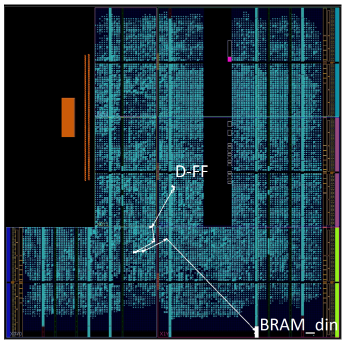

Figure 7: Zynq Design CLB utilization and mapping diagram showing a worst timing path

Various functionalities of MAYO are divided into separate modules, each described individually. That said, each module still has access to header-like files that declare the security level parameters, the memory space allocations, utility functions required to fetch offsets or even ROM secret keys specifically intended for non-debugging purposes. Numerous bit vectors are built upon these constants. The code's style guide itself heavily discourages simple number inclusion, but instead, it is expected to utilize these pre-defined macro-like lines to improve code readability and ensure that the overall architecture can fit different configuration sets, i.e., different security levels.

In addition to the hardware implementation, the utilization of the MAYO core necessitates the development of accompanying firmware. This firmware serves as the interface between the hardware core and the software MAYO application, setting AXI/AXI-Lite transactions up. The existing C Bit-fields feature Control and Status Registers that can enable debug mode, interrupts, and supply the ARM CPU with the end of executions information besides the interrupt signal.

4 Results, Comparison, and Discussion

4.1 Results

In Table 1, we show the resource consumption of the whole design and submodules for the first security level defined by the NIST PQC standardization process [NIS23b]. The parameters defining MAYO are q (the size of the finite field), n (the number of variables in the multivariate quadratic polynomials in the public key), m (the number of multivariate quadratic polynomials in the public key), o (the dimension of the oil space), and k (the whipping parameter, satisfying $k < n - o$). For our results, these parameters are set to $q = 31$, $n = 62$, $m = 60$, $o = 6$, and $k = 10$.

The proposed design uses approximately 50% of the available logic resources on the Zynq board. These resources are distributed among different sub-modules.

Keccak The Keccak core occupies the most slices, which cover almost half of the design. This is primarily due to its wide internal buffer and the tightly interconnected XOR network to generate the output hash. The descriptor read/write wrappers contribute to this massive FF utilization, as we employ 4 read and 20 write descriptors to mitigate synchronization issues between Keccak and the rest of the cores.

Gaussian Elimination Compared to the rest of the sub-mdoules, the Gaussian elimination unit stands out with the longest state machine, as it involves various steps and is designed to handle bigger matrices.

Vector-Matrix Multiplication The vector-matrix multiplication core also consumes a considerable amount of FFs and 8 DSPs due to the optimization techniques explained in Section 3.5.

FSM Signing The FSM Signing serves as the main control unit inside the overall design and occupies 5% of the LUTs. This FSM combines multiple functionalities, including key expansion, resulting in higher resource utilization. The system also utilizes 97.14% of the BRAMs on the board to incorporate all the vectors and matrices required by MAYO in a cohesive storage space, as explained in Section 3.6.

Data Flow Management Additionally, port and data flow management constitute an important part of our design. The summed multiplexers and various managers used in the implementation can consume up to 6% of the LUTs. For instance, the BRAM port manager is employed 5 times in the design, one for each BRAM port. These buses are crucial to facilitate arithmetic core inclusion and communication. However, they affect the amount of combinatorial logic residing between the start-FF and end-FF at each clock, hence, they introduce an additional delay.

We manage to synthesize the design at 100 MHz. However, any changes to reach higher frequencies are constrained by the inherent limitations of the BRAM-to-Submodule data transmission system and the associated net delay. On average, it takes approximately 5 ns for a data signal to reach the target core, leaving only about 4.5 ns (half a clock cycle) for logic operations. As illustrated in Figure 7, the worst path of the design starts indeed at an FF and ends at a BRAM port. Despite attempts to address congestion through P&R, the target BRAM slice remains the furthest from the programmable logic.

4.2 Discussion and Comparison

Though our design consumes about 2 W of energy and utilizes less than half of the available resources (24K FFs, 25K LUTs), Figure 7 indicates that the implementation is not compiled for efficient on-chip surface management but rather for better performance and routing. This explains the resource distribution throughout all slices of the FPGA. Since MAYO operates on diverse data widths, synthesis necessitates the use of various logic elements (such as LUT3, LUT4, LUT5, LUT6) which might not be located close to each other. One key aspect to modify the frequency lies in incorporating more idle cycles to account for the 2-clock cycle BRAM read delay. Moreover, incoming data is often stored in a nearby FF before applying any further operations. Therefore, we estimate that almost 30% of the total clocks are spent on memory operations (i.e., data fetch and storage). Therefore, fine-tuning the BRAM allocation by only dedicating the necessary amount for the targeted NIST security level or reducing q to 16 can clearly improve the implementation speed.

	Software Reference Implementation	Software Implementation with AVX Optimization		Optimized Hardware Implementation	
Scheme	MAYO	UOV	MAYO	UOV	MAYO
Source	[Beu22b]	[BCH+23, Table 3]	[Beu22b]	[BCH+23, Table 21]	Our Work
KeyGen	993,959	4,450,838	733,310	11,072,933	996,098
Sign	4,485,915	2,473,254	2,250,601	843,885	3,491,998

Table 2: Comparison with other hardware and software implementations

We present in Table 2 a comparison of the performance of different multivariate schemes' implementations, i.e., the software and hardware implementation of

UOV [BCH$^+$23], two software implementations of MAYO [Beu22b], and our implementation. Note that all the implementations correspond to the first security level. We compared with the UOV parameter set 'ov-Is-pkc+skc' from [BCH$^+$23], since MAYO also uses the technique to compress public and secret keys, which implies the requirement for key expansion before signing.

Our implementation demonstrates improved efficiency when compared to the reference software implementation of MAYO. This is due to the optimization of frequently used functions, such as mod-inverse in Gaussian elimination and the matrix-vector multiplications, resulting in a similar number of cycles for the key generation and a speedup of approximately 22% for signing as compared to the reference C implementation.

In comparison to the MAYO AVX optimized implementation, our work has an increase in cycle counts for key generation and for signing. This can be explained by the heavy impact of memory operations, which can be further optimized in software implementations. Observe from the cycle counts of the UOV key generation, that hardware implementations are not generally faster than optimized software implementations.

Compared to the hardware implementation of UOV, our implementation has considerably fast key generation, but slower signing. Even though we do not completely match the exceptionally good performance of the highly optimized hardware implementation of [BCH$^+$23], we are still in a compatible range. By moving to the newest parameters that use $q = 16$, further improvement can be expected.

The overall good performance of our implementation can also be attributed to the simplified memory design minimizing unwanted delays caused by CPU-Cache-RAM architecture. In addition, our implementation significantly enhances the performance of the scheme by utilizing pipelining techniques that process four values simultaneously. This stands in contrast to the non-optimized C implementation, which handles each value individually.

5 Conclusion

The implementation of multivariate signature schemes has faced challenges due to their large key sizes, impeding them from deployment on resource-constrained embedded devices. In response, the MAYO scheme was developed as a new modification of the mature UOV signature scheme. MAYO has successfully addressed the issue of large key sizes and can now be seen as one of the prominent candidates of NIST's call for additional digital signatures in regard of performance, key, and signature size. In this paper, we introduced the first full hardware, optimized, and reconfigurable implementation of MAYO. Our implementation serves as evidence of MAYO's practicality for real-world deployment. Notably, we deliberately chose a "modest" board that lacks sophisticated hardware encryption capabilities and extensive memory storage, demonstrating the ease of deploying the MAYO scheme.

However, the deployment of the MAYO scheme brings forth a set of new security challenges, particularly in terms of defending against side-channel attacks such as [AKKM22, ACK+23]. Therefore, we emphasize the importance of strengthening the security of our implementation against potential side-channel attacks.

References

ACK+23. Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. Separating Oil and Vinegar with a Single Trace: Side-Channel Assisted Kipnis-Shamir Attack on UOV. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 221–245, 2023.

AKKM22. Thomas Aulbach, Tobias Kovats, Juliane Krämer, and Soundes Marzougui. Recovering Rainbow's Secret Key with a First-Order Fault Attack. In *International Conference on Cryptology in Africa*, pages 348–368. Springer, 2022.

ARM. ARM. Armv7-m architecture reference manual. https://developer.arm.com/documentation/ddi0403/d/Application-Level-Architecture/The-ARMv7-M-Instruction-Set.

BCC+23. Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias Kannwischer. Mayo-algorithm specifications. mayo team, latest update: 28/02/2023 (2023). https://pqmayo.org/assets/specs/mayo.pdf, 2023.

BCH+23. Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and Vinegar: Modern Parameters and Implementations. *Cryptology ePrint Archive*, 2023.

BDH+22. Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak open-source hardware implementation. https://keccak.team/index.html, 2022.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy Signature Schemes. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part III*, pages 183–211. Springer, 2020.

Beu21. Ward Beullens. Improved cryptanalysis of UOV and Rainbow. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, pages 348–373. Springer, 2021.

Beu22a. Ward Beullens. Breaking Rainbow Takes a Weekend on a Laptop. In *Annual International Cryptology Conference*, pages 464–479. Springer, 2022.

Beu22b. Ward Beullens. MAYO: Practical Post-Quantum Signatures from Oil-and-Vinegar Maps. In *Selected Areas in Cryptography: 28th International Conference, Virtual Event, September 29–October 1, 2021, Revised Selected Papers*, pages 355–376. Springer, 2022.

BMSV22. Emanuele Bellini, Rusydi H Makarim, Carlo Sanna, and Javier Verbel. An Estimator for the Hardness of the MQ Problem. In *Progress in Cryptology-AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18–20, 2022, Proceedings*, pages 323–347. Springer, 2022.

CKPS00. Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings*, pages 392–407. Springer, 2000.

DS05. Jintai Ding and Dieter Schmidt. Rainbow, a new Multivariable Polynomial Signature Scheme. In *ACNS*, volume 5, pages 164–175. Springer, 2005.

Fau99. Jean-Charles Faugere. A new Efficient Algorithm for Computing Gröbner Bases (F4). *Journal of pure and applied algebra*, 139(1-3):61–88, 1999.

FG18. Ahmed Ferozpuri and Kris Gaj. High-speed fpga implementation of the nist round 1 rainbow signature scheme. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2018.

FI. Fraunhofer-IMS. A Tiny and Platform-Independent True Random Number Generator for any FPGA. https://github.com/stnolting/neoTRNG.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86: Proceedings 6*, pages 186–194. Springer, 1987.

HRSS16. Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass mq-based identification to mq-based signatures. *IACR Cryptol. ePrint Arch.*, 2016:708, 2016.

HZ18. Yi Haibo and Nie Zhe. High-speed Hardware Architecture for Implementations of Multivariate Signature Generations on FPGAs. In *EURASIP Journal on Wireless Communications and Networking*, pages 1687–1499, 2018.

Int. Intel. QDR II SRAM controller intel® FPGA IP function. https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/memory-interfaces-and-controllers/qdrii.html.

KPG99. Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar Signature Schemes. In *Eurocrypt*, volume 99, pages 206–222. Springer, 1999.

KS06. Aviad Kipnis and Adi Shamir. Cryptanalysis of the Oil and Vinegar Signature Scheme. In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings*, pages 257–266. Springer, 2006.

NIS23a. NIST. NIST post-quantum cryptography standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline, 2023.

NIS23b. NIST. NIST post-quantum cryptography standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria), 2023.

NIS23c. NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. https://csrc.nist.gov/publications/detail/fips/202/final, 2023.

Pat97. Jacques Patarin. The Oil and Vinegar Signature Scheme. In *Presented at the Dagstuhl Workshop on Cryptography September 1997*, 1997.

PQD23. PQDB data base. https://www.pqdb.info/, 2023.

REBG11. Andy Rupp, Thomas Eisenbarth, Andrey Bogdanov, and Oliver Grieb. Hardware SLE solvers: Efficient building blocks for cryptographic and cryptanalytic applications. *Integration*, 44(4):290–304, 2011.

TYD+11. Shaohua Tang, Haibo Yi, Jintai Ding, Huan Chen, and Guomin Chen. High-speed Hardware Implementation of Rainbow Signature on FPGAs. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*, pages 228–243. Springer, 2011.

Xil23a. AMD Xilinx. Vivado ML 2022.2. https://www.xilinx.com/products/design-tools/vivado.html, 2023.

Xil23b. AMD Xilinx. Zynq-7000 SoCs with Hardware and Software Programmability. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, 2023.

XL21. Yufei Xing and Shuguo Li. A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):328–356, Feb. 2021.

ZZW+21. Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):270–295, Nov. 2021.