ASIC Accelerator in 28 nm for the Post-Quantum Digital Signature Scheme XMSS

Prashanth Mohan*,*, Wen Wang*,†, Bernhard Jungk‡, Ruben Niederhagen§, Jakub Szefer† and Ken Mai*
* CMU †Yale Univeristy ‡Independent Researcher §Fraunhofer SIT

***Abstract*—This paper presents the first 28 nm ASIC implementation of an accelerator for the post-quantum digital signature scheme XMSS. In particular, this paper presents an architecture for a novel, pipelined XMSS Leaf accelerator for accelerating the most compute-intensive step in the XMSS algorithm. This paper then presents the ASIC designs for both an existing non-pipelined accelerator architecture and the novel, pipelined XMSS Leaf accelerator. In addition, the performance of the 28 nm ASIC is compared to the same designs on 28 nm Artix-7 FPGA. The novel pipelined XMSS Leaf accelerator is 25% faster compared to the non-pipelined version in the ASIC, and both accelerator architectures have a 10× lower power consumption than on the FPGA. The evaluation shows that the pipelining increases the frequency by 1.7× on the FPGA but only 1.2× on the ASIC, due to the critical path in the ASIC being in the memory. The non-pipelined XMSS Leaf accelerator is shown to have a significantly better area-delay and energy-delay metric on the ASIC, while the pipelined accelerator wins out in these metrics on the FPGA. Consequently, this work shows the different architectural decisions that need to be made between FPGA and ASIC designs, when selecting how to best implement post-quantum cryptographic accelerators in hardware.**

I. Introduction

Most common asymmetric cryptographic schemes such as RSA will become insecure once sufficiently large and fault-tolerant quantum computers are built. To address this security threat, alternative algorithms that are not known to be vulnerable to attacks using quantum computers — the post-quantum cryptographic (PQC) algorithms — have been proposed.

Starting from 2017, NIST launched a new standardization process [1] with the target of selecting the next generation of public-key cryptographic algorithms that are quantum-secure. As the standardization process evolved to the second round, the performance of different PQC algorithms is becoming an increasingly important metric in evaluation of the designs. Active research has been focused on the software implementations of PQC schemes, especially on high-end Intel CPUs. Apart from software implementations, some work has also explored the FPGA designs for some of the PQC schemes, e.g., [2]–[4]. However, today there is limited understanding on how to implement these algorithms on an ASIC. There are only few publications that explore ASIC designs of quantum-secure algorithms [5], [6].

To help expand understanding how to design ASIC accelerators for PQC algorithms, this work focuses on developing efficient ASIC designs for the post-quantum secure eXtended Merkle Signature Scheme (XMSS). XMSS is a stateful hash-based digital signature scheme [7] which has been standardized by the IETF [8] in 2018. Unlike modern digital signature schemes based on RSA or Elliptic Curves, XMSS builds its security upon secure hash algorithms. This gives XMSS well-understood security properties and makes it one of the most confidence-inspiring post-quantum signature schemes. Since XMSS is a stateful scheme, it supports a limited number of signature operations which is decided by the security parameters of the instantiation of the algorithm.

XMSS uses a binary tree structure with many one-time signatures on the leaf nodes [8]. The leaf nodes are effectively chains of hash computations, and the repeated hash computations are the key bottleneck in the key generation. Similarly, within the signature generation and signature verification steps, hash computations take a big portion of the execution time. Therefore, XMSS has a relatively high computation demand as thousands of hash computations for key generation, signing, and verification are needed.

This paper consequently presents an architecture for a novel, *pipelined* XMSS Leaf accelerator for accelerating the most compute-intensive leaf node generation operations in XMSS. We also explore an existing, *non-pipelined* XMSS Leaf accelerator [3] architecture, and present the comparison results on both FPGA and our ASIC.

A. Contributions

The contributions of this paper are as follows:

- We implement the hardware design of a four-stage, pipelined SHA-256 accelerator, and demonstrate that the pipelined architecture improves the achievable frequency of the SHA-256 core.
- We present the hardware design of a pipelined XMSS Leaf accelerator, which achieves a much better frequency compared to the existing, non-pipelined XMSS Leaf accelerator.
- We present the first 28nm ASIC implementation of the accelerators, for both the non-pipelined and the pipelined designs.
- By comparing the two accelerator designs on 28nm FPGAs and 28nm ASICs, we show how the area and performance metrics differ. Moreover, we show how to better design hardware architecture for post-quantum

* The two first authors contributed equally to this work.

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00112

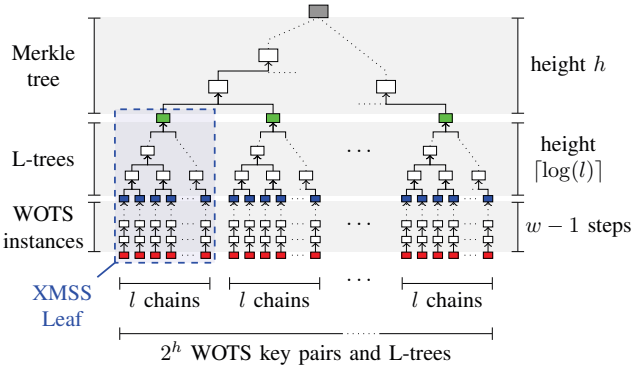

Fig. 1. The XMSS tree. Red nodes are the WOTS secret keys derived from the secret seed and blue nodes are the WOTS public keys. Green nodes are the L-tree roots (aka the Merkle tree leaf nodes) and the gray root node is the XMSS public key. The first XMSS leaf is marked with a dashed blue box. Figure based on [3].

cryptographic accelerators when targeting FPGAs and ASICs separately.

B. Paper Organization

The remainder of the paper is organized as follows. Section II discusses the preliminaries and the background on XMSS algorithm. Section III presents the hardware design, especially the pipelined architecture. Section IV gives the results, focusing on the ASIC chip evaluation and comparison to the FPGA implementation. The paper concludes in Section V.

II. Preliminaries

XMSS uses a binary Merkle tree [9] with Winternitz One-Time Signatures (WOTS) and unbalanced binary trees, called L-trees, as the leaf nodes [8]. As the name indicates, each one-time signature must only be used once. Therefore, along the private key, a state needs to be maintained (usually a counter), which ensures that each one-time signature is used only once. Each WOTS signature, which must be used only once, is computed using many hash chains.

In XMSS, *key generation* deals with generating cryptographic keys for the digital signatures. It is necessary to compute private keys that are later used in *signing* operations, and public keys that are later used in *verification* operations.

The private key of XMSS [8] is a secret seed that is used to generate the private keys of the WOTS signatures. The public key is the root node of the XMSS Merkle tree. The height h of the Merkle tree defines how many, specifically 2^h, signatures can be computed with a given private key. An overview over the XMSS tree structure is given in Figure 1.

A. Main Operations in the XMSS Scheme

For *key generation*, the entire tree needs to be computed: The secret seed is used to generate the secret keys of the one-time signatures, which are then used to compute the public keys of the one-time signatures. Each one-time signature public key consists of l distinct elements. These elements are then consolidated to a single element using an L-tree (i.e., an unbalanced binary tree with l leaf nodes) using pairwise hashing of nodes. These roots of the L-trees are the leaves of the Merkle tree. The root node of the Merkle tree is computed using pair-wise hashing of nodes as well.

For *signing*, first the next available one-time signature is used to sign the message and the corresponding L-tree is computed. Since the verifier only has the root node of the Merkle tree as public key available, the signer needs to provide a *verification path* through the Merkle tree to the verifier, i.e., each pairing node for the pairwise hashing in the tree. There are several algorithms for computing the verification path that differ in computing and storage cost. The simplest algorithm recomputes the entire tree and extracts nodes on the verification path for the signature during the computation as needed; there are alternative algorithms that store intermediate results in order to reduce the overall computational cost.

For *verification*, the receiver first verifies the one-time signature, computes the corresponding L-tree, and then recomputes the root node of the Merkle tree using the provided verification path. If the result is equal to the public key, the signature is accepted and otherwise rejected. The verification operation is the cheapest one among the three operations key generation, signing, and verification.

For the above mentioned *key generation*, *signing*, and *verification* operations, the computation of the WOTS signature scheme and the corresponding L-tree (bottom of Figure 1) takes the most time. The computation of a WOTS signature and the corresponding L-tree is in the following referred to as the *leaf computation*. For key generation, 2^h leaf computations need to be performed. The number of leaf computations for signing depends on the verification path-algorithm; in the best case, several computations are required, in the worst case 2^h leaf computations. For verification, only one leaf computation needs to be performed, which still takes up most of the verification time. Therefore, accelerating the leaf computations has a significant impact on the XMSS computation time.

B. XMSS Leaf Computation Overview

The leaves of the XMSS Merkle tree are WOTS instances [10], each with an L-tree. One WOTS instance contains l hash chains, each of length w (w is called the "Winternitz parameter"; l is defined by w). The start node of each chain is one WOTS secret key element; the end node is the corresponding WOTS public key element. The hash chains are iteratively computed by masking and hashing the previous node with a publicly keyed hash function. The masks are computed from a public seed together with individual addressing data using the publicly keyed hash function as well.

Once the l WOTS public keys have been computed by performing w steps in each hash chain, these l public keys elements are further compressed using an unbalanced binary L-tree. After the L-tree is fully constructed, its root node is returned as one leaf node of the XMSS Merkle tree.

For signing a message using WOTS, the message digest is split into l_1 base-w words. A checksum is computed over these words and split into l_2 base-w words as well. Each of the

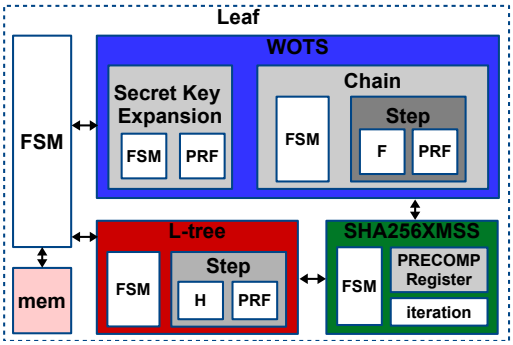

Fig. 2. High-level architecture of the XMSS Leaf accelerator (our *pipelined* version and the *non-pipelined* version [3] have the same architectural hierarchy). Colors of the highlighted submodules fit to the die microphotograph in Figure 6

$l = l_1 + l_2$ base-w words $w_1, \ldots, w_l$ now is interpreted as index into the l WOTS chains. The w_i-th nodes $n_{i,w_i}, i \in 1, \ldots, l$ of the l hash chains constitute the WOTS signature.

The verifier recomputes the l base-w words from the message digests and recomputes the WOTS public key elements: The hash chains are completed by computing $w - w_i$ further hash chain steps for each base-w word in the corresponding chain. Form the WOTS public key elements, the root of the L-tree and thus the leaf of the Merkle tree is recomputed and the verification is completed as described above.

C. XMSS Parameter Set

XMSS is parameterized for different hash functions and different hash-function parameter sets. In this work, we focus on the SHA-256 parameter set, which is required in the XMSS standard [8]. In this case, all secret and public key values have a length of 256 bits. The Winternitz-parameter $w = 16$ determines the length of the hash chains as well as the number $l = 67$ of hash chains within a WOTS instance. Standard values for tree height h are 10, 16, and 20. We are using $h = 10$ in this work without loss of generality.

There is also a multi-tree version XMSSMT of XMSS, which uses several layers of XMSS trees to obtain a larger number of possible signatures per private key at a moderate increase in computing cost but with significantly larger signature sizes. Our results can easily be mapped to XMSSMT.

III. Hardware Design

Figure 2 shows the diagram of our *pipelined* XMSS Leaf accelerator. The XMSS Leaf accelerator is composed of the WOTS module, L-tree module, and SHA256XMSS module as shown in the figure. The WOTS itself can be Key Expansion module and Chain module. Further the accelerator can be *pipelined* or *non-pipelined*, with details shown later in Figure 4.

A. Existing Work on Non-Pipelined XMSS Leaf Accelerator

Wang et al. [3] presented several XMSS hardware accelerators for FPGAs including a general-purpose SHA-256 accelerator and the following XMSS-specific hardware accelerators:

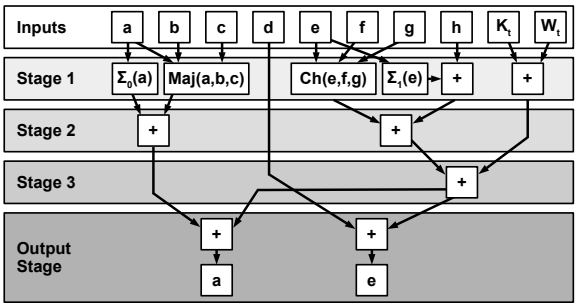

Fig. 3. The *pipelined* SHA-256 stages used to create the SHA256XMSS module presented in this work.

an XMSS-specific SHA-256 accelerator which contains fixed-length SHA-256 padding and the pre-computation feature, an optional internal storage for pre-computation, and an overall XMSS Leaf accelerator.

The key building block of the accelerators proposed in [3] is the implementation of an iterative version of SHA-256 where the computation of one round of SHA-256 takes one clock cycle. Due to the iterative design of the SHA-256 module, the XMSS Leaf accelerator proposed in [3] has a relatively long critical path. This accelerator is referred to as the *non-pipelined* XMSS Leaf accelerator in the following text.

B. Our Work on Pipelined XMSS Leaf Accelerator

In our design, we propose a *pipelined* version of the accelerator. The key part of our accelerator is the XMSS-specific pipelined SHA-256 accelerator module, called SHA256XMSS in the rest of this paper. Built upon this module, we then designed the *pipelined* XMSS Leaf accelerator that is used to compute one leaf node of the XMSS tree. Figure 2 shows the diagram of the *pipelined* XMSS Leaf accelerator, which shares the same architectural hierarchy as the *non-pipelined* XMSS Leaf accelerator proposed in [3]. Details of these hardware accelerators are provided as follows.

C. Our Pipelined SHA256XMSS Module

Each SHA-256 [11] round depends on all outputs of the previous round. Therefore, introducing p pipelining stages in the round function always increases the number of clock cycles by factor p. Simply adopting this idea leads to a decrease of the worst case latency and hence, an increased clock frequency, but at the same time low utilization of the implemented logic. Therefore, an efficient design can leverage multiple inputs in parallel to fill up the pipeline and use otherwise unused resources. With that approach, it is possible to increase the clock frequency, while keeping the average number of clock cycles per input the same. If the additional area consumption per processed input is less than the original design, the area-time product also improves.

For SHA256XMSS, the implementation of the round function is usually the critical path, because of the high amount of combinational logic that has to be placed between the two register stages in a single-cycle implementation. Therefore, we

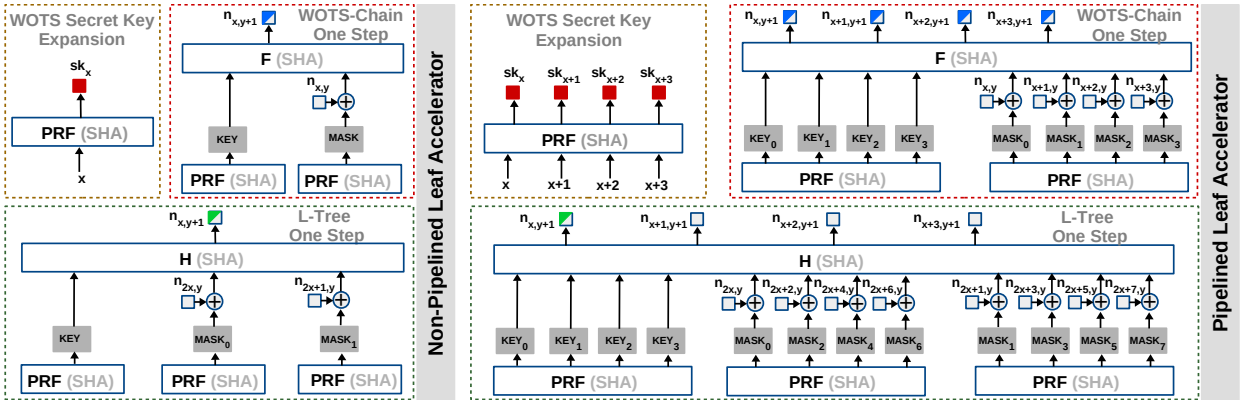

Fig. 4. Architecture of the XMSS Leaf accelerators: (left) *non-pipelined* and (right) *pipelined*. Red nodes are the WOTS secret keys and grey ones are nodes during WOTS and L-tree computations. Half-blue and half-green nodes represent the results from the last step. Grey boxes represent registers storing hash-function keys and masks. x shows node index and y shows the level in the chain/tree.

investigated the round function and designed our new pipelined architecture as depicted in Figure 3.

We used four register stages as this leads to a good balance between the number of registers used to forward previous intermediate results of the algorithm and the reduction of the worst case latency. Adding more register stages would likely further reduce the latency, but has the drawback of introducing too many registers that do not contribute to this latency reduction. Therefore, the time-area efficiency is likely to be negatively affected with more pipeline stages.

Beside the basic parallel pipelining, we also extended the "pre-computation" and "fixed input length" optimizations from [3] to support it for all four inputs independently, i.e., if one input uses the pre-computation feature, the next input can either be hashed without that feature enabled, or it might use a different pre-computed intermediate state. Furthermore, the scheduling of the pipeline is implemented in a flexible way, such that the computation of a input can be started at almost any time, independently of the other inputs processed in parallel.

D. Our WOTS Module

In our design, the secret keys (red nodes in Figure 1) for WOTS are generated using a secret seed and a pseudorandom number function (PRF). As the seeds of the PRF during expanding WOTS secret keys are secret, we denote this function as PRF_priv to distinguish its usage in different types of computations. The input of the PRF for generating different WOTS secret keys only depends on its chain index x (as shown in Figure 4) and the secret seed. Therefore multiple secret keys can be generated in parallel. In our design, using our pipelined SHA256XMSS module, four secret keys for WOTS are computed in a pipelined fashion in parallel by use of the PRF. Once the computation is finished, these four secret keys are forwarded directly to the WOTS chain module.

One WOTS instance is composed of l hash chains. These hash chains are used to compute the WOTS public keys. Once the secret keys (red nodes in Figure 1) of WOTS are available, they are forwarded to the Chain module and used as the starting nodes of the hash chains. By doing a chain of hash computations, the ending points of the hash chains will be stored in a single-port memory as the public keys of WOTS. As the computations of different hash chains only depend on its corresponding WOTS secret key, chain index and the public seed, multiple hash chains can be computed in parallel.

In our design, we can compute four public keys (blue nodes in Figure 1) of WOTS in parallel. As shown in Figure 4, one step in the hash chain takes in as input data the output from the previous step (or the WOTS secret key if it is the starting node of the hash chain), and the value of the ending node will be returned as the result of the Chain module. The computation of one step is composed of two PRF functions as well as a F function. The first PRF function is used to compute a 256-bit hash function key, which is stored in a 256-bit register. Similarly, the second PRF function computes a 256-bit value and stores it in a register. The mask value then gets XOR-ed with the input data of the step, and forwarded together with hash-function key as the input of the F function. The final result of the F function is returned as the result of the step computation. The above computations are repeated for $(w-1)$ times until a hash chain of length w is fully computed. Once the four hash chains are fully computed, the values of their ending nodes will be written to the memory in order.

To compute the public keys (blue nodes in Figure 1) for one WOTS instance composed of l chains, the computations of expanding four secret keys of WOTS followed by four parallel hash chain computations have to be repeated for $\lceil \frac{l}{4} \rceil$ times. Note that for computing the last $(l-4\times\lceil \frac{l}{4} \rceil)$ hash chains, the pipelines within the SHA256 module may not be fully filled. After all the hash chains are computed, the public keys of one WOTS instance are stored in a single-port memory which is of width 256 bit and of depth l.

E. Our L-tree Module

To further compress l public keys of one WOTS instance, an unbalanced L-tree is used in the XMSS scheme and its

root node composes one leaf node for the top level Merkle tree (green nodes in Figure 1) .

The construction of an L-tree starts from its leaf nodes or namely the nodes on the first level of the tree. Similar to the WOTS computations, the computations in the L-tree construction can be well parallelized as long as the nodes are lying on the same level, since there is no data dependency between the computations. As shown in Figure 4, to compute the nodes for the upper level of the L-tree, eight adjacent nodes are first read out from the memory. These eight adjacent nodes are further processed in pairs as follows: at first, a PRF function is computed and its results are stored as four 256-bit keys, which will be later used as the hash-function keys. After computing the keys, two PRF functions follow, each of them computing four masks. The values of the masks are later XOR-ed with the values of the input nodes, and the results are forwarded together with the hash-function keys to the last function H. Once the function H finishes, its results are returned as the four parent nodes of the eight input nodes. Note that since an L-tree is not a balanced binary tree, it may happen that there are odd number of nodes on one level of the tree. In this case, the last node on that level will simply be lifted to the upper level. This process is repeated for all the levels of the L-tree until the root node of the L-tree is reached.

Another design point worth noting is that, for the lower levels of the L-tree, the pipelines of the SHA256XMSS module can be mostly filled as there are enough nodes to be computed. However, as the computation reaches the top three levels, the pipelines simply cannot get filled, e.g., for computing the last level (root node) of the L-tree, only one stage out of the four pipelines of the SHA256XMSS module will be filled.

The "fixed input length" and "pre-computation" optimizations as proposed in [3] are enabled in our design. During the WOTS hash chain computations, each step involves the computations of two PRF functions. Similarly, for L-tree constructions, every time when two leaf nodes are merged together to compute their parent node, three PRF computations are required. The pre-computation feature can be easily enabled as follows: when the PRF functions are first called (with the public seed), the `store_intermediate` input signal of the SHA256 module are raised high to make sure that the pre-computed state is stored in a state register; and for all the following PRF computations, the `continue_intermediate` signal will be raised high to resume the state based on the value of the previously stored state. By doing this, we can save more than thirty percent of the hash computation time. More details about how to enable this feature can be found in [3].

IV. Evaluation Results

We implemented both the *non-pipelined* and *pipelined* versions of the XMSS Leaf accelerator in a 28nm bulk industrial CMOS process. The die microphotograph of the ASIC implementation is shown in Fig. 5. In addition to the ASIC, we implemented the same designs on a 28nm Artix-7 FPGA for evaluating the trade-offs between FPGA and ASIC implementations of *pipelined* and *non-pipelined* XMSS Leaf accelerators.

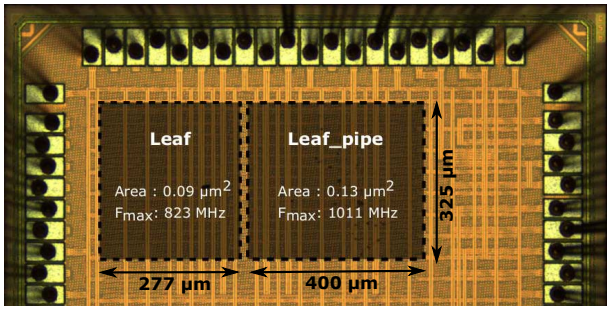

Fig. 5. Die microphotograph of the 28nm ASIC with XMSS Leaf accelerators

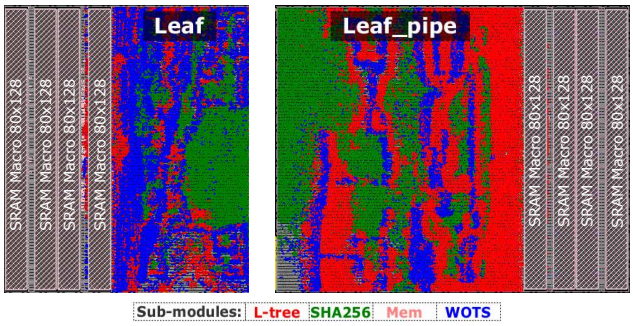

Fig. 6. Die microphotograph of the 28nm ASIC with XMSS Leaf accelerators

A. ASIC Results

Area: The instance count and the area breakdown of the sub-modules along with the total area are detailed in Table I. From the table we can see that the *pipelined* XMSS Leaf accelerator is 44% larger than the *non-pipelined* XMSS Leaf accelerator. The placement of the key sub-modules (SHA256XMSS, L-tree, WOTS, and memory) of the XMSS Leaf accelerators are highlighed in Fig 6. Almost all of the area increase in the *pipelined* version is due to pipelined SHA256XMSS and L-tree modules which grew by 2.6× and 2.1× respectively. This is because of the high logic complexity in these two designs when implemented in a pipelined fashion. While the sequential cell area increased by 239% due to the addition of pipeline flip-flops, the combinational cell area increased by just 43%. The area of the memory and the WOTS modules remained the same as pipelining of the SHA256XMSS module neither increases the memory requirements nor changes the WOTS computation.

Delay and Power: The operating frequency and power consumption of the two XMSS Leaf accelerators are shown in Table II. The *pipelined* and *non-pipelined* version of the XMSS Leaf accelerators take 296.1K cycles and 302.8K cycles to finish the computation respectively. While pipelining SHA256XMSS, which is iterative, can reduce the depth of the critical path and increase the operating frequency, it does not

TABLE I
AREA BREAKDOWN OF PIPELINED AND NON-PIPELINED XMSS LEAF ACCELERATORS. *Comb* IS THE COMBINATORIAL LOGIC AND *FF* ARE THE FLIP-FLOPS.

Design	Leaf [3]		Leaf_pipe	
	#Inst (K)	Area (mm^2)	#Inst (K)	Area (mm^2)
SHA256XMSS	13.8	0.013	31.8	0.034
L-tree	13.5	0.013	31.5	0.028
WOTS	14.9	0.014	15.8	0.015
Memory	0.12	0.031	0.03	0.031
Total (Comb)	41.1	0.030	70.3	0.043
Total (FF)	9.5	0.018	19.5	0.043
Layout		0.090		0.130

TABLE II
DELAY AND POWER OF PIPELINED AND NON-PIPELINED XMSS LEAF ACCELERATORS. THE PIPELINED XMSS LEAF ACCELERATOR IS 25% FASTER BUT CONSUMES 50% MORE POWER COMPARED TO THE NON-PIPELINED VERSION.

Design	Leaf [3]			Leaf_pipe		
	Freq. MHz	Power mW	Time ms	Freq. MHz	Power mW	Time ms
Fast (0.99 V)	823	67	0.36	1011	157	0.30
Typ (0.90 V)	659	45.9	0.45	806	103.5	0.37
Slow (0.81 V)	507	28	0.58	634	66.4	0.48

reduce the number of computation cycles. While pipelining results in the fastest version of the XMSS Leaf accelerator with the lowest latency (0.83×) and highest throughput (1.2×), it consumes 2.3× more power.

B. FPGA Results

The performance, resource utilization, and power consumption of the XMSS Leaf accelerators implemented on an 28nm Artix-7 FPGA are provided in Table III. Compared to the *non-pipelined* XMSS Leaf design [3], the *pipelined* XMSS Leaf accelerator achieves much higher frequency while maintaining similar cycle counts. Therefore, our *pipelined* XMSS Leaf accelerator is about 1.7× faster. However, it requires 1.65X more slices and consumes 2.5X more power.

C. Comparison between FPGA and ASIC

The *non-pipelined* XMSS Leaf accelerator is around an order of magnitude faster on the ASIC compared to an FPGA. However, the improvement in the *pipelined* XMSS Leaf accelerator implementation is only 6.3×. Moreover, while pipelining increases the frequency by 1.7× on the FPGA, the frequency only improved by 1.2× on the ASIC implementation. This is because pipelining moves the critical path to the memory stage in the ASIC design, which is not the case for the FPGA implementation. The ASIC also consumes 10× lower energy than the FPGA for both XMSS Leaf accelerators. Since pipelining results in a reduced frequency improvement for the ASIC compared to the FPGA, the *non-pipelined* XMSS Leaf accelerator has significantly better area-delay and energy-delay metric in an ASIC while there is no significant difference in these metrics in the FPGA implementation.

TABLE III
RESOURCE UTILIZATION, PERFORMANCE, AND POWER CONSUMPTION OF THE XMSS LEAF ACCELERATORS IMPLEMENTED ON AN 28NM ARTIX-7 FPGA

	Leaf [3]	Leaf-pipe
Utilization (Slices)	2730	4498
LUTs / FFs / BRAMs	6289 / 8597 / 16	10351 / 14788 / 16
Fmax (MHz)	90.9	161.3
Exec. time (ms)	3.3	1.9
Cycles	296144	302859
Power (mW) @ Fmax	96	240
Power (mW) @ 90 MHz	96	156

TABLE IV
COMPARISON OF *non-pipelined* AND *pipelined* XMSS LEAF ACCELERATOR IMPLEMENTATIONS ON ASIC AND FPGA.

	ASIC		FPGA	
	Leaf [3]	Leaf_pipe	Leaf [3]	Leaf_pipe
Delay @ Fmax (ms)	0.36	0.30	3.3	1.9
Power (mW)	67	157	96	240
Energy per Op (uJ)	24.1	47.1	316.8	456
Energy-delay	**8.7**	14.1	1045	**866**
Area (mm2 / Slices)	0.09	0.13	2730	4498
Area-delay	**0.032**	0.041	9009	**8546**

V. CONCLUSION

This paper presented the first 28 nm ASIC implementation of an XMSS Leaf accelerator, including the first architecture for a novel, *pipelined* XMSS Leaf accelerator for accelerating the most compute-intensive step in the XMSS algorithm. This paper also presented the ASIC designs for both an existing non-pipelined accelerator architecture and the novel, *pipelined* accelerator. In addition, the performance of the 28 nm ASIC was compared to the same designs on 28 nm Artix-7 FPGAs. We showed that the novel *pipelined* XMSS Leaf accelerator is 25% faster compared to the *non-pipelined* version in the ASIC, and both accelerator architectures have a 10× lower power consumption than on the FPGAs. The evaluation showed that the pipelining increases the frequency by 1.7× on the FPGA but only 1.2× on the ASIC, since the critical path in the ASIC was found to be in the memory. We also showed that the *non-pipelined* XMSS Leaf accelerator has a significantly better area-delay and energy-delay metric on the ASIC, while the *pipelined* XMSS Leaf accelerator wins out in these metrics on the FPGA. In summary, this work showed the different architectural decisions that need to be made between FPGA and ASIC designs, when selecting how to best implement such post-quantum cryptographic accelerators in hardware.

ACKNOWLEDGEMENTS

The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research under the project "QuantumRISC" (ID 16KIS1033K). This work was also funded in part by NSF grant 1716541.

REFERENCES

[1] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography Standardization," https://csrc.nist.gov/projects/post-quantum-cryptography, 2017, accessed: 2020-06-11.

[2] W. Wang, J. Szefer, and R. Niederhagen, "Fpga-based niederreiter cryptosystem using binary goppa codes," in *International Conference on Post-Quantum Cryptography*. Springer, 2018, pp. 77–98.

[3] W. Wang, B. Jungk, J. Wälde, S. Deng, N. Gupta, J. Szefer, and R. Niederhagen, "XMSS and embedded systems – XMSS hardware accelerators for RISC-V," in *International Conference on Selected Areas in Cryptography*. Springer, 2019, pp. 523–550.

[4] T. Oder and T. Güneysu, "Implementing the newhope-simple key exchange on low-cost fpgas," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2017, pp. 128–142.

[5] U. Banerjee, T. S. Ukyab, and N. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 17–61, 2019.

[6] U. Banerjee, A. Pathak, and A. P. Chandrakasan, "An energy-efficient configurable lattice cryptography processor for the quantum-secure internet of things," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 46–48.

[7] NIST, *FIPS PUB 186-4: Digital Signature Standard*. National Institute of Standards and Technology, 2013.

[8] A. Hülsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme," RFC 8391, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8391

[9] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology – CRYPTO 1989*, ser. LNCS, G. Brassard, Ed., vol. 435. Springer, 1990, pp. 218–238.

[10] A. Hülsing, "W-OTS+ – shorter signatures for hash-based signature schemes," in *Progress in Cryptology – AFRICACRYPT 2013*, ser. LNCS, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds., vol. 7918. Springer, 2013, pp. 173–188.

[11] NIST, *FIPS PUB 180-4: Secure Hash Standard*. National Institute of Standards and Technology, 2012.