A Configurable Hardware Implementation of XMSS*

Jan Philipp Thoma[1] and Tim Güneysu[1,2]

[1] Ruhr University Bochum, Horst Görtz Institute Bochum, Germany
jan.thoma@rub.de, tim.gueneysu@rub.de
[2] DFKI GmbH, Cyber-Physical Systems, Bremen, Germany

Abstract. As a fundamental building block in today's digital world, Digital Signature Schemes (DSS) provide the ability to authenticate messages exchanged over untrusted channels. Unfortunately, virtually all currently used DSS are built upon mathematical problems that can efficiently be solved using quantum computers, thus rendering schemes such as RSA and ECC insecure. Due to its conservative security properties, the eXtended Merkle Signature Scheme (XMSS) is an outstanding candidate for a quantum-secure DSS which has already been standardized by NIST and IETF.
In this paper we present the first full hardware accelerator for XMSS whose generic design approach allows matching the requirements of several projected use-cases. In particular, we provide a full design exploration regarding the choice of parameters and hash functions to identify configurations for optimal performance and area utilization.

Keywords: XMSS · Hardware Implementation · Digital Signatures.

1 Introduction

Today's digital signature algorithms are mostly built on hard mathematical problems such as prime factoring of large numbers or the discrete logarithm problem. These mathematical problems have been thoroughly studied for decades and are hence genuinely believed to be hard on a classical computer, i.e. they cannot be solved in polynomial time. Cryptographic schemes based on these mathematical problems, such as RSA, ElGamal, Diffie-Hellmann and DSA are widely deployed over a large range of devices. However, this pleasant story of success is about to be rudely interrupted by the introduction of large-scale quantum computers. In contrast to classical computers, quantum computers are able to perform *quantum parallelism* [7] which is useful to speedup some probabilistic algorithms. As a consequence to this threat, many institutions are currently working on standardizing post-quantum cryptography, laying the foundation for industry to adapt these algorithms.

* The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research (BMBF) under the project "QuantumRISC" (ID 16KIS1038) [20] and project "PQC4MED" (ID 16KIS1044).

In this work we will investigate XMSS which was recently standardized by the IETF [12] and is recommended by NIST [6] and the Federal Cyber Security Authority of Germany (BSI) [2]. In addition to the standardization efforts by federal agencies, a security proof has been given in [4], showing that the security requirements of XMSS are minimal, i.e. *if any digital signature scheme exists at all, there exists a secure instantiation of XMSS.*

Generally speaking, post-quantum cryptography is more expensive in terms of computational complexity, be it due to larger key and signature sizes or due to more complex arithmetic operations. Hence, for many use cases the requirements to expected latency or computing capacities do not fit the new algorithms. Dedicated hardware accelerators like the one presented in this work can be used to increase the performance of the schemes or outsource the computation from the main processing unit. Since the cryptographic primitives are in an early stage of standardization, hardware accelerators face the challenge of agility - if the parameter set changes throughout the process, be it due to security considerations or practicality, the hardware accelerators must be adapted with high effort. We address this challenge by making our hardware implementation highly configurable and easily adaptable to algorithmic changes without the need of a complete re-implementation. The configurability includes many XMSS parameters like the tree height, the WOTS chain length and most importantly the used hash algorithm which is easily exchangeable. We further provide a *cryptoagile* configuration that allows selecting the hash algorithm at runtime. All files of the hardware implementation are available at `https://github.com/Chair-for-Security-Engineering/XMSS-VHDL`

1.1 Our Contributions

We present the first full-fledged hardware accelerator for the emerging quantum secure signature standard XMSS. As part of our work we conduct an in-depth design exploration to investigate optimal performance and area results for different possible use-cases and configurations. Our final design proposal is highly configurable to adapt to various use-cases, including an agile instantiation for long-term security in case one of the deployed hash algorithms should become obsolete. The configurability, that is inherent part of our design, enables a large spectrum of time/area trade-offs.

1.2 Related Work

XMSS was introduced in 2011 as a quantum-secure digital signature scheme with minimal security assumptions in [4] and has since been standardized by the IETF in RFC 8391 [12]. The scheme is closely related to Leighton-Micali Signatures (LMS) which is also standardized by the IETF [16], tough LMS does not have such minimal requirements for security. A comparison between the two schemes has been conducted in [5, 13]. The side-channel security of XMSS has been studied in [14] which was improved upon in [22]. A hardware/software co-design of XMSS using the RISC-V architecture was implemented in [21]. The

entirely hardware-based accelerator of this work, however, can specifically adapt its performance to the individual use-cases.

There are several other proposals for quantum secure signature schemes, most of which are part of the NIST post-quantum cryptography (PQC) competition [18]. In [19], a hardware accelerator for the number theoretic transform is implemented using high level synthesis. This accelerator can be utilized to speed up quantum-secure signature schemes such as Falcon [11] and Crystals-Dilithium [9]. In [1], an agile crypto co-processor is implemented that can accelerate various lattice based schemes. The multivariate Rainbow signature scheme was presented in [8] and implemented in hardware in [10].

1.3 Organization

The remainder of this paper is structured as follows: In Section 2 we introduce the scheme and its underlying algorithms. Key features of our implementation are presented in Section 3. Then, we evaluate the implementation in Section 4 with focus on different target use cases. We provide both performance figures as well as data on the area utilization and compare our implementation against related work. Finally, we conclude in Section 5.

2 Preliminaries

In this section, we introduce a variant of the Winternitz One-Time Signature (WOTS) scheme dubbed WOTS+ and then present XMSS which uses WOTS+.

2.1 Winternitz One-Time Signatures

Most hash-based signature schemes build upon one-time signatures, such as the Winternitz One-Time Signature (WOTS) scheme [17] which originates in 1989. The WOTS scheme improves upon the Lamport Diffie one-time signature scheme [15].

The WOTS scheme is defined by a Winternitz parameter w and the digest length n of the cryptographic one-way function $H : \{0,1\}^* \rightarrow \{0,1\}^n$. To sign a n-bit message, it is first split into len_1, w-bit blocks such that $m = m^{(1)}||...||m^{(len_1)}; m^{(i)} \in \mathbb{Z}_w \forall i \in \{1, ..., len_1\}$ where

$$len_1 = \left\lceil \frac{n}{lg(w)} \right\rceil \tag{1}$$

Additionally, a checksum is computed over the message and converted into a w-bit representation as follows

$$c = \sum_{i=1}^{len_1} w - 1 - m^{(i)} = c^{(1)}||...||c^{(len_2)}; c^{(i)} \in \mathbb{Z}_w \tag{2}$$

The length of the checksum in base-w representation len_2 computes as

$$len_2 = \left\lfloor \frac{lg(len_1 * (w-1))}{lg(w)} \right\rfloor + 1 \tag{3}$$

We define $m' = m^{(1)}||...||m^{(len_1)}||c^{(1)}||...||c^{(len_2)}$ as the concatenation of the message and the checksum in base-w representation. The WOTS secret key consists of $len = len_1 + len_2$ randomly chosen n-bit values. The signature and public key are generated by applying the one-way function $H(\cdot)$, $m^{(i)}$- respectively w-times to the secret key, thus generating a *WOTS chain*. Hence, both the signature and the public key consist of len pseudorandom n-bit values. The secret key is the start of the chain, the signature an intermediate value and the public key the last value form the chain.

To verify a WOTS signature, the verifier applies $H(\cdot)$, $w - m'^{(i)} - 1$-times to the signature for each $i \in \{1, ..., len\}$ which yields the public key if and only if the signature is valid.

A variant of the WOTS scheme dubbed WOTS+ has been introduced in [3]. A pseudorandom function (PRF) is used to generate a unique hash key and bitmask during each iteration of the WOTS+ chaining. The process is depicted in Figure 1. To generate the key and the bitmask, the PRF is used with a public seed

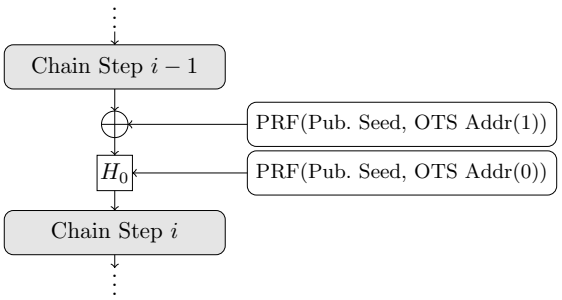

Fig. 1. Chain iteration in WOTS+. Each hash operation is keyed and obfuscated using a PRF.

and a 256-bit OTS address which is defined in the XMSS standardization [12]. The address differs in one bit which per definition of a PRF yields completely unrelated pseudorandom results.

Each WOTS key must only be used to sign one message. Using a key for more than one signature enables trivial attacks as the attacker then knows intermediate values for each of the WOTS chains, giving him/her some degrees of freedom to modify the signature while still maintaining its validity

2.2 XMSS

The eXtended Merkle Signature Scheme (XMSS) is based on Merkle Signatures [17] which construct a *many-time* signature scheme from a one-time signature using a balanced binary tree. The tree structure of XMSS is shown in Figure 2. Each leaf node of the tree structure holds a WOTS+ public key that is com-

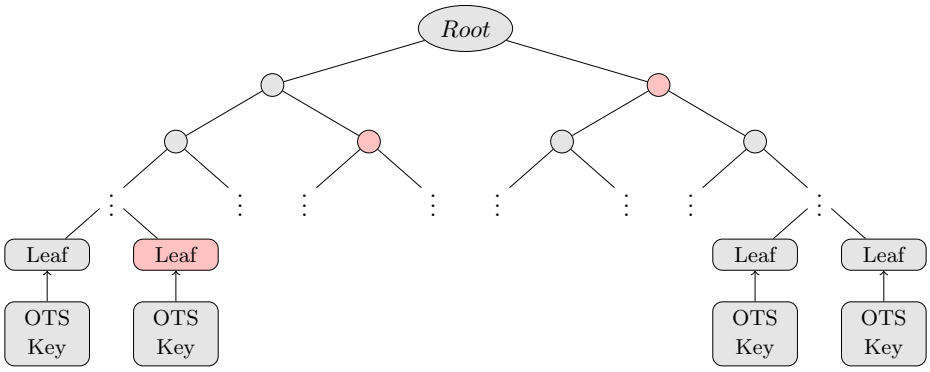

Fig. 2. Balanced binary tree structure of XMSS. The authentication path nodes for the leftmost leaf node are marked in red.

pressed to a n-bit value. To achieve this, each $len * n$-bit WOTS+ public key is represented as an unbalanced binary tree (L-Tree) where the leaf nodes contain a single n-bit key value each. The L-Tree hence has len leaf nodes. Then, two sibling nodes from the L-Tree are iteratively compressed using the hash function, yielding a single root node of the L-Tree which is used as leaf node in the XMSS Merkle Tree. Similar to the WOTS+ structure, the hashing of sibling nodes in the L-Tree makes use of a keyed hash function as well as bitmasks to randomize the hash function input. In each instance, first a n-bit key is pseudorandomly generated using the PRF, followed by two pseudorandom n-bit bitmasks which are *xor*-ed to one of the sibling nodes each. The extensive use of pseudorandom keys and bitmasks throughout XMSS is attributed to the minimal security requirements mentioned beforehand. The generation of the XMSS root node follows the same procedure as for the L-Trees: Two sibling nodes are hashed using a pseudorandom key and a bitmask to generate the parent node. XMSS distinguishes between PRF operations and normal hash operations by prepending an unambiguous padding before each operation of the hash function.

XMSS is a *stateful* signature scheme since it must be ensured that each leaf node is used only once to sign a single message. The state (i.e. the current leaf node index) is part of the XMSS secret key. Additionally, the secret key contains a seed used to pseudorandomly generate the WOTS+ seeds and a n-bit value SK_PRF which is used as a hash key in the signing algorithm to randomize the

message hash. The public key holds the root node of the balanced binary tree as well as a n-bit public seed.

During the key generation of XMSS, the binary tree is constructed. Hence, 2^h WOTS+ public keys are generated, where h is the tree height. This is done pseudorandomly using the seed which is part of the XMSS secret key. To sign a message of arbitrary length, it is hashed together with a n-bit randomization value R as well as the leaf index of the one-time key. Then, the WOTS+ signature of the resulting n-bit message hash is computed. Finally, on each level of the tree, the sibling node of the node located directly on the path to the root node is stored as part of the authentication path (see Figure 2). The signature consists of the $len * n$-bit WOTS+ signature, the leaf node index, the n-bit randomization value R, and the $h * n$-bit authentication path. To verify a message, the verifier generates the randomized message hash using the leaf index and the randomization value R. Then s/he computes a candidate for the root node of the tree using the authentication path from the signature. Only if the computed root node candidate matches the root node from the XMSS public key, the signature is valid.

3 Configurable and Efficient Hardware Implementation

We now provide the details for our XMSS hardware accelerator. The main goal of our implementation is a universal and agile design, offering high performance. The development of hardware accelerators for hash-based signature schemes has not been investigated yet in favor of adding a hardware-accelerated hash core which achieves some speed-up compared to a pure software solution. However, our implementation can easily be configured with an arbitrary number of hash cores and WOTS+ chaining modules, allowing increased performance by true parallel computation of WOTS+ chains. We implement a low overhead hash bus design that allows optimal utilization of all configured hash cores to address various use-cases. Moreover, we provide a simple interface between the hash core and the bus system, such that it is easy to implement different hash algorithms.

In the provided implementation, a global configuration file allows switching the hash algorithm, setting the block RAM (BRAM) address layout, configuring the XMSS tree height as well as the Winternitz parameter w and defining the number of hash cores and WOTS+ chaining modules present. We further implement support for cryptoagile versions of XMSS, allowing to instantiate multiple hash algorithms in parallel and configuring the implementation during runtime. Finally, we implement a embedded-verification unit that strips all logic needed for key generation and sign to create a low-area verification unit.

3.1 Selection of Parameters

Following the standardization document [12], we implement the SHA2-256 hash function as a primary candidate. Furthermore, we implement the optional SHAKE-128 extendable-output function (XOF) to demonstrate the feasibility of a simple

exchange of hash algorithms. The XMSS tree height is freely configurable which includes the mandatory values 10, 16 and 20. The Winternitz parameter can be chosen as any power of two.

3.2 WOTS+ Chain Parallelism

XMSS is strongly dependent on the performance of the deployed hash core. All bitmasks and keys used throughout XMSS are generated pesudorandomly using the hash function which results in a high capacity utilization of the hash core. However, simply adding a large number of hash cores is not sufficient to parallelize the scheme effectively since most of the hash operations depend on the result of prior operations. For example, during the WOTS+ chaining algorithm, a bitmask and the hash key are generated using the hash core in PRF mode. The next hash operation takes the key and some value obfuscated by the bitmask as input. Since this operation depends on the previously generated key and bitmask, the implementation would need to stall regardless of the amount of hash cores available. This problem can be solved by computing multiple WOTS+ chains in parallel, yielding a much improved utilization of the available hash cores. Write collision between the chaining modules on the hash bus are prevented using time-division multiplexing (TDM).

3.3 Signature Generation

The key generation is the computationally most expensive part of XMSS. That is, since all WOTS+ public keys need to be generated to compute the root node of the Merkle Tree. In a naive implementation, one would need to generate the whole tree during each sign operation in order to compute the authentication path for a given leaf node. To thwart this necessity and speed up the signature generation by several orders of magnitude, our implementation stores the compressed leaf nodes during key generation in BRAM. Naturally, this solution does not scale for arbitrary tree heights since the amount of memory is limited. However, at a tree height of 20 which is the maximum height standardized by the IETF, the required storage for the leaf nodes is $2^{20} * 32B \approx 35MB$. This is still feasible on many FPGA boards.

3.4 Constant Time

The minimal configuration of the key generation and signature generation is isochronously designed, operating with one hash core and one WOTS+ chain. Most importantly, the time needed for signature generation does not depend on the message since regardless of the message, the complete WOTS+ chain is computed. For configurations with multiple hash cores and WOTS+ chains, collisions on the bus may lead to minor delays due to arbitration. Such a collision occurs for example, if two hash cores try to request the next message block at the same time. These timing deviations lay in the nature of a small bus architecture but are not related to the message and thus do not incur any timing issues.

4 Evaluation

We now evaluate our implementation for different use-cases. For all configurations, we fix a XMSS tree height of 10 which allows 1024 messages to be signed. Larger tree heights are configurable in the global parameter file.

4.1 Use-Case: Time/Area Tradeoff

First, we target an optimal time/area trade-off using the SHA2-256 hash core. We first measure the performance of the XMSS implementation with up to 6 hash cores and WOTS+ chains. This is multiplied by the lookup table (LUT) utilization, yielding a time/area product. The results are shown in Figure 3. We performed the same analysis for the flip-flop-based area utilization which lead to a similar result. The figure shows an optimal configuration for the configuration with three hash cores and two WOTS+ chains. We ran the place-and-route algo-

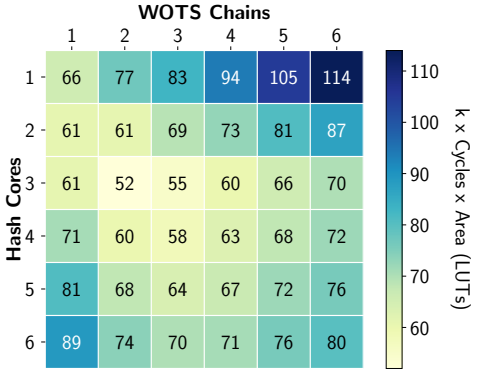

Fig. 3. Time (cycles) / area (LUTs) product for up to 6 hash cores and WOTS+ chains. (H=10, w=16)

rithm with an Xilinx Artix-7 FPGA as target device which is the recommended hardware platform by the NIST. The configuration allows clock frequencies of up to 100 MHz, utilizes 12,463 LUTs and 6,525 flip flops. The key generation takes 1.68 seconds, sign takes 4.98 ms and verify about 1.01 ms, depending on the message.

4.2 Use-Case: High Performance

For a performance-oriented version, we first measure the latency of the XMSS sign algorithm for configurations with up to 6 hash cores and WOTS+ chains. The resulting heatmap is depicted in Figure 4. It shows that configuring multiple

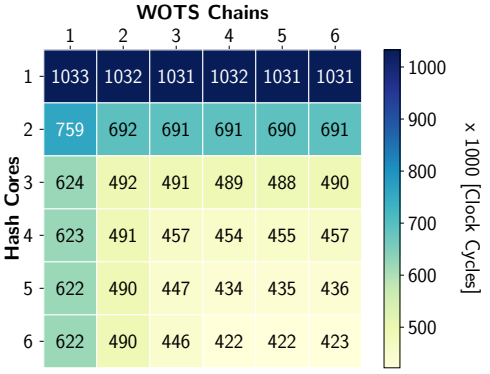

Fig. 4. Heatmap for the latency (measured in 1000 clock cycles) for single sign operation in XMSS with varying number of hash cores and WOTS+ chains using SHA2-256. (H=10, w=16)

WOTS+ chains with only one hash core does not improve the performance. That is, since the hash core is used heavily throughout the WOTS+ chaining algorithm and hence, such configurations need to stall frequently. The addition of a second hash core drastically improves the performance by more than 25%. At this point, the effect of parallel WOTS+ chain computation starts showing, gaining another 9% performance compared to the configuration with two hash cores and one WOTS+ chain. A further observation is, that using many parallel hash cores is not sufficient to accelerate the sign algorithm when only one WOTS+ chain is configured. That is, due to dependencies within the hash operations of the WOTS+ chains which limit parallelism in hash computation.

While the performance of our implementation generally increases the more hash cores and WOTS+ chains are configured, the effectiveness of additional hash cores and WOTS+ chains fades for a larger number of cores and chains. That is, since the sign operation of XMSS consists of two phases: First, a WOTS+ signature is generated and secondly, the Treehash authentication path is computed. The WOTS+ part can efficiently be parallelized using multiple WOTS+ chains and a sufficiently large number of hash cores. That is, since the WOTS+ chains are independent of each other. The second part of the algorithm allows only limited parallelization and hence limits the overall speedup during signature generation.

Although larger - and hence faster - configurations are possible with our implementation, for means of demonstration we choose the configuration with 9 hash cores and 7 WOTS+ chains as exemplary performance-oriented version which takes about 400,000 clock cycles to complete a sign operation. The configuration can be placed on a Xilinx Artix-7 FPGA using 30,454 LUTs, 15,191 flip flops and a maximum clock frequency of 95 MHz. The key generation takes 0.77

seconds, the sign algorithm 1.32 ms and signature verification approximately 561.01 us depending on the message.

4.3 Use-Case: Embedded Verification

Next to our main implementation that can accelerate all XMSS functions, we provide a minimal version that can only verify signatures. Therefore, we remove all unused modules as well as the logic that is used to switch between different modes in the original XMSS implementation. This version addresses the common use case, where a signature needs to be verified by a low-cost embedded device, e.g., for firmware updates. Though the implementation is equally configurable regarding hash cores, WOTS+ chains and XMSS parameters, in this setting we focus on a very low area footprint to especially address that use case. For the verification, the XMSS signature is expected in BRAM according to the layout defined in the global parameter file.

The embedded verification unit utilizes 4,596 LUTs and 2,646 flip flops when placed on an Artix-7 FPGA. We used area optimized synthesis tools which is why the maximum clock frequency is at 91 MHz. In our tests, the verification of a signature on average takes approximately 2.92 ms for random messages.

4.4 SHA vs. SHAKE

Additionally to the SHA2-256 hash function, we implement SHAKE-128 as defined as an optional XOF in the standardization [12]. For XMSS the most relevant input lengths to the hash function are 768 and 1,024 bits. For SHA2, messages with 768 bit length require two blocks whereas 1,024 bits inputs require 3 hash blocks. Hence, for 768 bit messages it takes 132 clock cycles for the hash output to be present whereas for 1,024 bit messages it takes 197 clock cycles. The SHAKE-128 hash core we use in our implementation is significantly faster, requiring 33 clock cycles for messages with up to 1,344 bit length. This performance gap has a direct impact to the performance of the overall XMSS scheme as shown in Figure 5. The performance of the sign algorithm is approximately quartered for all configurations compared to the SHA2 version (c.f. Fig. 4). The overall characteristic regarding the interplay of WOTS+ chains and hash cores remains similar.

It has to be noted that the SHAKE-128 hash core is much larger than the SHA2-256 core. We placed the time / area tradeoff variant (which also yields three hash cores and two WOTS+ chains for SHAKE) on an Artix-7 FPGA. The implementation utilizes 20,651 LUTs and 12,368 flip flops, allowing a maximum clock frequency of 95 MHz. The key generation takes 0.48 seconds, signing takes 1.32 ms and verification approximately 243.86 us.

4.5 Crypto-Agile Configuration

The crypto-agile configuration allows simultaneous placement of multiple hash cores using different algorithms and hence, runtime configurability of the algorithm. This is especially desirable for products with expected long life cycles as

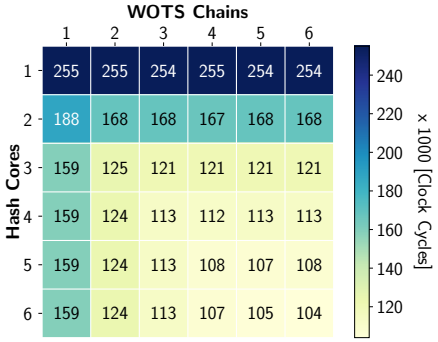

Fig. 5. Heatmap for the latency (measured in 1000 clock cycles) for single sign operation in XMSS with varying number of hash cores and WOTS+ chains using the SHAKE-128 hash function. (H=10, w=16)

the crypto-agility allows reacting to new attacks on one of the hash algorithms. While the performance is unaffected by this, the parallel instantiation of different hash cores naturally affects the area utilization.

We implemented a crypto-agile version using one SHA2 core, one SHAKE-128 core and one WOTS+ chain on an Artix-7 FPGA. The design utilizes 13,373 LUTs and 7,026 flip flops and can be run at 100 MHz. The timing results are equal to the minimal configurations using SHA and SHAKE respectively (c.f. Table 1).

4.6 Comparison

We gather the performance and area figures for each configuration in Table 1. We further compare the results to the hardware/software co-design approach form [21].

The results demonstrate the flexibility of our implementation. The minimal SHA2 version using one hash core and one WOTS+ chain is roughly half the size of the time/area tradeoff configuration. By stripping the logic required for key generation and sign, we derived an embedded verify accelerator that is more than 30% smaller than the minimal configuration performing all XMSS functions. In general, the SHAKE-128 configurations are much larger than the SHA2 counterparts which is due to the increased size of the hash core itself. Our implementation makes it easy to replace the hash core implementation and hence, yields more opportunities to achieve a specifically tailored implementation for each use-case.

The performance of the minimal version of our implementation featuring only one hash core is comparable to the hardware/software co-design presented in [21], though our implementation is slightly slower in key generation but much faster in verification. It has to be noted that [21] uses an XMSS-optimized SHA2

Table 1. Comparison between all configurations in area utilization and performance. The minimal configurations are equipped with one hash core and one WOTS+ chain. The HW / SW co-design from [21] uses a XMSS optimized version of SHA2 in contrast to our general purpose hash accelerator. All variants use h=10, n=32, w=16.

	Logic		**Memory**		**Usage**	**FMax**	**Gen**	**Sign**	**Verify (avg.)**
	LUT	MUX	FF	BRAM	Slices				
HW/SW [21] (SHA*)	-	-	-	-	-	93 MHz	3.44 s	9.95 ms	5.68 ms
Minimal (SHA)	7 177	552	3 027	14.5	2 128	100 MHz	4.63 s	10.33 ms	2.68 ms
Time /Area (SHA)	12 463	340	6 525	14.5	3 666	100 MHz	1.68 s	4.98 ms	1.01 ms
Performance (SHA)	30 454	307	15 191	14.5	8 675	95 MHz	0.77 s	4.20 ms	561.01 us
Minimal (SHAKE)	10 932	269	5 960	14.5	3 248	100 MHz	1.18 s	2.58 ms	575.24 us
Time /Area (SHAKE)	20 651	340	12 368	14.5	5 753	95 MHz	0.48 s	1.32 ms	243.86 us
Embd. Vrfy (SHA)	4 596	48	2 646	14.5	1 467	91 MHz	-	-	2.92 ms
Cryptoagile (SHA+KE)	13 373	304	7 026	14.5	3 893	100 MHz	4.63/ 1.18 s	10.33/ 2.58 ms	2.68/ 0.58 ms

hash core and not a general-purpose hash core as part of this work. The larger configurations which make use of the hash bus and WOTS+ chain parallelism outperform the hardware / software approach by several orders of magnitude. The performance version using the SHA2 hash algorithm is nearly 6 times faster in key generation, about twice as fast during sign and more than 10 times faster during verification while still maintaining a reasonable area utilization.

5 Conclusion

In this work, we presented the first configurable hardware implementation of the eXtended Merkle Signature Scheme. We demonstrated the feasibility of such XMSS hardware accelerators fitting the area requirements of even entry-level FPGAs. We conducted a thorough design evaluation which gives an intention for area and speed of the implementation far beyond the shown configurations. It is important to notice that each of the parameters can be adjusted individually, thus yielding a huge range for optimizations and improvements suited for many specific use-cases.

Our implementation outperforms the existing hardware/software co-design approach by multiple orders of magnitude, though even faster configurations of our implementation are possible by adding even more hash cores and WOTS+ chains. The embedded verification implementation instantiates a minimal configuration that can be used to verify XMSS signatures on embedded devices.

A crypto-agile version can be used to configure the underlying hash algorithm at runtime. This is especially interesting because, despite of implementation errors, according to the proof the single point of failure of the XMSS remains its hash function. In this context, we consider our crypto-agile design as currently best option to achieve a long-term secure hardware implementation of a digital signature scheme.

References

1. Banerjee, U., Ukyab, T.S., Chandrakasan, A.P.: Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(4), 17–61 (2019). https://doi.org/10.13154/tches.v2019.i4.17-61, https://doi.org/10.13154/tches.v2019.i4.17-61
2. BSI: Technical Guideline - Cryptographic Mechanisms: Recommendations and Key Lengths. Report BSI TR-02102-1, Bundesamt für Sicherheit in der Informationstechnik (Mar 2020), version: 2020-01
3. Buchmann, J., Dahmen, E., Ereth, S., Hülsing, A., Rückert, M.: On the Security of the Winternitz One-Time Signature Scheme. In: International Conference on Cryptology in Africa. pp. 363–378. Springer (2011)
4. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: International Workshop on Post-Quantum Cryptography. pp. 117–129. Springer (2011)
5. Campos, F., Kohlstadt, T., Reith, S., Stöttinger, M.: LMS vs XMSS: comparison of stateful hash-based signature schemes on ARM cortex-m4. In: Nitaj, A., Youssef, A.M. (eds.) Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12174, pp. 258–277. Springer (2020). https://doi.org/10.1007/978-3-030-51938-4_13, https://doi.org/10.1007/978-3-030-51938-4_13
6. Cooper, D.A., Apon, D.C., Dang, Q.H., Davidson, M.S., Dworkin, M.J., Miller, C.A.: Recommendation for Stateful Hash-Based Signature Schemes. NIST Special Publication **800**, 208 (2020)
7. Deutsch, D.: Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **400**(1818), 97–117 (1985)
8. Ding, J., Schmidt, D.: Rainbow, A New Multivariable Polynomial Signature Scheme. In: International Conference on Applied Cryptography and Network Security. pp. 164–175. Springer (2005)
9. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://doi.org/10.13154/tches.v2018.i1.238-268
10. Ferozpuri, A., Gaj, K.: High-Speed FPGA Implementation of the NIST Round 1 Rainbow Signature Scheme. In: 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig). pp. 1–8. IEEE (2018)
11. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-Fourier lattice-based

compact signatures over NTRU. Submission to the NIST's PQC standardization process (2018)
12. Hülsing, A., Butin, D., Gazdag, S., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. Internet Research Task Force (IRTF), RFC **8391**, 1–74 (2018)
13. Kampanakis, P., Fluhrer, S.R.: LMS vs XMSS: A comparison of the stateful hash-based signature proposed standards. IACR Cryptol. ePrint Arch. **2017**, 349 (2017), `http://eprint.iacr.org/2017/349`
14. Kannwischer, M.J., Genêt, A., Butin, D., Krämer, J., Buchmann, J.: Differential power analysis of XMSS and SPHINCS. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 168–188. Springer (2018)
15. Lamport, L.: Constructing digital signatures from a one-way function. Tech. rep., Technical Report CSL-98, SRI International (1979)
16. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali Hash-Based Signatures. Internet Research Task Force (IRTF), RFC **8544** (2019)
17. Merkle, R.C.: A Certified Digital Signature. In: Conference on the Theory and Application of Cryptology. pp. 218–238. Springer (1989)
18. Moody, D.: Round 2 of NIST PQC Competition. PQCrypto (2019)
19. Nguyen, D.T., Dang, V.B., Gaj, K.: A High-Level Synthesis Approach to the Software/Hardware Codesign of NTT-Based Post-Quantum Cryptography Algorithms. In: 2019 International Conference on Field-Programmable Technology (ICFPT). pp. 371–374 (2019). https://doi.org/10.1109/ICFPT47387.2019.00070
20. QuantumRISC: Quantumrisc — next generation cryptography for embedded systems (2020), `https://www.quantumrisc.org/`
21. Wang, W., Jungk, B., Wälde, J., Deng, S., Gupta, N., Szefer, J., Niederhagen, R.: XMSS and Embedded Systems. In: International Conference on Selected Areas in Cryptography. pp. 523–550. Springer (2019)
22. Zujko, K.: Improving Differential Power Analysis of XMSS. In: The Book of Articles National Scientific Conference "Science and Young Researchers" IV edition. p. 112 (2020)