Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium

Naina Gupta, Arpan Jati, Anupam Chattopadhyay, and Gautam Jha

***Abstract*—The looming threat of an adversary with Quantum computing capability led to a worldwide research effort towards identifying and standardizing novel post-quantum cryptographic primitives. Post-standardization, all existing security protocols will need to support efficient implementation of these primitives. In this work, we contribute to these efforts by reporting the smallest implementation of CRYSTALS-Dilithium, a finalist candidate for post-quantum digital signature.**

By invoking multiple optimizations to leverage parallelism, pre-computation and memory access sharing, we obtain an implementation that could be fit into one of the smallest Zynq FPGA. On Zynq Ultrascale+, our design achieves an improvement of about 36.7%/35.4%/42.3% in Area×Time (LUTs×s) trade-off for KeyGen/Sign/Verify respectively over state-of-the-art implementation. We also evaluate our design as a co-processor on three different hardware platforms and compare the results with software implementation, thus presenting a detailed evaluation of CRYSTALS-Dilithium targeted for embedded applications. Further, on ASIC using TSMC 65nm technology, our design requires 0.227mm^2 area and can operate at a frequency of 1.176 GHz. As a result, it only requires 53.7μs/96.9μs/57.7μs for KeyGen/Sign/Verify operation for the best-case scenario.

***Index Terms*—post-quantum, cryptography, PQC, CRYSTALS-Dilithium, FPGA, hardware, ASIC, hardware accelerator**

I. Introduction

The threat of an adversary with Quantum computing capability is getting increasingly realistic [1], [2], with rapid growth in the capacity of Quantum computers, as well as, optimized implementations of the current cryptographic primitives using Quantum circuit simulators [3], [4]. It is predicted that current public-key primitives could be broken within hours [5], thus necessitating the search for alternative cryptographic primitives in the era of Quantum computing. This is systematically undertaken by NIST through its Post-Quantum Cryptography (PQC) contest [6], which plans to roll out the winners of this contest as a standard. Naturally, there is a pressing need to study efficient hardware implementations of the PQC candidates, which not only plays an important role in the contest judgement process but also helps in the rapid adoption of the standard.

The current finalists of the NIST PQC contest consist of 3 candidates in digital signature scheme, one of which has recently been attacked [7]. There have been several previous works as well compromising the security of this scheme [8]–[10]. This leaves two other digital signature candidates among which Dilithium is one. Naturally, it is of high importance to study efficient implementations of Dilithium [11]. Digital signature being an integral part of numerous security protocols, the use-cases and the platform constraints range from high-speed servers to highly resource-constrained IoT platforms. As a result, there have been several efforts towards optimizing Dilithium for different security parameters and different platforms such as FPGAs and ASICs targeting pure-hardware based implementation [12]–[16], HLS based implementations [17]–[19] or as a software-hardware co-design [20]. Further, few works focused on integration in TLS protocol [21], as a GPU accelerator [22] and for developing a quantum secure blockchain [23].

From our studies on the existing Dilithium implementations, we found that there is a significant room for improvement in terms of area-efficiency, which sets the motivation for this work. The main contributions of this work are:

1) In this work, we designed a lightweight hardware accelerator for CRYSTALS-Dilithium. We used multiple optimization strategies such as resource and control logic sharing, fusion of modules, pre-computed LUTs etc.
2) By achieving a reduction of about 24% in LUTs and FFs than state-of-the-art implementation, this work presents the smallest hardware accelerator for Dilithium. As a result, it can now be fit into one of the smallest Zynq FPGA.
3) In our design, we have leveraged both pipelining as well as parallelism to achieve a good balance of performance and area. Thus, on Zynq Ultrascale+, our design achieves efficiency of more than 35% for Area×Time compared to existing implementation.
4) To present a fair comparison with the existing implementations, we used two metrics - Area×Time and number of operations that can be performed per second per LUT on a particular platform. On Zynq Ultrascale+, our design outperforms the state-of-the-art. Whereas, on Artix-7, our design has better performance for signing operation.
5) Using TSMC 65nm library, we also implemented the design on ASIC platform and report numbers for major modules as well as the overall design. On ASIC, our design can run at 1.176 GHz with 0.227 mm^2 area

N. Gupta and A. Chattopadhyay, School of Computer Science and Engineering, Nanyang Technological University, Singapore.

A. Jati, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore.

G. Jha, Department of Electronics and Electrical Communication Engineering, IIT Kharagpur, India.

achieving a reduction of 1.4× in area and improvement of more than 1.7× in execution times for KeyGen, Sign and Verify.

6) Further, we performed hardware evaluation of the implemented design as an accelerator on three different hardware platforms and achieved a speedup of 6-15× for modern high performance CPUs and about 105-261× for Microblaze compared to software implementations for different operations.

The paper is organized as follows. Section II starts with the notations and presents a brief background about Dilithium. Section III shows the overall system architecture alongwith the design decisions. It is then followed by the different experimental results and performance comparison with the state-of-the-art implementations in Section IV. The performance evaluation as a hardware accelerator is presented in Section V and Section VI finally concludes the work.

II. Preliminaries

A. Notations

Throughout this work, we use the following notation. Lower-case letters are used to represent vectors (e.g. **e**) and the polynomials in NTT domain are represented using a hat over the symbol (**ê**). Matrices are represented using bold upper-case letters (e.g. **A**). $\mathscr{R} = \mathbb{Z}[X]/(X^n+1)$ denotes the ring of integer polynomials modulo (X^n+1), where n is a power of 2. $\mathscr{R}_q$ is the polynomial ring with coefficients modulo q. For dilithium, n = 256 and modulus q = 2^{23} - 2^{13} + 1.

B. Protocol Description

The digital signature scheme CRYSTALS-Dilithium is based on the hardness of the Module Learning with Errors (MLWE) and the Short Integer Solution (SIS) problems. In this section, we briefly discuss about the hardness problems and the protocol. Interested readers are referred to [24] for more details. Let us consider a matrix **A** of dimension $k \times l$ and vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ of dimensions l and k sampled uniformly. Then, the MLWE problem can be defined as: Given (**A**, $\mathbf{As}_1$ + $\mathbf{s}_2$) and (**A**, **b**) where $\mathbf{b} \approx \mathbf{As}_1 + \mathbf{s}_2$ and is a uniformly sampled vector, the goal is to distinguish (**A**, $\mathbf{As}_1 + \mathbf{s}_2$) from (**A**, **b**). One can note that they are approximately equal, but the unknown error vector ($\mathbf{s}_2$) makes it quite difficult to distinguish. The SIS problem can be defined as: Given **A**, the goal is to find a vector **x** such that $\mathbf{Ax} = 0$ and the norm of **x** is smaller than an integer value called norm bound β. The dilithium signature scheme consists of three algorithms key generation (KeyGen), signature generation (Sign) and verification (Verify) shown in Fig. 1.

1) KeyGen(): Key generation algorithm generates a keypair consisting of a public verification key (*pk*) and a private signing key (*sk*). It utilizes two random seeds public seed (ρ) and noise seed (ρ') and expands them using a variant of SHAKE-128 to generate the matrix **A** and two vectors $\mathbf{s}_1$ and $\mathbf{s}_2$. It then computes $\mathbf{t} = \mathbf{As}_1 + \mathbf{s}_2$ to generate the final keypair.
2) Sign(sk, M): The purpose of this algorithm is to take the message M and the signing key sk and generate the signature sig. For this, first a masking vector **y** is generated using SHAKE-256 and then it is used to compute **w** = **Ay**. The high-order bits of **w** (denoted as $\mathbf{w}_1$) are then hashed with the message M to generate the challenge c. This challenge is used to generate the signature as **z** = **c**$\mathbf{s}_1$ + **y**. Apart from generating **z**, the algorithm also generates some hints h for the verifier. If the signature passes all the correctness and security checks, then sig consisting of (**c**, **z**, **h**) is sent as the final signature to the verifier. Otherwise, the signature is re-computed again as shown in the rejection loop.
3) Verify(pk, sig, M): The verifier computes $\mathbf{Az} - \mathbf{ct}_1$ and set the high-order bits in $\mathbf{w}_1$. It is then hashed with the message M to generate the challenge **c**. This challenge is compared with the one received in the signature. If the challenges match and also the norm of **z** is valid, then the signature is accepted and the algorithm returns $valid\_sig$ as true, otherwise signature is rejected.

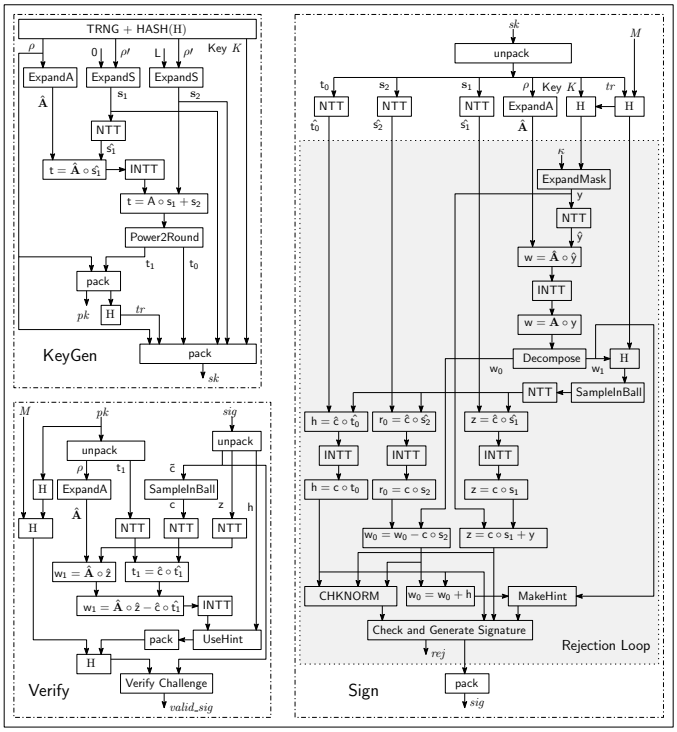

Fig. 1. Dilithium Protocol

III. Architecture and Design Decisions

Here, we present the overall system architecture and discuss various design decisions which led to the overall optimized modules.

A. System Architecture

The architecture for three algorithms KeyGen, Sign and Verify is combined together and the resources are extensively shared to keep the memory footprint as small as possible. A global input enable signal is used to start the required operation. The modules having similar control logic are also combined together to further reduce the resource utilization.

Such modules are shown together in the overall design architecture shown in Fig. 2.

The hardware accelerator is designed using a dedicated FSM based control unit. All the modules have separate enable and done signals and are connected to the memory controller. The control logic and sequencer unit is responsible for enabling different modules depending on the required functionality. The memory controller consists of a dual-port switch matrix and is responsible to connect different modules with the RAMs depending on the input and the output for the operation.

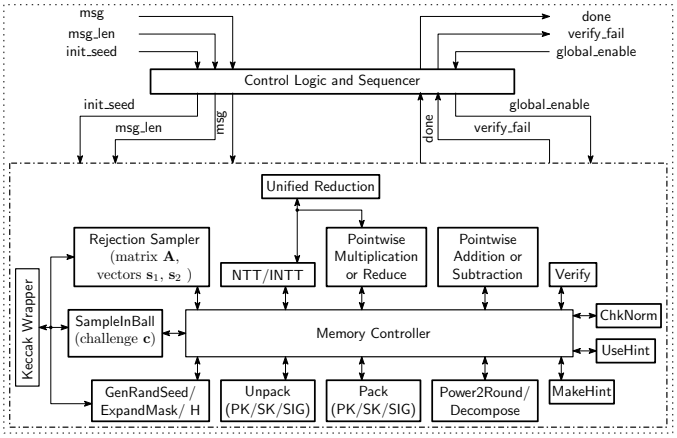

Fig. 2. High-level System Architecture of Dilithium

B. Memory Requirements

The upper bound on the memory storage is decided based on the signature generation part. In the worst-case scenario for Level V parameters, one needs to store about 118 polynomials. This huge requirement is mainly because of matrix $\mathbf{A}$ of size $k \times l \times 256$. In case of KeyGen and Verify, one can generate partial matrix of size $l \times 256$, perform the computation and store the final result. One can execute these steps k times separately to save resources and generate the final output of pointwise operation. But, in case of Sign operation, due to multiple rejections, there is a trade-off between pre-computing and storing the complete matrix $\mathbf{A}$ or re-computing partial matrix coefficients. The former approach requires more RAMs and less clock cycles whereas the latter results in less storage but more clock cycles. In our design we chose to compute it fully (56 polynomials) only once at the expense of RAM requirements. To balance this requirement, we analyzed the Sign algorithm to determine which variables are never accessed simultaneously (for instance, $\mathbf{z}$ and $\mathbf{y}$) and utilized same RAMs for both the variables. Also, the internal bus-width for each polynomial is variable and optimized based on the coefficient size. For instance, the zetas are only of size 23-bit, hence the BRAM data width is set to 23-bit instead of 32-bit. Similarly, most of the coefficients for Dilithium can be fit into 26-bit resulting in a reduction of about 16.8% BRAM requirements compared to an implementation with a fixed bus width of size 32-bit.

C. Parallel Processing

In order to reduce clock cycles, the data independent operations can be executed in parallel. One can perform fine-grained parallel execution of operations such as in [13], but, this leads to significant area overheads, mainly because of increase in multiplexing, additional module instances etc. We implemented parallel execution of modules wherever possible while ensuring low area.

D. SHAKE

Dilithium uses SHAKE-128 and SHAKE-256 extendable-output functions (XOFs) of the SHA-3 [25] family which is based around the Keccak permutation [26] for different functionalities with minor variations. For instance, SHAKE-256 and its variations are used to generate random seeds from an initial seed, for the hashing (H) and also to generate the challenge ($\mathbf{c}$). Whereas, to generate Matrix $\mathbf{A}$, vectors $\mathbf{s}_1$, $\mathbf{s}_2$ and masking vector $\mathbf{y}_1$, variants of SHAKE-128 are used.

As the XOFs are one of the most expensive and time consuming operations, it is important to design them carefully. Consequently, there can be multiple possible design choices. One can have multiple instances of the Keccak as in [15], cascade two rounds of Keccak [13] to increase performance or have a single shared instance to achieve all the functionalities to save resources at the expense of some performance. In our design, we chose the latter approach and created a unified wrapper around Keccak to support all the required variations. For Keccak, we used the implementation provided by the designers [27]. As Dilithium is a digital signature scheme,

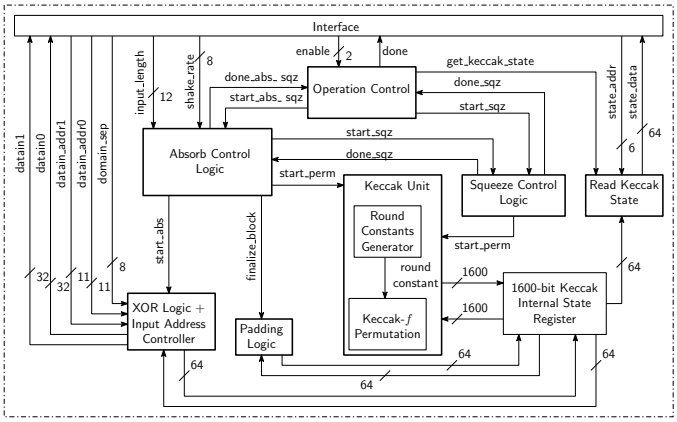

Fig. 3. Unified SHAKE Wrapper

the architecture should support variable input message length. Also, the requested output length from the XOFs is different depending on the operation being performed. Such variations with different input/output length, different padding logic, nonce support, etc. makes it quite challenging to have one module which is efficient in hardware as well as serve all the required functionalities. As a result, the overall logic for this module is quite complex. In order to provide support for variable input and output message length, the design has three configurable modes of operation - perform absorb followed by one squeeze (mode = 1), perform only squeeze (mode = 2) and read Keccak state (mode = 3). The modes are

extremely useful when the number of executions for squeeze is unknown (for instance, in case of rejection sampling). Also, mode = 3 allows the Keccak state to be accessed from outer modules, as a result we do not need extra resources to store additional copies of the 1600-bit register elsewhere compared to the work in [13]. The internal state remains valid after each squeeze operation unless reset from outside. Fig. 3 shows the architecture for the implemented SHAKE wrapper module. The interface is used to set the corresponding mode using enable signal, provide necessary information such as number of bytes to absorb (input_length), shake rate (shake_rate), etc. and to communicate with the different outer modules.

The control logic for all the modes are responsible to start the corresponding operation. For example, for a SHAKE-128 operation, the outer module first sets the enable signal to mode = 1 alongwith the input_length and shake_rate. The operation control decodes the mode and transfers the control to absorb control logic. This starts the absorb phase by reading in the domain separator and input. The XORing of the input with the Keccak state and the generation of corresponding input addresses are handled by XOR Logic + Input Address Controller. In our design, we are absorbing data in chunks of 64-bits per clock cycle. The absorb control logic is responsible for monitoring the absorbed input length and starts the permutation if required (when the input message length is greater than the shake rate). Further, when the input is completely absorbed, it enables the padding logic to finalize the absorbed block. The absorb control logic then starts a squeeze operation by setting start_sqz signal. The squeeze block starts the Keccak permutation and waits for it to finish execution using a dedicated counter (25 clock cycles as Keccak has 25 rounds). It then sends the done_sqz signal to the absorb control logic which then triggers the done_abs_sqz signal to operation control. Once the outer module receives the done signal, it starts reading the Keccak state by setting mode = 3, and providing the corresponding state address. In a single clock cycle, 64-bit data can be fetched from the wrapper. The outer module processes this data first and if more data is required, it simply increments the address, otherwise it sends a read done signal to reset the internal Keccak state. If the required output length is more than the shake rate, then after reading enough data (= shake rate), the outer module sets mode = 2 to start another squeeze operation and then read the data using mode = 3. Such an approach allows us to call squeeze back and forth only when required and when we have exhausted all the available Keccak output. Thus, saving us clock cycles as well as area as we do not need to pre-compute and store extra output bytes for example, in case of rejection sampling.

E. Sampling Modules

Dilithium requires four different types of sampling logic to generate matrix $\mathbf{A}$, the vectors $\mathbf{s}_1$, $\mathbf{s}_2$, challenge $\mathbf{c}$ and the masking vector $\mathbf{y}_1$. Even though all of them have different sampling logic, but the coefficients are sampled based on the output of a variant of XOF function. Hence, the sampling modules in our design only contain the sampling and control logic to start and fetch coefficients from the Keccak wrapper as discussed in III-D.

```
t0 = buf[pos] & 0x0F;
if (t0 < 15) {
      t0 = t0 - (205 * t0 >> 10) * 5;
      a[ctr++] = 2 - t0;
}
```

Listing 1. Rejection Eta

Listing 1 shows a small code snippet from the software reference implementation for rejection sampling. It is used to generate vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ from the output of SHAKE-128. One can note that the value for t0 can only be between 0 – 14. Thus, the whole computation (requiring multiple multiplication and subtraction operations) can be completely avoided and possible output values for polynomial coefficient can be pre-computed and stored in a LUT. This resulted in saving a lot of resources. The same logic is used for another value t1. Similar optimizations are used for other sampling modules wherever possible alongwith pipelining for good performance.

F. Combined Power2Round and Decompose

Fig. 4 shows the combined module for Power2Round and Decompose modules. As can be seen from the figure, the controller unit is shared between both the modules, thus saving about 180 LUTs and 120 FFs. The controller unit is used to enable the operation depending on the mode and generate address and write enable signals after receiving done from the respective module. For simplicity we have not shown the individual enable and done signals in the diagram. The

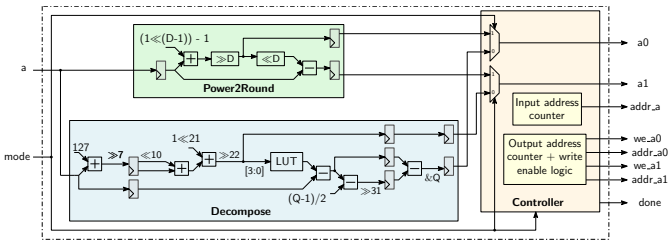

Fig. 4. Power2Round and Decompose module

software reference implementation of decompose module for Dilithium-V requires two multiplication operations to realize high-order and low-order bits of the polynomial coefficient. One is (a1*1025) and other one is (a1*2*GAMMA2) where GAMMA2 = 261888, resulting in an expensive implementation for Decompose. In order to prevent DSP usage for the two cases, we used two different strategies. We realized the first multiplication using an addition operation as (a1*1025) is equivalent to (a1*1024 + a1) and (a1*1024) is very efficient in hardware and can be realized by shift logic. For the second multiplication, we used a small 4-bit lookup-table (LUT) as the input a1 can only have 16 possible values. We pre-computed the output of (a1*2*GAMMA2) for all 16 possible values and used the LUT whenever required. The rest of the operations are mostly either shift operations or addition/subtraction with a constant value. Because of the optimizations employed, the

resource utilization of decompose is thus significantly reduced to only about 120 LUTs and 203 FFs. Power2Round is a very simple module requiring only one addition and subtraction operation. We have performed such control logic unification with multiple modules and benefited in terms of area.

G. NTT/INTT

The NTT/INTT is one of the most computationally expensive operations in Dilithium. Fig. 5 shows the basic architecture utilized in this implementation. As the NTT operations in Dilithium allows implementations with two simultaneous butterfly operations we have utilized two 64×256 dual-port memories attached with a *dual butterfly* unit. This allows for two butterfly-operations to be performed per clock cycle. The *dual butterfly* unit internally utilizes two multiplication-reduction units as described in section III-H.

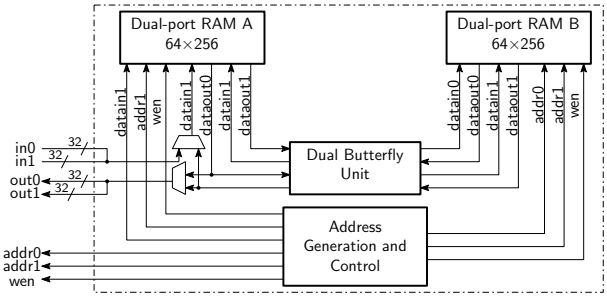

Fig. 5. NTT/INTT Architecture

The NTT operation is performed in three stages. First, the data from external memory is copied over to RAM-A. Second, the data in RAM-A is processed (128 butterfly operations) and written over to RAM-B and reverse, until all the layers have been processed. Finally the result is written back to the external RAM. The entire operation requires multiple addresses, write enables and zetas to be generated and provided to various modules. This is performed by the address generation and control state machine. At the end of INTT, the required scaling by $1/n$ is performed by the butterfly unit as well. So, no additional resources or clock cycles are utilized.

H. Modular Reductions

The work in [28], [29] proposed an efficient method for modular reduction in hardware for NewHope-NIST and CRYSTALS-Kyber. In this work, we followed a similar approach for efficient modular reduction in Dilithium. In order to compute a mod q, we used Dilithium modulus property $2^{23} = 2^{13} - 1 \pmod{8380417}$ recursively. The corresponding equations are as shown below:

$$\begin{aligned}
a &= 2^{23}a[45:23] + a[22:0] \\
&= 2^{13}a[45:23] - a[45:23] + a[22:0] \\
&= 2^{23}a[45:33] + 2^{13}a[32:23] - a[45:23] + a[22:0] \\
&= 2^{13}a[45:33] - a[45:33] + 2^{13}a[32:23] \\
&\quad - a[45:23] + a[22:0] \\
&= 2^{23}a[45:43] + 2^{13}a[42:33] + 2^{13}a[32:23] \\
&\quad - (a[45:33] + a[45:23]) + a[22:0] \\
&= 2^{13}(a[45:43] + a[42:33] + a[32:23]) \\
&\quad - (a[45:43] + a[45:33] + a[45:23]) + a[22:0] \\
&= 2^{13}c - e + a[22:0] (\bmod q)
\end{aligned}$$

where $c = (a[45:43] + a[42:33] + a[32:23])$ and $e = a[45:43] + a[45:33] + a[45:23]$. Using the same modulus property again, c can be further reduced as:

$$\begin{aligned}
2^{13}c &= 2^{23}c[11:10] + 2^{13}c[9:0] \\
&= 2^{13}(c[11:10] + c[9:0]) - c[11:10] \\
&= 2^{13}f - c[11:10] (\bmod q)
\end{aligned}$$

Further reduction is possible for f as shown below:

$$\begin{aligned}
2^{13}c &= 2^{23}f[10:10] + 2^{13}f[9:0] - c[11:10] \\
&= 2^{13}(f[10] + f[9:0]) - (f[10] + c[11:10])
\end{aligned}$$

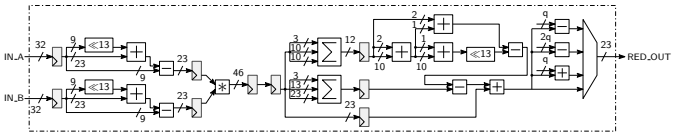

Fig. 6. Efficient Modular Reduction Module

Fig. 6 shows the implemented reduction module. The extra multiplication operation required before modular reduction operation is also integrated in this module. In order to reduce the input to 23-bit before multiplication, we first performed an initial reduction using the same recursive property. This helped in the reduction of 2 DSP resources per such module. The back-to-back flip flops correspond to the pipeline delays required for optimal DSP implementation. Two such modules are instantiated in hardware to speed up operations.

IV. Results and Performance Comparison

In this section, we present resource utilization and performance results for both FPGA and ASIC implementations. We also compare our results with the state-of-the-art implementations for Dilithium-V. The implemented design is realized in Verilog and the results are presented after synthesis and place and route using Vivado 2020.1 targeting two platforms Xilinx Artix-7 (XC7A200T-2) and Zynq UltraScale+ (XCZU9EG-2). The correctness of the implementation is verified using the KAT provided in the NIST package. From here on, Artix-7 is referred as A7 and Zynq UltraScale+ is referred as ZUS+. For ASIC, we synthesized the design for TSMC 65nm process technology. We used Synopsys Design Compiler version R-2020.09-SP5 for synthesis and Cadence Innovus

version V19.10-P002 for place-and-route of the design. We used CCS-based standard cell library for accurate results. Synopsys DesignWare library components were used wherever applicable.

A. Resource Utilization

Table I shows resource utilization for the major components required in Dilithium. Through extensive resource sharing between the modules, we obtained a significant reduction in overall resource utilization. One can note that the resource utilization for Sampling is quite low compared to the other existing implementations. This is mainly because the different Sampling modules contain only the minimal arithmetic and control logic with the XOFs shared between them. The slice registers usage is somewhat higher in our case because of the deep pipelines needed for good performance on both FPGA and ASIC. The various modules were individually optimized for hardware using multiple strategies wherever applicable.

TABLE I
RESOURCE UTILIZATION FOR FPGA AND ASIC. THE ASIC RESOURCES ARE REPORTED IN GATE EQUIVALENTS (GES).

Sub-module	FPGA				ASIC	Reference
	LUT	F/F	DSP	BRAM	GEs	
MakeHint	2389	740	0	0	-	[15]
	67	85	-	-	1423.7	**This Work**
UseHint	6453	2808	0	0	-	[15]
	186	279	0	0	4740	**This Work**
Encode +	1626	461	0	0	-	[15]
Pack	650	603	0	0	8884	**This Work**
Decode +	2189	239	0	0	-	[15]
Unpack	694	568	0	0	9458.3	**This Work**
Decompose	1437	680	0	0	-	[15]
	120	203	0	0	3028.7	**This Work**
NTT +	4509×2	3146×2	16	0	-	[15]
PolyArith	5676	1218	41	1	-	[16]
	2759	2037	4	7	40182	**This Work**
SampleA +	3548	1015	0	0	-	[15]
SampleS	1479	189	-	-	-	[16]
	360	355	0	0	4710.3	**This Work**
SampleY	2220	630	0	0	-	[15]
	469	48	-	-	-	[16]
	99	199	0	0	2353	**This Work**
SampleC	1856	868	0	0	-	[15]
	384	662	-	-	-	[16]
	289	244	0	0	3782.7	**This Work**
Keccak	5483×3	4451×3	0	0	-	[15]
	3708	1623	-	-	-	[16]
	4202	1800	0	0	42155.3	**This Work**

B. FPGA Implementation

Table II presents detailed results for our implementation as well as existing implementations for Dilithium-V. In our case, the combined architecture (for KeyGen, Sign and Verify) requires about 13.9k LUTs, 6.8k Slice registers, 35 BRAMs and 4 DSPs. Due to a deeply pipelined architecture, our design can run at a maximum frequency of 163 MHz on Artix-7 and 391 MHz on Zynq Ultrascale+. As a result, we require 387μs, 699μs and 416μs for KeyGen, Sign and Verify operations on Artix-7. For Zynq Ultrascale+, the respective operations can be finished in 161μs, 291μs and 173μs. We also report number of operations that can be performed per second (OP/s) for all the implementations.

Currently, to the best of our knowledge, there are four known hardware implementations which report results for Dilithium Security Level V. Hence, we compare our results with only these implementations.

*1) **Comparison with [15].**:* The work by Beckwith et. al. present results for multiple platforms as well as for all the security levels. For a fair comparison, we compare our results with the similar FPGA plaform. The authors in [15] targeted a high-performance implementation. As a result, they can perform Keygen, Sign and Verify in 121μs, 210μs and 126μs achieving a reduction of about 3.2×, 3.33× and 3.30× compared to our implementation. Even though the performance achieved is better, their design requires 3.81×, 4.14× and 4× more LUTs, FFs and DSPs than our design. This is because their architecture utilizes multiple cores to perform expensive operations such as NTT, Keccak whereas we are using only one core for almost all the modules. Thus, the high latency in our design is compensated by low area utilization as well as higher achievable frequency. As a result, their design has a higher Area×Time trade-off metric of 18.8%, 14.3% and 15.3% for KeyGen, Sign and Verify operations respectively.

*2) **Comparison with [16].**:* The authors in [16] target to reduce the area in terms of LUTs and FFs by utilizing more DSP resources in their architecture. Their design requires 3.2×, 2.02× and 11.25× more LUTs, FFs and DSP resources and about 1.13× less BRAMs compared to our design. Even though, our implementation has slightly more latency for the three operations, the overall improvement in terms of efficiency (Area×Time) is quite high. Compared to ours, their design has a lower Area×Time trade-off efficiency by 200%, 129% and 188% for KeyGen, Sign and Verify operations respectively.

*3) **Comparison with [13].**:* Zhao et. al. proposed a compact and high-performance hardware design. They did several optimizations such as segmented pipeline processing to achieve a high-performance architecture. As a result, they can perform KeyGen/Sign/Verify in 90μs/505μs/93μs which is 4.3×/1.38×/4.47× lower than our design execution time. One can note that the improvement in execution time for Sign is not that significant. We believe this is because the Sign operation does not offer much scope for parallel execution of operations. Further, even though their design offers high-performance, it comes at an expense of consuming large amounts of resources. Compared to our design, their implementation requires 114.7% and 51% more LUTs and FFs. As a result, the Area×Time trade-off is better in our case for Sign operation whereas it is better in [13] for KeyGen and Verify.

*4) **Comparison with [14].**:* The design by Aikata et. al. presents a unified architecture for Dilithium and Saber. The authors in [14] target a low area implementation. Hence, the

TABLE II
PERFORMANCE AND COMPARISON FOR DILITHIUM-V FOR BEST-CASE SCENARIO (SIGNATURE IS VALID AFTER THE FIRST ITERATION) ON FPGA AND ASIC. THE THE AREA FOR ASIC IS EXCLUDING ON-CHIP MEMORY AND IS REPORTED IN mm^2 AND GATE EQUIVALENTS (GES)

Reference	Family	Freq.	Area				KeyGen			Sign			Verify		
		(MHz)	LUT	FF	RAM	DSP	Cycles ($\times 10^3$)	Time (μS)	OP/s	Cycles ($\times 10^3$)	Time (μS)	OP/s	Cycles ($\times 10^3$)	Time (μS)	OP/s
FPGA Results															
This Work	Artix-7	163	13,975	6,845	35	4	63.2	387	2580	113.9	699	1430	67.9	416	2401
Beckwith et. al. [15]	Artix-7	116	53,187	28,318	29	16	14.0	121	8263	24.4	210	4762	14.6	126	7922
Land et. al. [16]	Artix-7	140	44,653	13,814	31	45	51.0	364	2746	70.4	503	1989	52.7	377	2656
Zhao et. al. [13]	Artix-7	96.9	29,998	10,336	11	10	8.8	90	11055	49.0	505	1977	9.0	93	10720
Beckwith et. al. [15]	Kintex-7	173	54,468	28,639	29	16	14.0	81	12324	24.4	141	7102	14.6	85	11815
This Work	Zynq Ultrascale+	391	13,975	6,845	35	4	63.2	161	6189	113.9	291	3431	67.9	173	5759
Aikata et. al. [14]	Zynq Ultrascale+	200	18,406	9,323	24	4	38.8	194	5149	68.5	342	2920	45.8	229	4368
Beckwith et. al. [15]	Virtex Ultrascale+	256	53,907	28,435	29	16	14.0	55	18238	24.4	95	10509	14.7	57	17483
ASIC Results															
This Work	TSMC 65nm	1176	0.227 mm^2		157 kGE		63.2	53.7	18614	113.9	96.9	10320	67.9	57.7	17322
Aikata et. al. [14]	UMC 65nm	400	0.317 mm^2		220 kGE		38.8	97.1	10298	68.5	171.3	5839	45.8	114.5	8735

Keccak core and the polynomial multiplier are shared between the two algorithms. But, there are many specialized modules required just for Dilithium, similarly for Saber. As a result, their implementation requires 31.7% and 36.2% more LUTs and FFs compared to our design but the BRAM utilization is comparatively low in their implementation. One interesting thing to note is our design can run at about twice the speed of their design. As a result, we are able to achieve an improvement of 17%, 14.9% and 24.5% in the execution time of KeyGen, Sign and Verify operations. Further, the Area×Time trade-off efficiency is lower by 57.9%, 54.8% and 73.3% compared to ours for the respective operations.

Efficiency Metrics. In order to collate all the results, we compare our work with the existing implementations using two metrics - Area×Time (LUTs×s) trade-off (shown in Fig. 7) and Number of operations (KeyGen/Sign/Verify) that can be performed per second per LUT (shown in Fig. 8). Both metrics are used to quantify the efficiency of an implementation. In the former case, a lower value denotes a better design, whereas for the latter, a higher value is considered to be good.

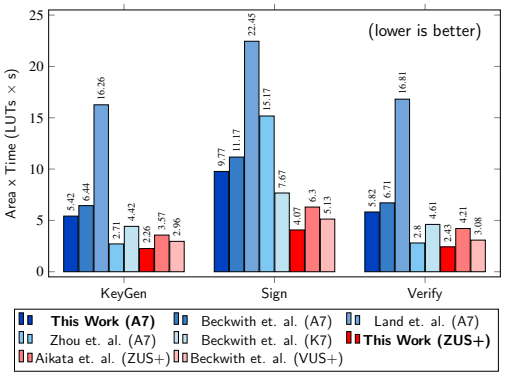

Fig. 7. **Area×Time (LUTs×s) trade-off:** Our design has better Area×Time trade-off in case of Ultrascale+ compared to the existing implementations. For A7 the implementation by Zhou et. al. [13] has better Area×Time trade-off for KeyGen and Verify, whereas our design is better for Sign operation. One interesting thing to note is that our result on Zynq Ultrascale+ is best across all platforms.

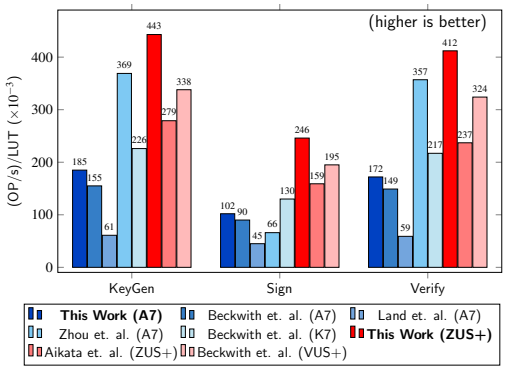

Fig. 8. **Operations/second/LUT:** In case of Zynq Ultrascale+, we are able to achieve an improvement of about 58.78%, 54.72% and 73.84% for KeyGen, Sign and Verify operations over the best known state-of-the-art implementation [14]. Whereas, when comparing our design on Artix-7, the improvement is about 54.5% for Sign operation. The improvement in ZUS+ is quite high for all the operations because our design is better in terms of both area as well as performance than the existing implementation.

C. Post-layout ASIC Implementation

Table II shows the results for the ASIC implementation. Our design requires an area of about 0.227 mm^2 ($\approx$ 157k GEs) after place-and-route excluding on-chip memory. Because of the highly pipelined architecture, the design can run at 1.176 GHz. As a result, the KeyGen/Sign/Verify takes only 53.7μs/96.9μs/57.7μs. Compared to state-of-the-art implementation, we achieve a reduction of 1.4× in area and improve the runtime of KeyGen/Sign/Verify by 1.81×/1.77×/1.98×.

The physical layout of the implemented design after place-and-route is shown in Fig. 9. We have highlighted the regions for individual modules in Dilithium. One can see that majority of the area in the design is consumed by Keccak, Sampling, Reduction and NTT modules with Keccak being the largest. In our design, we are using 2 instantiations of reduction modules (marked as Reduction-1 and Reduction-2). The remainder of the space is utilized by the multiplexers connecting the various modules and the FSM based control logic.

V. HARDWARE EVALUATION AS AN ACCELERATOR

To demonstrate the effectiveness of the developed hardware accelerator, we evaluated our design on three test platforms:

TABLE III
PERFORMANCE AS AN ACCELERATOR FOR **BEST-CASE SCENARIO**: SIGNATURE IS VALID IN THE FIRST ITERATION AND IN THE **WORST-CASE SCENARIO**: REJECTION LOOP IS EXECUTED 17 TIMES BEFORE A VALID SIGNATURE IS GENERATED. THE NUMBERS ARE CPU CLOCK CYCLES ($\times 10^3$) ELAPSED WHEN THE OPERATION IS EXECUTED ON HARDWARE VERSUS ON SOFTWARE.

FPGA	Processor	KeyGen			Sign: Best-case			Sign: Worst-case			Verify		
		HW	SW	Speedup	HW	SW	Speedup	HW	SW	Speedup	HW	SW	Speedup
XC7A75T-2	Microblaze	48.2	12613.3	261.7	85.9	16210.8	188.7	796.7	84363.9	105.9	51.4	12857.7	250.1
XC7Z010-1	ARM Cortex-A9	337.5	5079.6	15.05	607.9	6907.6	11.36	5661.3	39156.1	6.92	362.2	5248.9	14.49
XCZU3EG-1	ARM Cortex-A53	254.1	2134.5	8.4	456.3	3747.8	8.21	4245.9	27240.7	6.42	271.7	2347.7	8.64

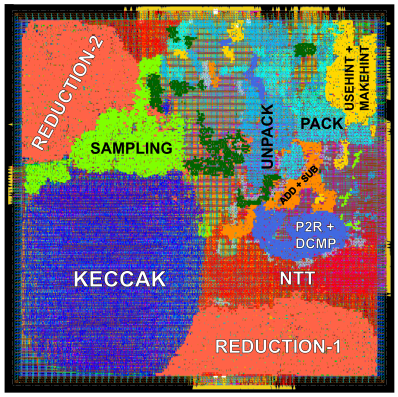

Fig. 9. Physical layout of the Dilithium-V Core after place-and-route.

Artix-7 (XC7A75T-2) with Microblaze@100MHz processor, Zynq-7000 (XC7Z010-1) with ARM Cortex-A9@667MHz and Zynq Ultrascale+ (XCZU3EG-1) with ARM Cortex-A53@1200MHz. It is important to quantify the real practical benefits of a hardware accelerator in a complete system as in many situations the full potential of an accelerator might not be achieved. Fig. 10 shows setup for the Zynq Ultrascale+ platform using the Ultra96 board.

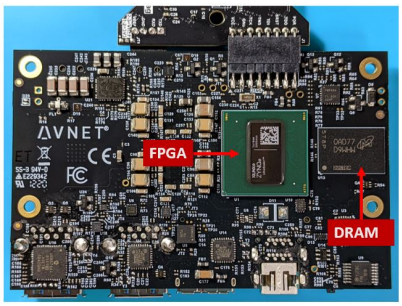

Fig. 10. Setup with Ultra96-v2 (XCZU3EG-1) as one of the hardware evaluation board.

Fig. 11 shows the block diagram of the accelerator connected as an AXI Peripheral applicable for both the Zynq Platforms. A very similar block design is used for the Microblaze setup as well. We created an AXI4-lite memory mapped register based peripheral and implemented support for control and data logic to connect with the designed Dilithium core. The AXI packet handler is responsible for receiving and sending AXI commands from CPU core and decoding them. Based on the received command, it sends the corresponding operation request to the state handler. It also communicates with the data handler for address and data read/write operations. Both the state and data handler act as a bridge between the AXI packet handler and the Dilithium accelerator. Since, the accelerator is designed to be configurable from outside, one can also use it to only accelerate individual operations like KeyGen, Sign or Verify.

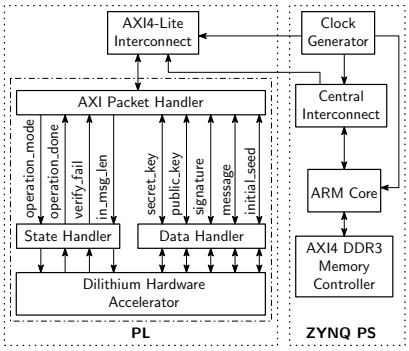

Fig. 11. Dilithium as an Accelerator on Xilinx Zynq. The same AXI peripheral IP was used for the Microblaze processor as well. Due to connectivity using AXI4-lite bus, the accelerator can easily be connected with other processors as well such as RISC-V.

In Table III, we report results for the hardware accelerator. For comparison, we chose one best-case scenario where a valid signature is generated after the first iteration itself and a worst-case scenario where the valid signature is generated after the rejection loop is executed for 17 times. It is clear from the figure that the speedup obtained is inversely proportional to the performance of the associated CPU. In case of a modern high performance CPUs like ARM cortex A53 and A9 we obtain speedups of about 6.42 to 15.05×. Whereas, for Microblaze the speedup is significantly higher at 105-261×.

VI. CONCLUSION AND FUTURE WORK

By utilizing multiple optimization strategies such as resource and control logic sharing, pre-computed LUTs, fusion of modules etc. we achieved a reduction of 24% in LUTs and FFs in area compared to the best known implementation. As a result, our design requires only 13.9k LUTs, 6.8k FFs, 4 DSPs and 35 BRAMs. To the best of our knowledge, this work presents the smallest hardware accelerator for CRYSTALS-Dilthium which can now be fit into the smallest Zynq FPGA. We also present detailed comparison with existing implementations and show that our design achieves more than 35% efficiency for Area×Time product on Zynq UltraScale+ for all the operations.

We also implemented our design on ASIC and report numbers for major components of Dilithium as well as the overall design. Compared to the state-of-the-art implementation, our design requires about 157 kGE with 0.227 mm^2 area, achieving a reduction of about 1.4×. In terms of performance, the implemented design can run at 1.176 GHz achieving an improvement of about 2.95×.

Further, this work presents the first hardware evaluation for complete Dilithium-V as an accelerator on three different platforms and demonstrate that achieved speedup is significantly high, about 105-261× for Microblaze and about 6.42-15.05× for ARM Cortex compared to the software implementations on these platforms.

In the future, we plan to integrate low-cost side-channel countermeasures in the design.

REFERENCES

[1] J. Chow, O. Dial, and J. Gambetta, "Ibm quantum breaks the 100-qubit processor barrier," *IBM Research Blog, available in https://research. ibm. com/blog/127-qubit-quantum-process-or-eagle*, 2021.

[2] C. Wang, X. Li, H. Xu, Z. Li, J. Wang, Z. Yang, Z. Mi, X. Liang, T. Su, C. Yang *et al.*, "Towards practical quantum computers: transmon qubit with a lifetime approaching 0.5 milliseconds," *npj Quantum Information*, vol. 8, no. 1, pp. 1–6, 2022.

[3] Z. Wang, S. Wei, and G. Long, "A quantum circuit design of aes," *arXiv preprint arXiv:2109.12354*, 2021.

[4] J. Zou, Z. Wei, S. Sun, Y. Luo, Q. Liu, and W. Wu, "Some efficient quantum circuit implementations of camellia," *Quantum Information Processing*, vol. 21, no. 4, pp. 1–27, 2022.

[5] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[6] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf.

[7] W. Beullens, "Breaking rainbow takes a weekend on a laptop," *Cryptology ePrint Archive*, 2022.

[8] Beullens, Ward, "Improved cryptanalysis of uov and rainbow," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 348–373.

[9] D. Pokornỳ, P. Socha, and M. Novotnỳ, "Side-channel attack on rainbow post-quantum signature," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 565–568.

[10] C. Tao, A. Petzoldt, and J. Ding, "Improved key recovery of the hfev-signature scheme," *Cryptology ePrint Archive*, 2020.

[11] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[12] S. Ricci, L. Malina, P. Jedlicka, D. Smékal, J. Hajny, P. Cibik, P. Dzurenda, and P. Dobias, "Implementing crystals-dilithium signature scheme on fpgas," in *The 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–11.

[13] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A compact and high-performance hardware architecture for crystals-dilithium," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 270–295, 2022.

[14] A. C. Mert, D. Jacquemin, A. Das, D. Matthews, S. Ghosh, S. S. Roy *et al.*, "A unified cryptoprocessor for lattice-based signature and key-exchange," *Cryptology ePrint Archive*, 2021.

[15] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of lattice-based digital signatures," *Cryptology ePrint Archive*, 2022.

[16] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal-implementing dilithium on reconfigurable hardware," *Cryptology ePrint Archive*, 2021.

[17] K. Basu, D. Soni, M. Nabeel, and R. Karri, "Nist post-quantum cryptography-a hardware evaluation study," *Cryptology ePrint Archive*, 2019.

[18] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, *Hardware Architectures for Post-Quantum Digital Signature Schemes*. Springer, 2021.

[19] D. Soni, K. Basu, M. Nabeel, and R. Karri, "A hardware evaluation study of nist post-quantum cryptographic signature schemes," in *Second PQC Standardization Conference*. NIST, 2019.

[20] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K. R. Choo, "A software/hardware co-design of crystals-dilithium signature scheme," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 2, pp. 1–21, 2021.

[21] A. F. De Abiega-L'Eglisse, K. A. Delgado-Vargas, F. Q. Valencia-Rodriguez, V. G. Gonzalez-Quiroga, G. Gallegos-Garcia, and M. Nakano-Miyatake, "Performance of new hope and crystals-dilithium postquantum schemes in the transport layer security protocol," *IEEE Access*, vol. 8, pp. 213 968–213 980, 2020.

[22] J. Wright, M. Gowanlock, C. Philabaum, and B. Cambou, "A crystals-dilithium response-based cryptography engine using gpgpu," in *Proceedings of the Future Technologies Conference*. Springer, 2021, pp. 32–45.

[23] L. Sharma and A. Mishra, "Analysis of crystals-dilithium for blockchain security," in *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*. IEEE, 2021, pp. 160–165.

[24] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: Digital signatures from module lattices," 2021, https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[25] M. J. Dworkin *et al.*, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[26] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," *Submission to NIST (Round 2)*, vol. 3, no. 30, pp. 320–337, 2009.

[27] Bertoni, Guido and Daemen, Joan and Peeters, Michaël and Van Assche, Gilles, "Keccak hardware implementation," https://keccak.team/hardware.html.

[28] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of newhope-nist on fpga using low-complexity ntt/intt," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 49–72, 2020.

[29] F. Yarman, A. C. Mert, E. Öztürk, and E. Savaş, "A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1020–1025.