

A Latency-Aware FPGA Architecture for SHAKE256 with Guarded Final-Block Construction: Targeting Post-Quantum XMSS Signatures

Maryam Taghi Zadeh^a

^aDepartment of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA

Abstract

The NIST SP 800-208 eXtended Merkle Signature Scheme (XMSS) profile XMSS-SHAKE256_10_256 repeatedly invokes SHAKE256 primitives, making field-programmable gate array (FPGA) accelerator latency sensitive to Keccak permutation time and absorb/finalization-control overhead. Established round unrolling reduces permutation-side latency but leaves residual controller work. We propose SAFE Final-Block Construction (SAFE FBC), combining guarded SHAKE256-rate-aware eligibility, direct eligible final-block construction, and sequential-finalization bypass without modifying Keccak. A controlled 12-configuration study crosses four controller/finalization organizations with one-, two-, and three-round-per-cycle datapaths over nine XMSS-oriented cases: 96-byte F and pseudorandom-function (PRF) inputs, 128-byte H and PRF_{keygen} inputs, and selected message-hash cases around the SHAKE256 rate boundary. At every fixed depth, SAFE FBC reduces aggregate $L2$ -final latency by 13 cycles versus the Structural Controller Decoupling (SCD) baseline; $L2$ -final spans the first accepted AXI input beat to the final accepted AXI output beat. At three rounds per cycle, aggregate latency falls from 305 to 292 cycles, with no case regressing. F and ordinary PRF show the largest per-call gain, from 30 to 25 cycles (16.67%). Component measurements localize the saving to absorb/finalization control; Keccak permutation latency, squeeze latency, and input-transfer span remain unchanged. SAFE FBC complements unrolling, trading latency for substantial lookup-table/slice growth with very small flip-flop overhead.

Keywords: SHAKE256, eXtended Merkle Signature Scheme (XMSS), FPGA acceleration, Keccak, latency optimization, final-block processing

1. Introduction

Stateful hash-based signatures are post-quantum digital signature schemes whose security depends on the security of their underlying hash functions rather than on additional algebraic hardness assumptions [1, 2]. Their principal deployment constraint is state management: each one-time signature (OTS) private key must be used to sign at most one message [1, 2]. NIST SP 800-208 therefore positions stateful hash-based signatures primarily for applications requiring near-term deployment, long service lifetimes, and little practical ability to migrate to another signature scheme after deployment. Authentication of firmware updates for constrained devices is given as one example [2]. SP 800-208 further requires key and signature generation to be performed in hardware cryptographic modules [2]. This work targets the SP 800-208 XMSS-SHAKE256_10_256 parameter set, based on the XMSS construction introduced in the original XMSS work and subsequently specified in RFC 8391, with the selected SHAKE256 parameterization defined by NIST SP 800-208 [1–3]. The selected profile uses SHAKE256/256 with $n = 32$ bytes [2].

For the selected profile, the evaluated SHAKE256 workload includes the fixed-format F , ordinary PRF , H , and PRF_{keygen} primitives together with selected H_{msg} characterization cases

around the 136-byte SHAKE256 rate boundary [1, 2, 4]. The F and ordinary PRF calls are especially relevant to WOTS+ because they are repeatedly invoked during chain processing; consequently, even a small reduction in their per-call latency can accumulate across repeated WOTS+ operations [1, 5].

At the permutation level, Keccak round unrolling is an established FPGA optimization that reduces the permutation cycle count by evaluating multiple rounds within one clock cycle. We therefore use one-, two-, and three-round-per-cycle (1R, 2R, and 3R) datapaths as controlled permutation-side configurations rather than as an architectural contribution of this work. Round unrolling reduces the cycles spent executing Keccak- $f[1600]$, but it does not inherently eliminate controller work required to complete and deliver the final SHAKE256 rate block before the final absorb permutation. The complementary question addressed here is whether this residual absorb/finalization-control latency can also be reduced while the Keccak datapath is held fixed. Our Final-Block Construction (FBC) approach targets this controller-side latency, allowing the permutation-side and finalization-control optimization dimensions to be evaluated separately and in combination.

To isolate the controller-side optimization from changes in the Keccak datapath, we evaluate four controller/finalization organizations. NON-SCD retains the starting SHAKE controller organization derived from the SPHINCSLET SHAKE/Keccak RTL of Deshpande *et al.* and serves as the inherited architec-

Email address: mtaghizadeh2023@fau.edu (Maryam Taghi Zadeh)

tural reference [6]. Structural Controller Decoupling (SCD) then separates absorb/finalization progress tracking from squeeze-side progress tracking while retaining the conventional finalization sequence and the shared Keccak datapath. SCD is used as a structural controlled baseline rather than introduced as a latency-reduction mechanism. Naive FBC adds a deliberately restrictive direct-final-block path as an intermediate ablation, whereas the proposed SAFE FBC replaces the restrictive eligibility condition with guarded, SHAKE256-rate-aware eligibility that directly constructs eligible final-block content and bypasses the corresponding sequential finalization-control path. At each fixed unrolling depth, the SCD-to-SAFE comparison therefore isolates the effect of SAFE FBC on the common restructured controller, whereas comparisons that change both the controller/finalization organization and the unrolling depth combine SAFE-FBC finalization-control optimization with deeper Keccak unrolling. Accordingly, this work makes three principal contributions:

- We introduce SAFE FBC, a guarded, SHAKE256-rate-aware controller mechanism that directly constructs eligible final rate-block content and bypasses sequential finalization-control work without modifying the Keccak- $f[1600]$ permutation.
- We establish a controlled 12-configuration study that crosses the four controller/finalization organizations with 1R, 2R, and 3R Keccak datapaths, allowing fixed-depth SAFE-FBC effects to be distinguished from cross-depth changes that include the effect of round unrolling.
- We evaluate the design through accelerator-level and component-level latency measurements, common-condition FPGA resource and timing-closure results, and analytical XMSS primitive- $L2$ workload projections that accumulate the measured primitive latencies under explicitly stated workload assumptions.

For system integration, the SHAKE256 accelerator is implemented in the programmable logic (PL) of the PYNQ-Z2 platform. Its Zynq-7020 system-on-chip combines an Arm processing system (PS) and PL on the same device, supporting future XMSS hardware/software partitioning [7, 8]. In the present study, however, the experimental scope is limited to the SHAKE256 hardware kernel and its accelerator-local and AXI4-Stream transaction boundaries; complete XMSS computation on the PS is neither executed nor measured as part of the reported results. The accelerator uses a 64-bit AXI4-Stream data interface, whose accepted transfers define the principal accelerator-level observation boundary used in this study. Specifically, $L2$ -final spans from the first accepted AXI4-Stream input beat to the final accepted AXI4-Stream output beat and excludes DMA submission, PS orchestration, and software-call overhead. The Experimental Methodology section defines the accelerator-local $L0$ – $L2$ measurement boundaries used in the controlled evaluation and separately defines $L3$ – $L6$ system- and software-level observation scopes, which are not quantified in the present study.

The measured results support this separation of optimization boundaries. For the evaluated XMSS-oriented characterization workload, neither the SCD restructuring alone nor the restrictive Naive FBC path produces a measured $L2$ -final reduction. In contrast, SAFE FBC removes 13 aggregate $L2$ -final cycles across the nine cases at every fixed unrolling depth, with no evaluated case incurring a latency regression. The largest per-call reduction is five cycles for F and ordinary PRF ; at 3R, each decreases from 30 to 25 cycles, a 16.67% reduction. Component-level timing localizes the entire measured 13-cycle aggregate fixed-depth saving to absorb/finalization control, while Keccak request-to-completion latency, squeeze latency, and the AXI4-Stream input span remain unchanged at the corresponding unrolling depth. When SAFE FBC is combined with deeper round unrolling, moving from the 1R SCD baseline to 3R SAFE FBC reduces the nine-case aggregate $L2$ -final latency from 464 to 292 cycles, or 37.07%; this cross-depth reduction reflects both design dimensions and is not attributed to SAFE FBC alone. In the analytical full-tree key-generation primitive- $L2$ projection, F and ordinary PRF account for 99.17% of the projected SAFE-FBC cycle saving. The implementation results further expose a latency–area tradeoff: SAFE FBC substantially increases LUT and slice utilization while adding very little FF overhead. The evaluation in this work focuses on XMSS-SHAKE256 workloads, but SAFE FBC modifies SHAKE256 rate-block handling and controller behavior rather than XMSS-specific logic. This suggests that the same approach could be considered for other SHAKE256-based systems after analyzing their input formats and final-block occupancy conditions.

2. Background and Preliminaries

2.1. XMSS-SHAKE256_10_256 and WOTS+ Hash Dependencies

XMSS is a stateful hash-based signature scheme that combines the Winternitz One-Time Signature Plus (WOTS+) scheme with a binary Merkle tree [1]. Each XMSS leaf is derived from a WOTS+ public key, and the authentication path included in an XMSS signature enables the verifier to reconstruct the Merkle-tree root. The private-key state includes an index identifying the next unused WOTS+ key pair [1]. Within an individual WOTS+ chain, each step depends on the preceding chain value, so the steps of that chain are sequentially dependent.

The selected SP 800-208 XMSS-SHAKE256_10_256 parameter set uses $n = 32$ bytes, tree height $h = 10$, Winternitz parameter $w = 16$, and WOTS+ length $\text{len} = 67$ [2]. This parameter set is distinct from the similarly named XMSS-SHAKE_10_256 profile in RFC 8391, which uses SHAKE128 rather than SHAKE256 for $n = 32$ [1, 2].

For the randomized hash computation within a WOTS+ chain transition, two ordinary PRF evaluations derive the keyed-hash key and bitmask, after which F is evaluated on the masked chain value [1, 5]. Thus, each transition involves two ordinary PRF evaluations followed by one F evaluation. This repeated primitive dependence is relevant to WOTS+ chain-processing latency,

while the exact SHAKE256 formatting of the corresponding primitive inputs is defined in Section 2.3.

2.2. SHAKE256 Sponge and Keccak-f[1600]

For the XMSS profile considered in this work, SHAKE256 is the extendable-output function specified in FIPS 202 [4]. SHAKE256 uses a 1600-bit Keccak state with a capacity of $c = 512$ bits and a rate of $r = 1088$ bits, or 136 bytes [4]. During absorption, the padded input is processed in r -bit blocks, with each block XORed into the rate portion of the state before the Keccak-f[1600] permutation is applied. During squeezing, output is taken from the rate portion, with additional permutations applied when further output blocks are required. The 136-byte rate therefore determines the amount of input that can be absorbed between successive permutation invocations.

The Keccak-f[1600] state is represented as a 5×5 array of 64-bit lanes, and the permutation consists of 24 rounds [4]. Each round applies the five step mappings θ , ρ , π , χ , and ι . The θ mapping combines column parities, ρ performs fixed rotations within the lanes, and π rearranges the lane positions. The non-linear χ mapping updates each row, after which ι incorporates a round-dependent constant into lane (0, 0) [4]. These operations constitute the internal Keccak permutation; the final-block construction considered in this work operates outside these round transformations.

2.3. XMSS SHAKE256 Input Formation and Final-Block Semantics

The SHAKE256 rate-block position at which finalization occurs follows from the standardized formatting of the XMSS internal functions. RFC 8391 defines F with an n -byte key and an n -byte input, PRF with an n -byte key and a 32-byte input, H with an n -byte key and a $2n$ -byte input, and H_{msg} with a $3n$ -byte key and an arbitrary-length message [1]. NIST SP 800-208 instantiates these functions for XMSS-SHAKE256_10_256 using SHAKE256/256 with $n = 32$ and assigns the 32-byte identifiers toByte(0, 32) through toByte(4, 32) to F , H , H_{msg} , PRF , and PRF_{keygen} , respectively [2]. The resulting formatted inputs to F and ordinary PRF are 96 bytes, while H occupies 128 bytes. SP 800-208 also generates WOTS+ private-key elements as $PRF_{\text{keygen}}(S_{\text{XMSS}}, SEED \parallel ADRS)$, producing another 128-byte input because the function identifier, S_{XMSS} , $SEED$, and $ADRS$ each contribute 32 bytes [2]. For randomized message hashing, the $3n = 96$ -byte key comprises an n -byte random value, the XMSS root, and the encoded signature index; including the 32-byte function identifier gives H_{msg} a fixed 128-byte portion followed by the external message M [1, 2]. Table 1 summarizes these standardized function formats together with the corresponding input cases evaluated in this work.

These lengths determine the occupancy of the SHAKE256 rate block when finalization begins. The 96-byte F and ordinary PRF inputs leave 40 bytes in the current rate block, while each 128-byte case leaves 8 bytes. For the byte-aligned inputs evaluated here, the SHAKE256 suffix and multi-rate padding

can be represented at the byte level by 0x1F following the final input byte, with the terminal padding bit set at the end of the rate block [4]. The $H_{\text{msg}129}$ and $H_{\text{msg}135}$ cases leave 7 and 1 byte, respectively. In the 135-byte case, only one padding byte remains, so the 0x1F contribution and terminal padding bit occupy the same final rate byte, yielding 0x9F as specified for the one-byte XOF-padding case in FIPS 202 [4]. An input of 136 bytes fills the rate block exactly, so finalization requires a new rate block, whereas the 137-byte case begins that second block with one external-message byte before finalization [4]. Figure 1 visualizes these byte-level occupancies for the complete nine-case characterization set.

2.4. Keccak Round Unrolling in Hardware

Keccak round unrolling reduces the permutation cycle count by cascading multiple round functions combinationally within one clock interval [9]. In a one-round-per-cycle (1R) datapath, the state is registered after each Keccak round, whereas the 2R and 3R datapaths cascade two and three rounds, respectively, before registering the updated state. The 24-round Keccak-f[1600] permutation therefore requires 24, 12, and 8 active permutation cycles for 1R, 2R, and 3R, respectively. Cascading additional rounds increases combinational depth and can tighten logic and routing timing constraints. Round unrolling therefore reduces the permutation-side cycle count, while, in the evaluated SHAKE256 architecture, the rate-dependent final-block processing described above remains in the surrounding absorb/finalization control.

3. Related Work

3.1. FPGA SHA-3 and SHAKE Microarchitectures

FPGA implementations of SHA-3 candidates and standardized SHA-3 span a wide range of datapath parallelism. Hom-sirikamol *et al.* study folding, unrolling, and pipelining across the Round-3 SHA-3 candidates and report the corresponding FPGA throughput–area tradeoffs, including results for Keccak [10]. For compact standardized SHA-3 implementations, Jungk and Stöttinger study lane- and slice-oriented architectures that serialize processing within individual Keccak rounds, reducing area at the cost of longer execution time. Their slice-oriented architecture can process one or several slices per clock cycle, exposing an area–throughput tradeoff [11]. Sideris *et al.* provide an FPGA example with an unrolling factor of two, evaluating two rounds per cycle and reducing the 24-round permutation to 12 round-processing cycles [9]. These works establish serialization, unrolling, and related datapath-level tradeoffs as prior Keccak implementation choices.

Optimization outside the Keccak permutation is also established in prior hardware, but the corresponding mechanisms operate at different points in the hashing workflow, as summarized in Fig. 2. At the inter-block boundary, prior SHA-3 FPGA architectures overlap preparation or transfer of the next message block with processing of the current block, reducing subsequent

Table 1: XMSS SHAKE256/256 functions and evaluated input shapes for the SP 800-208 XMSS-SHAKE256_10_256 profile

Function	Formatted SHAKE256 input	Role	Length $n = 32$	Evaluated case(s)
F	$\text{toByte}(0, 32) \parallel \text{KEY} \parallel M$	WOTS+ chain hash on an n -byte masked chain value	$32 + 32 + 32 = 96$ B	F_{96}
H	$\text{toByte}(1, 32) \parallel \text{KEY} \parallel M$	Keyed hash for L-tree and Merkle-tree node computation	$32 + 32 + 64 = 128$ B	H_{128}
H_{msg}	$\text{toByte}(2, 32) \parallel \text{KEY} \parallel M$	Randomized message hash with a $3n$ -byte key and variable-length external message	$128 + M $ B	$H_{\text{msg}128}, H_{\text{msg}129}, H_{\text{msg}135}, H_{\text{msg}136}, H_{\text{msg}137}$
PRF	$\text{toByte}(3, 32) \parallel \text{KEY} \parallel M$	Pseudorandom generation of keyed-hash keys and bitmasks	$32 + 32 + 32 = 96$ B	PRF_{96}
PRF_{keygen}	$\text{toByte}(4, 32) \parallel \text{KEY} \parallel M$	SP 800-208 pseudorandom generation of WOTS+ private-key elements	$32 + 32 + 64 = 128$ B	$PRF_{\text{keygen}128}$

Note: For the evaluated SP 800-208 PRF_{keygen} semantics, $\text{KEY} = S_{\text{XMSS}}$ and $M = SEED \parallel \text{ADRS}$. The five H_{msg} lengths are characterization points selected in this work under the standardized variable-length H_{msg} construction; they are not five fixed input lengths prescribed by SP 800-208.

block-loading overhead at the cost of additional buffering and associated input-handling logic [10, 11]. At the inter-call boundary, Saarinen’s SLoTH architecture reduces processor involvement across successive SLH-DSA hash invocations by retaining recurring seed and address fields in hardware and automating hash-primitive formatting, padding, and Winternitz iteration [12]. The present work instead targets intra-call finalization: residual final-block and absorb/finalization-control work within a single SHAKE256 invocation. As established in Section 2.3, this distinction is especially relevant to the 96-byte F and ordinary PRF calls of the evaluated XMSS profile, whose inputs terminate within one 136-byte SHAKE256 rate block and therefore provide no subsequent message block whose loading can be overlapped with processing of the current block. SAFE FBC accordingly operates at a different optimization boundary by bypassing eligible sequential finalization-control work while

leaving the Keccak permutation unchanged.

3.2. SHAKE and Hash Acceleration in PQC Hardware

Broader PQC accelerators expose additional ways of organizing Keccak/SHAKE computation. In a SABER ASIC accelerator, Imran *et al.* serially connect two Keccak sponge/round-processing functions in a double-sponge organization, reducing the reported Keccak-core schedule from 28 to 14 cycles [13]. LightHD uses a different organization for CRYSTALS-Dilithium: Ni *et al.* operate two independent Keccak cores with a fixed 26-cycle offset, interleaving their absorb, permutation, and squeeze activity instead of extending the critical path through cascaded Keccak processing [14]. Pandey *et al.* move resource sharing to the algorithm level in their unified HQC architecture, where common modules, including SHAKE256, are shared

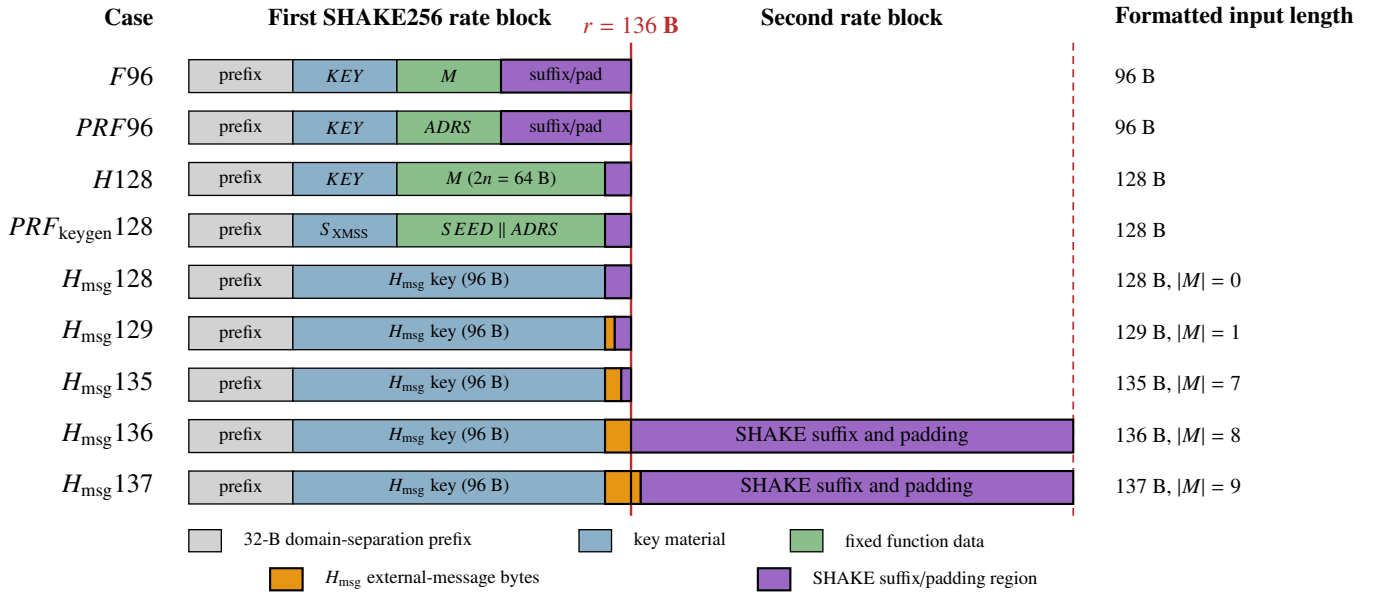


Figure 1: Byte-level composition and SHAKE256 rate-block occupancy for the nine evaluated XMSS-SHAKE256_10_256 characterization cases. Each input includes a 32-byte domain-separation prefix. Inputs up to 135 bytes leave room for SHAKE finalization in the first 136-byte rate block; 136 bytes fills the first block, while the 137-byte case begins a second block before finalization. Both rate blocks are shown at equal overall width. One-byte regions in selected H_{msg} cases are schematically widened for visibility; labels indicate the actual byte counts.

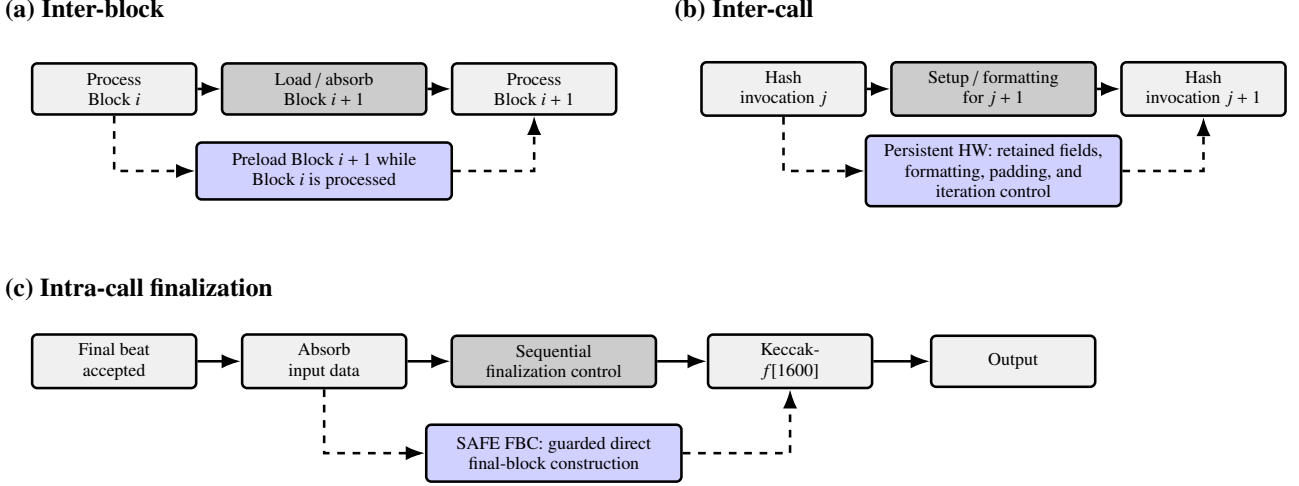


Figure 2: Comparison of non-permutation optimization boundaries. (a) Inter-block optimization overlaps preparation of the next block with processing of the current block. (b) Inter-call optimization reduces preparation overhead between successive hash invocations. (c) SAFE FBC targets intra-call finalization by bypassing eligible sequential finalization-control work after the final input beat is accepted.

across KeyGen, Encaps, and Decaps [15]. These works respectively demonstrate permutation-side cascading, independent-core scheduling, and algorithm-level resource sharing, and therefore address architectural dimensions distinct from the intra-call finalization-control path studied here.

Hash-based-signature accelerators also exploit fixed input structures and repeated hash dependencies at the scheme level. In their RISC-V XMSS co-design, Wang *et al.* progress from a general-purpose SHA-256 accelerator, whose padding remains in software, to an XMSS-specific accelerator that accepts fixed-length inputs and performs SHA-256 padding in hardware; this specialized core is subsequently used for WOTS-chain and XMSS-leaf acceleration [16]. Their work therefore establishes fixed-length hardware padding and WOTS-chain acceleration as prior techniques. For SHAKE256-based hash-based signatures, Deshpande *et al.* present SPHINCSLET and a full-width iterative SHAKE256 datapath in which the Keccak round logic is combinational and each round result is registered before being fed back [6]. We use the corresponding open-source SHAKE/Keccak RTL as the source of the inherited NON-SCD reference organization. SAFE FBC differs from these prior padding and scheme-level mechanisms in that it uses a guarded, SHAKE256-rate-aware direct final-block path that bypasses eligible sequential finalization-control work without modifying the Keccak permutation.

At the system-integration level, Lopez-Valdivieso and Cumplido present a hardware/software architecture for the SPHINCS+ hash-based signature scheme [17]. Such work illustrates scheme-level partitioning of hash-based-signature processing across hardware and software resources. The scope of the present study is narrower: the PYNQ-Z2 serves as the integration platform for the programmable-logic SHAKE256 accelerator, while complete XMSS computation on the processing system is neither executed nor measured in the reported evaluation. Accordingly, the results reported here characterize the SHAKE256 hardware kernel and its accelerator-local transaction boundaries rather than a measured end-to-end XMSS hardware/software

implementation.

Direct XMSS hardware implementations further demonstrate scheme-level acceleration beyond individual hash invocations. Cao *et al.* present a full-hardware implementation of XMSS, establishing complete scheme-level hardware realization as prior work [18]. Their subsequent multicore XMSS architectures use multiple hash cores to reduce the latency of key generation and signature generation [19]. These designs operate at the complete-XMSS architecture level and exploit parallel hash resources to accelerate larger scheme-level workloads. They are therefore conceptually complementary at a different optimization level rather than direct substitutes for the primitive-latency optimization studied here: SAFE FBC targets residual finalization-control work inside an individual SHAKE256 invocation while holding the Keccak configuration fixed.

Thoma *et al.* likewise show how primitive-level hash latency interacts with scheme-level parallelism in configurable XMSS/LMS hardware. Their architecture separates signature logic from configurable parallel hash cores through a hash controller, allowing the number of hash cores to be configured independently of the surrounding signature logic [20]. Their reported XMSS evaluation includes a configuration using $n = 32$, $h = 10$, and $w = 16$, corresponding to 67 WOTS+ chains [20]. Hash operations are dynamically scheduled across the available hash resources and WOTS+ chaining modules [20]. Consistent with the intra-chain dependency described in Section 2.1, an individual WOTS+ chain step still depends on the preceding chain value and cannot execute until the required pseudorandom key and mask are available [20]. Their XMSS evaluation uses SHA2-256 rather than SHAKE256 and considers a complete signature accelerator, so its absolute performance is not directly compared with the SHAKE256 transaction measurements reported here [20]. The study nevertheless shows that intra-chain hash dependencies remain relevant even when inter-chain parallelism and multiple hash resources are available. The present work targets the primitive side of that dependency by reducing eligible finalization-control latency within individual SHAKE256

calls; the later XMSS-level workload results are therefore analytical primitive-*L2* projections rather than measured end-to-end speedups.

3.3. Synthesis and Scope

The reviewed literature establishes several distinct optimization levels for SHA-3/SHAKE and hash-based-signature hardware. Keccak serialization, unrolling, and related datapath organizations modify permutation-side processing [9–11], while buffering, fixed-length hardware padding, persistent formatting support, multiple hash resources, hardware/software partitioning, and WOTS-chain scheduling address processing or coordination elsewhere in the hashing or signature workflow [11, 12, 16–20]. These mechanisms are treated as prior techniques rather than contributions of this work. The question addressed here is whether residual final-block formation and absorb/finalization-control latency can be reduced for the XMSS-oriented SHAKE256 characterization cases while the Keccak configuration is held fixed. SAFE FBC is therefore positioned as a controller-side complement to established permutation-side and scheme-level optimizations, not as a replacement for round unrolling, prior fixed-length hardware-padding approaches, multicore hashing, or complete-signature acceleration.

Fault and side-channel protection are outside the scope of this study. Amiet *et al.* demonstrate supply-voltage-glitch fault injection against an FPGA SPHINCS+ implementation and investigate hardware-replication countermeasures [21]; the present architecture does not evaluate such countermeasures and makes no claim regarding fault, side-channel, or broader physical-attack resistance.

4. Proposed Architecture

4.1. Accelerator Overview and Design Provenance

The SHAKE256 accelerator development in this work starts from the open-source SHAKE/Keccak RTL associated with the SPHINCSLET architecture of Deshpande *et al.* [6]. Their SHAKE256 implementation uses a full-width, round-based iterative Keccak-*f*[1600] datapath in which the round transformations are implemented combinationally and the output of each round is registered and fed back for the next round [6]. We use this RTL organization as the starting NON-SCD reference. From this reference, SCD restructures the controller, Naive FBC introduces a deliberately restrictive direct-final-block path, and SAFE FBC combines the SCD control structure with guarded, SHAKE256-rate-aware final-block construction. At each fixed unrolling depth, the four organizations therefore retain a common Keccak processing basis for comparing controller organization and final-block handling.

Across all four organizations, the accelerator retains the same top-level dataflow. Input data are accepted through a 64-bit AXI4-Stream interface and transferred to an internal buffer under SHAKE-controller control. During absorption, the controller tracks the occupancy of the active rate block and initiates a Keccak-*f*[1600] permutation when an intermediate or

final block is ready. After permutation completion, the updated 1600-bit state is returned from the Keccak datapath, and its SHAKE256 rate portion is made available for squeezing and delivered through the 64-bit output stream. The controller also coordinates input back-pressure, final-input handling, permutation request and completion, and output availability. This organization is common to NON-SCD, SCD, Naive FBC, and SAFE FBC; their differences are confined to control and final-block processing around the shared Keccak datapath.

The NON-SCD reference controller uses six states to coordinate the SHAKE processing sequence: *S_IDLE*, *S_ABSORB*, *S_FULL*, *S_APPEND*, *S_LAST_FULL*, and *S_SQUEEZE*. Input words are accumulated during *S_ABSORB*; when a complete rate block is reached, *S_FULL* initiates an intermediate Keccak permutation. For a final input that leaves unused space in the active rate block, the controller enters *S_APPEND*, where final-block completion proceeds through 64-bit buffer advancement until the final block is ready. The controller then enters *S_LAST_FULL* for the final absorb permutation before transitioning to *S_SQUEEZE*. In the inherited organization, absorb/finalization and squeeze activity reuse buffer-progression control that is subsequently separated by SCD. This shared phase-progress organization motivates the restructuring described next.

4.2. Structural Controller Decoupling

Structural Controller Decoupling (SCD) separates the progress tracking used during absorb/finalization from that used during squeezing while retaining the same shared physical data buffer and the conventional SHAKE processing sequence. On the absorb/finalization side, accepted input words and sequential final-block completion advance the buffer while an independent occupancy tracker records progress within the active rate block and detects block completion. On the squeeze side, accepted output transfers advance the same physical buffer, but output progress and buffer exhaustion are tracked independently of absorb/finalization progress. As illustrated in Fig. 3, SCD therefore preserves the common storage datapath while separating the phase-specific control used to determine progress through absorption/finalization and squeezing. This separation establishes a well-defined absorb/finalization control path on which the subsequent SAFE FBC and SAFE-FBC mechanisms operate, while the squeeze path is retained unchanged. SCD does not alter the conventional *S_APPEND* finalization sequence or the Keccak datapath and therefore serves as the structural baseline for the two FBC variants rather than as a latency-reduction mechanism.

4.3. From Restrictive Naive FBC to Rate-Aware SAFE FBC

The first Final-Block Construction variant, denoted Naive FBC, was introduced as a deliberately restrictive ablation to determine whether direct construction of the final SHAKE256 rate block could bypass the conventional sequential finalization path. The direct path is considered only when the final input beat is accepted during absorption. Naive FBC further requires the active rate block to contain no previously accepted message bytes at that point, together with a valid fit of the constructed

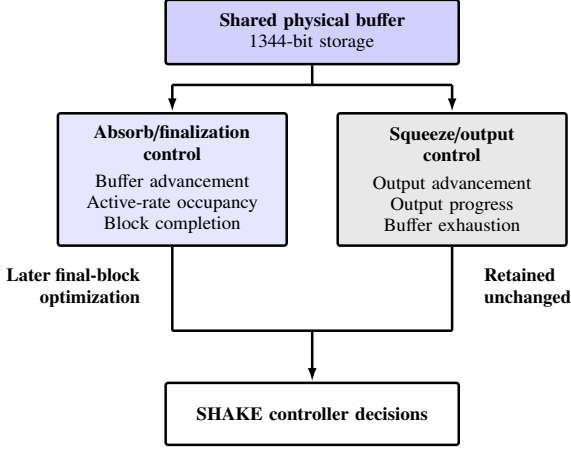


Figure 3: Conceptual hardware organization of Structural Controller Decoupling. A shared physical buffer is retained, while absorb/finalization and squeeze/output progress are controlled independently. The absorb/finalization path is the branch modified by the subsequent final-block optimizations; the squeeze/output path remains unchanged.

final representation within the SHAKE256 rate. When these conditions are satisfied, the final input word and SHAKE delimiter are placed directly into a zero-initialized final block, the constructed block is loaded into the shared buffer, and the controller bypasses the sequential `S_APPEND` completion path before entering the existing final-absorb stage. The final Keccak permutation and subsequent squeeze processing are unchanged.

Naive FBC therefore isolates the architectural opportunity associated with eliminating sequential final-block completion, but its empty-block requirement is narrower than the condition actually needed for safe direct construction. SAFE FBC retains the same bypass objective while replacing this restriction with an occupancy-aware rate-fit criterion: a partially occupied rate block may use the direct path when the accepted final input and its required finalization representation fit completely within the remaining SHAKE256 rate space. The two variants consequently differ in their eligibility and final-block-placement logic rather than in the downstream Keccak or squeeze processing. Fig. 4 summarizes this architectural distinction.

The empty-block rule excludes important partially occupied rate-block conditions. For a 96-byte formatted input, 88 bytes are already present in the active rate block before the final full 64-bit word is accepted; for a 128-byte formatted input, the corresponding occupancy is 120 bytes. Such cases cannot satisfy the Naive empty-block guard even though sufficient SHAKE256 rate space remains to form the final representation directly. Partial final beats near the rate boundary present another case in which eligibility and cycle-saving opportunity need not coincide, because the conventional controller may already avoid repeated `S_APPEND` advancement. These occupancy patterns motivate a guard based on the actual remaining rate space rather than on whether the active rate block is empty. Because SHAKE256 absorbs external input only through its 1088-bit rate while the 512-bit capacity lies outside the input region [4], any direct final-block construction must likewise remain confined to the SHAKE256 rate window.

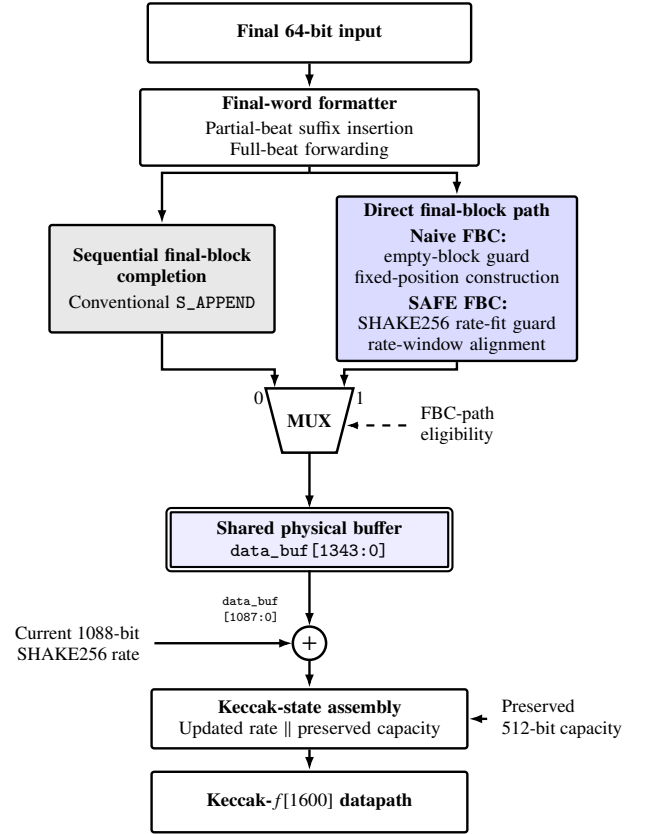


Figure 4: Comparison of the Naive- and SAFE-FBC direct final-block paths within the common SHAKE256 architecture. Naive FBC uses an empty-block eligibility condition with fixed-position construction, whereas SAFE FBC uses a SHAKE256 rate-fit guard with rate-window alignment. All remaining finalization and Keccak processing is shared between the two variants.

4.4. SAFE FBC Data and Control Path

SAFE FBC implements the rate-aware criterion by testing whether the accepted final input and its required finalization representation fit completely within the remaining SHAKE256 rate space. Eligibility is evaluated only during the absorb phase when the final input beat is accepted, so direct-path activation is tied to completion of an accepted input transfer rather than to the final-input indication alone. The required remaining rate space depends on whether the accepted final beat is full or partial. For a full 8-byte final beat, direct construction must accommodate the eight message bytes and a following 8-byte slot containing the SHAKE delimiter and zero fill. For a partial 1–7-byte final beat, the valid message bytes and delimiter occupy the same final 8-byte slot. If b denotes the number of bytes already occupying the active SHAKE256 rate block before final-beat acceptance, the SAFE-FBC rate-fit condition is

$$b + \begin{cases} 16, & \text{full final beat,} \\ 8, & \text{partial final beat} \end{cases} \leq 136. \quad (1)$$

Thus, the full- and partial-beat guards in (1) correspond to $b \leq 120$ and $b \leq 128$, respectively. Unlike the Naive empty-block requirement, this condition permits direct final-block construction when the active rate block is already partially occupied,

provided that the complete constructed representation remains within the 136-byte SHAKE256 rate.

When the SAFE-FBC guard is satisfied, the constructed and aligned final rate block is loaded directly into the shared buffer, and the controller proceeds from S_ABSORB to S_LAST_FULL without traversing the sequential S_APPEND path. If the final beat is accepted outside the required absorb-phase condition, or if the rate-fit criterion is not satisfied, the direct path is disabled and the conventional SCD finalization sequence is retained. A completely occupied 136-byte rate block, for example, leaves no room for finalization within that block and therefore cannot satisfy the direct rate-fit condition. In either path, S_LAST_FULL , the final Keccak- $f[1600]$ permutation, and subsequent squeeze processing remain unchanged. SAFE FBC therefore shortens only the controller path used to prepare eligible final rate blocks, as summarized in Fig. 5.

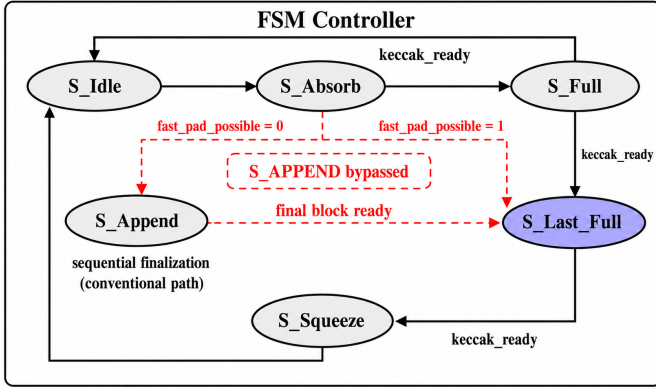


Figure 5: Controller-state view of the SAFE-FBC finalization path. Eligible final beats bypass the conventional S_APPEND state and proceed directly from S_ABSORB to S_LAST_FULL using the constructed final rate block, whereas ineligible cases retain the conventional finalization path. The subsequent final-permutation and squeeze progression is unchanged.

Once the SAFE-FBC eligibility condition is satisfied, the final representation is constructed while preserving the message data already accumulated for the active rate block. For a full 8-byte final beat, the accepted word is appended to the buffered message and the SHAKE delimiter $0x1F$ is placed in the following byte position; for a partial final beat, the valid message bytes and the delimiter are incorporated into the same 64-bit word. The remaining positions are zero filled. Because SAFE FBC may activate when earlier message bytes already occupy part of the active rate block, the constructed representation must then be aligned according to the existing rate-block occupancy before it is loaded into the shared buffer.

To perform this placement without advancing the constructed data iteratively through the buffer, SAFE FBC uses a combinational variable-alignment datapath. Let u denote the total rate-window occupancy represented by the direct path after accounting for the accepted final beat and its finalization slot, so that $u = b + 16$ for a full final beat and $u = b + 8$ for a partial final beat. For SHAKE256, the required displacement is

$$s = 8(136 - u). \quad (2)$$

The expression in (2) follows from the remaining number of bytes in the 136-byte rate, with the factor of eight converting the displacement from bytes to bits. Because the direct representation is aligned at the 8-byte input-word granularity, s assumes up to 17 values from 0 to 1024 bits in 64-bit increments. Five logical binary-selection decisions are therefore sufficient to represent these word-aligned positions. Conceptually, the variable alignment can be decomposed into conditional-shift levels of 64, 128, 256, 512, and 1024 bits, following the conventional logarithmic barrel-shifter organization described in [22]. In the RTL, however, the alignment is expressed as a variable shift rather than as five explicitly instantiated stages; the exact combinational topology is determined by FPGA synthesis. The barrel-shifter interpretation is therefore an implementation model for rate-aware direct placement, not a separate architectural contribution. The logical alignment function is illustrated in Fig. 6.

For SHAKE256, the SAFE-FBC direct path operates only on the 1088-bit rate region of the physical buffer; its remaining 256 buffer bits are forced to zero and must not be confused with the 512-bit Keccak capacity. During final absorption, only the 1088-bit rate portion of the current 1600-bit sponge state is XORed with the constructed rate block, whereas the 512-bit capacity is preserved. The terminal multi-rate padding bit is applied by the existing S_LAST_FULL final-absorb logic. SAFE FBC therefore changes the construction, alignment, and controller delivery of an eligible final rate block without changing the SHAKE256 padding semantics or the subsequent Keccak- $f[1600]$ permutation.

Fig. 7 places the controller behavior shown in Fig. 5, together with the preceding SAFE-FBC data-path mechanisms, within the overall SHAKE256 accelerator. The shared 1344-bit physical buffer and Keccak- $f[1600]$ datapath are retained, while SCD keeps absorb/finalization progress tracking separate from squeeze-side progress tracking. SAFE FBC is confined to the absorb/finalization path: acceptance of the final input beat is combined with rate-occupancy information to evaluate eligibility, an eligible final representation is constructed and aligned within the 1088-bit SHAKE256 rate window, and the direct or conventional path is selected before the final permutation request. The squeeze path and Keccak round datapath remain independent of this selection.

Algorithm 1 summarizes the controller transformation from the inherited NON-SCD organization to SCD and then to SAFE FBC. The first stage separates absorb/finalization and squeeze progress tracking while retaining the shared physical buffer, conventional finalization behavior, and Keccak datapath. The second stage adds the guarded, rate-aware direct-final-block path on top of the SCD organization.

For an eligible accepted final beat, SAFE FBC therefore changes the controller path used to prepare and deliver the final SHAKE256 rate block rather than the Keccak permutation itself. When any eligibility condition is not satisfied, the architecture retains the ordinary SCD absorb/finalization sequence. The optimization consequently preserves the conventional path for noneligible transactions while bypassing sequential finalization-control work only when the guarded rate-aware condition per-

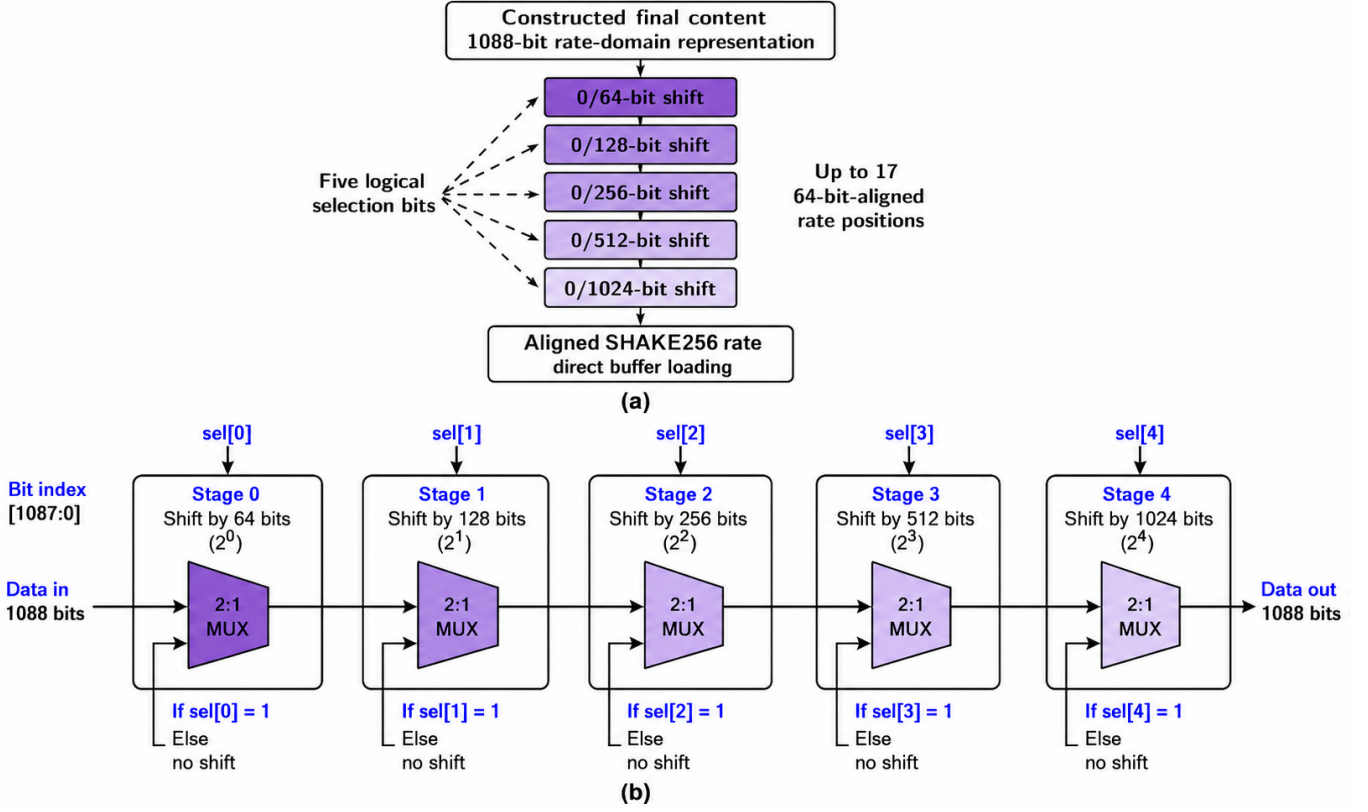


Figure 6: SAFE-FBC SHAKE256 rate-alignment datapath. (a) Logical decomposition of the word-aligned placement function within the 1088-bit SHAKE256 rate region using five binary-selection levels for 64-, 128-, 256-, 512-, and 1024-bit displacements, covering 17 aligned positions from 0 to 1024 bits. (b) Conceptual barrel-shifter realization of the same alignment function using cascaded binary selections [22]. Both panels represent the same combinational operation and do not imply pipeline stages or additional clock cycles. In the RTL, the alignment is expressed as a variable shift, with the final structure determined by FPGA synthesis.

Algorithm 1 Controller Transformation from NON-SCD to SCD + SAFE FBC

Require: Baseline NON-SCD SHAKE256 core C_{NS} with 64-bit AXI4-Stream interface
Ensure: SCD + SAFE-FBC core C_{SAFE} with guarded conventional fallback

- 1: $r \leftarrow 136$ bytes
- 2: $C_{SCD} \leftarrow C_{NS}$
- 3: **Stage 1: Structural Controller Decoupling**
- 4: Separate absorb/finalization progress from squeeze progress
- 5: Preserve the shared buffer, conventional S_APPEND path, and Keccak- $f[1600]$ datapath
- 6: **Stage 2: Guarded SAFE Final-Block Construction**
- 7: **if** a final input beat is accepted in S_ABSORB **then**
- 8: $b \leftarrow$ active-rate occupancy
- 9: $n \leftarrow \text{NormalizedValidBytes}(\text{final-beat metadata})$
- 10: $\delta \leftarrow 16$ if $n = 8$; $\delta \leftarrow 8$ if $1 \leq n \leq 7$
- 11: $e \leftarrow [(n = 8) \vee (1 \leq n \leq 7)] \wedge (b + \delta \leq r)$
- 12: **if** $e = 1$ **then**
- 13: Construct the final representation and insert $0x1F$ after the final valid message byte
- 14: $s \leftarrow 8(r - (b + \delta))$
- 15: Align only within the 1088-bit SHAKE256 rate and force buffer bits [1343 : 1088] to zero
- 16: Load the 1344-bit physical buffer and set $state \leftarrow S_{LAST\_FULL}$
- 17: **else**
- 18: Retain the conventional SCD finalization path
- 19: **end if**
- 20: **end if**
- 21: Execute the unchanged final Keccak- $f[1600]$ permutation and squeeze processing
- 22: $C_{SAFE} \leftarrow C_{SCD} + \text{SAFE-FBC path}$
- return** C_{SAFE}

mits direct construction.

4.5. Integration With 1R, 2R, and 3R Keccak Datapaths

SAFE FBC is evaluated with the 1R, 2R, and 3R Keccak- $f[1600]$ datapaths defined in Section 2.4, which require 24, 12,

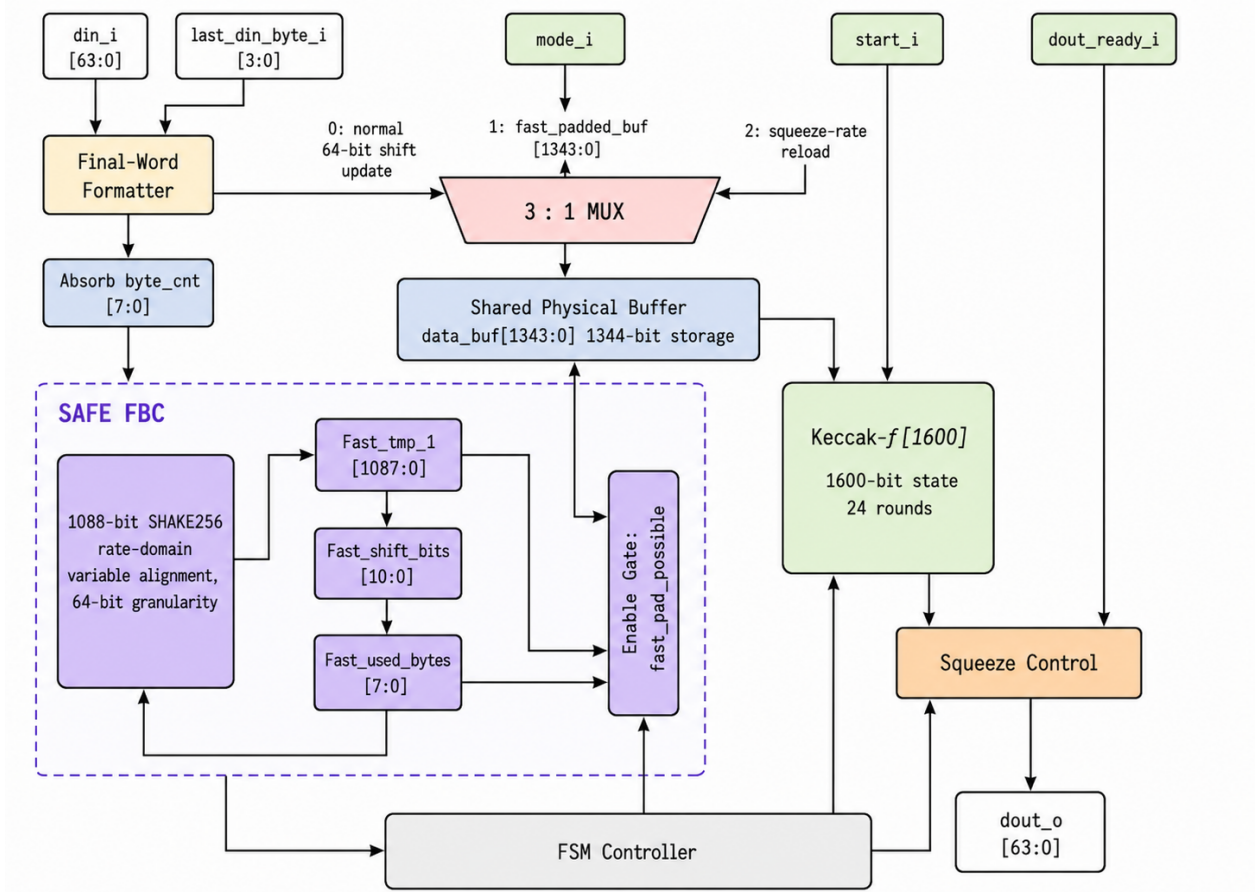


Figure 7: Hardware architecture of the SAFE-FBC SHAKE256 accelerator.

and 8 active permutation cycles, respectively. SAFE FBC is located on the absorb/finalization path upstream of the final permutation request: it evaluates eligibility, constructs and aligns an eligible SHAKE256 rate block, and bypasses the corresponding sequential finalization-control path without changing the Keccak round datapath. Round unrolling and SAFE FBC therefore act on different latency components. Increasing unrolling depth reduces permutation-side latency, whereas SAFE FBC targets residual absorb/finalization-control latency outside the permutation. At each fixed unrolling depth, NON-SCD, SCD, Naive FBC, and SAFE FBC use the same Keccak round datapath, allowing controller/final-block effects to be separated from permutation-side unrolling effects.

5. Experimental Methodology

5.1. Evaluated Architectures and Controlled Comparison Matrix

The evaluation crosses the four controller/finalization organizations defined in Section 4—NON-SCD, SCD, Naive FBC, and SAFE FBC—with the 1R, 2R, and 3R Keccak- $f[1600]$ datapaths, yielding 12 configurations evaluated under common implementation conditions. At each fixed unrolling depth, the

NON-SCD-to-SCD comparison characterizes controller restructuring, the SCD-to-SAFE FBC comparison characterizes the restrictive direct-final-block ablation, and the SCD-to-SAFE-FBC comparison is the principal fixed-depth comparison used to isolate SAFE FBC. Comparisons of 1R, 2R, and 3R within the same controller/finalization organization characterize the separate round-unrolling dimension. Comparisons that change both the controller/finalization organization and the unrolling depth combine the two design dimensions and are therefore not interpreted as FBC-only comparisons.

5.2. XMSS-Oriented Characterization Workload

The accelerator is characterized using nine XMSS-oriented SHAKE256 formatted-input cases derived for this study from the standardized function formats of the NIST SP 800-208 XMSS-SHAKE256_10_256 profile [2]. The characterization set comprises F_{96} , PRF_{96} , H_{128} , $PRF_{keygen 128}$, $H_{msg 128}$, $H_{msg 129}$, $H_{msg 135}$, $H_{msg 136}$, and $H_{msg 137}$, where the numerical suffix denotes the total number of bytes presented to SHAKE256. For every case, the requested SHAKE256 output length is 32 bytes. The F , ordinary PRF , H , and PRF_{keygen} cases represent fixed formatted input shapes for the selected profile, whereas the five H_{msg} lengths are characterization points selected under the standardized variable-length H_{msg} construction. These points span inputs below, exactly at, and beyond the 136-byte

SHAKE256 rate boundary [4], thereby exercising the final-block conditions relevant to the evaluated architectures. They are experimental characterization lengths and must not be interpreted as five fixed H_{msg} calls prescribed by SP 800-208. The byte-level composition of all nine cases is summarized in Table 1.

5.3. Functional Verification and Correctness Criteria

Before a latency result is admitted to the comparison, the corresponding accelerator output is required to satisfy the same case-level correctness criterion across all 12 configurations. Each formatted input is supplied as a sequence of 64-bit AXI4-Stream beats, and a beat is accepted only when the corresponding TVALID and TREADY signals are asserted simultaneously [23]. The terminal accepted transfer is identified by the final-beat indication and associated valid-byte metadata. The implementation normalizes the supported full-beat metadata encodings before distinguishing a complete eight-byte final beat from a partial final beat containing one through seven valid bytes.

For each evaluated input, the required 32-byte accelerator result is compared with an independently generated software reference value for the identical formatted byte string. The reference output is computed in Python using a SHAKE256 implementation conforming to FIPS 202 [4], independently of the accelerator RTL. The same formatted input bytes, output length, transfer interpretation, and comparison criterion are applied to NON-SCD, SCD, Naive FBC, and SAFE FBC at 1R, 2R, and 3R. A latency result is retained only when the corresponding accelerator output matches the independently computed reference value. This verification is limited to the evaluated XMSS-oriented SHAKE256 characterization cases and does not constitute exhaustive validation, formal equivalence, or NIST CAVP/CMVP certification of the FPGA implementation.

5.4. Zynq-7000 Platform and System Integration

The SHAKE256 accelerator is integrated in the programmable logic (PL) of the PYNQ-Z2 platform, whose Zynq-7020 system-on-chip combines an Arm processing system (PS) and PL on the same device [7, 8]. The platform provides the system-integration context for accelerator configuration, data movement, and potential future XMSS hardware/software partitioning. In the present evaluation, however, complete XMSS computation on the PS is neither executed nor measured; the characterized object is the SHAKE256 PL accelerator and its accelerator-local transaction boundary.

The implemented integration organization is summarized in Fig. 8. Accelerator configuration is conveyed through the memory-mapped control path, while bulk input and output data are transferred through AXI DMA and the AXI4-Stream datapath [24]. Within the PL, stream buffering and data-width conversion connect the data path to the custom SHAKE256 accelerator, which contains the SHAKE controller/final-block logic and the Keccak- f [1600] datapath. Implementation-specific port wiring and detailed clock/reset distribution are abstracted because they do not change the measurement boundaries defined in the following subsection.

5.5. Latency Taxonomy and Measurement Events

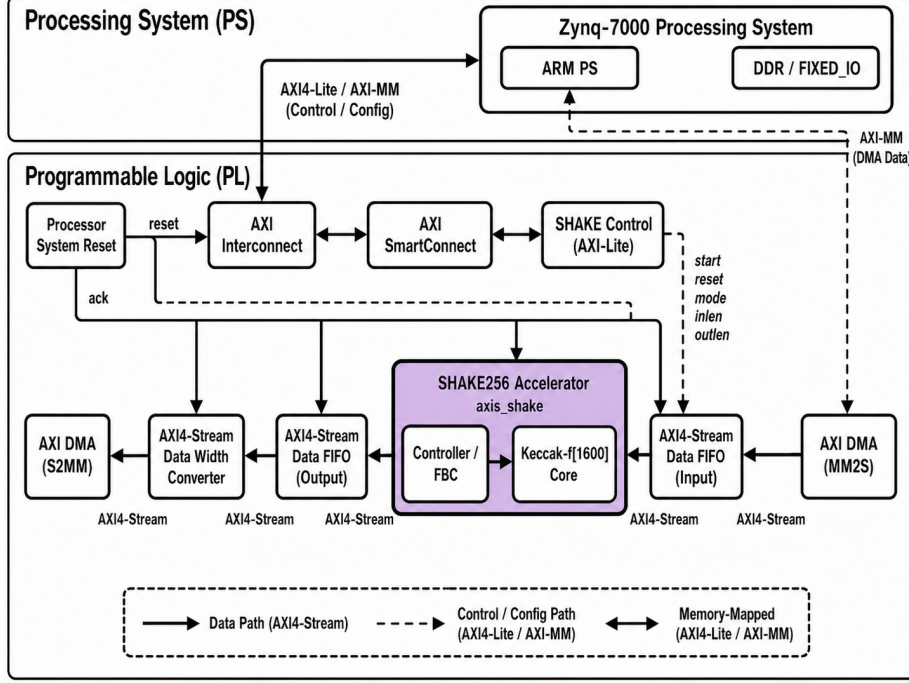
Because latency can be observed at several boundaries of the implemented system, we use a fixed hierarchy that separates Keccak execution, accelerator-internal behavior, AXI4-Stream-visible transfers, and higher-level system activity. At the lowest level, $L0$ denotes the interval from a Keccak permutation request to its completion. At the accelerator level, $L1$ extends from accelerator start to the first internally generated valid output. $L1^*$ -first extends from accelerator start to the first accepted AXI4-Stream output beat, whereas $L1^*$ -final extends from the same accelerator start event to the final accepted AXI4-Stream output beat. For stream-visible measurements, $L2$ -first extends from the first accepted AXI4-Stream input beat to the first accepted AXI4-Stream output beat, and $L2$ -final extends from the same first accepted input beat to the final accepted output beat.

An AXI4-Stream transfer is counted as accepted only on a cycle in which TVALID and TREADY are asserted simultaneously [23]. These accelerator-local observation boundaries and their corresponding measurement events are summarized conceptually in Fig. 9. The definitions keep permutation latency distinct from controller and interface activity and provide the principal observation boundaries used in the controlled architectural comparisons.

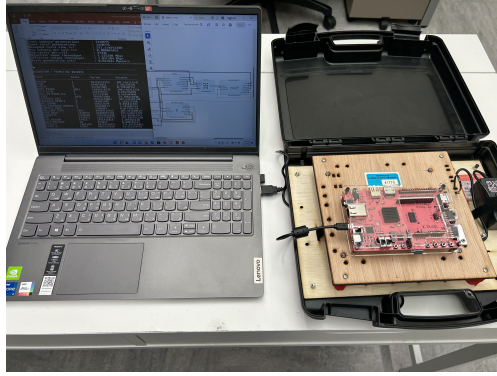
For completeness, the measurement hierarchy also defines higher system-level observation boundaries beyond the accelerator-local intervals shown in Fig. 9. $L3$ denotes a PL-subsystem request-to-completion interval, $L4$ spans DMA submission or activation to completion, $L5$ denotes PS orchestration, and $L6$ extends from software-visible accelerator-call entry to return. These higher boundaries are defined only to separate accelerator-local measurements from progressively broader system and software scopes; they are not instrumented or quantified in the present architectural comparison. The latency levels are therefore treated as distinct observation scopes rather than additive components. In particular, DMA, PS, or software overhead is not inferred by subtracting one latency level from another.

5.6. Internal Component Timing

To localize architectural latency changes beyond the observation boundaries defined above, the evaluation also records directly instrumented timing intervals for SHAKE256 absorb/finalization-control and squeeze processing. Keccak execution is tracked independently through the $L0$ request-to-completion interval. AXI4-Stream input and output spans are also recorded: the input span is the cycle-index difference between the first and final accepted input beats, and the output span is defined analogously between the first and final accepted output beats. These quantities are measured directly rather than derived by subtracting latency levels. The same component definitions are applied to all four controller/finalization organizations at each fixed unrolling depth, allowing an observed $L2$ -final difference to be localized to the accelerator component, in which the timing change occurs.



(a)



(b)

Figure 8: System-integration views of the PYNQ-Z2 SHAKE256 acceleration platform. (a) Vivado block design showing the Zynq-7000 processing system, memory-mapped control, AXI DMA, AXI4-Stream buffering and data-width conversion, and the custom SHAKE256 accelerator. (b) Laboratory PYNQ-Z2 setup used for integration and evaluation. The present study evaluates the PL-side SHAKE256 accelerator and its local transaction boundary; complete XMSS processing on the PS is not measured. Panel (a) was refined with generative-AI assistance using the author’s Vivado block design and specified architecture as the technical basis; the resulting content was reviewed and verified by the author. Panel (b) is an original photograph and was not AI-generated.

5.7. FPGA Implementation Conditions and PPA Methodology

For the implementation-level comparison, all 12 configurations are evaluated under a common FPGA tool flow and top-level harness. Each design targets the Xilinx Zynq-7000 xc7z020c1g400-1 device and is synthesized and implemented using Vivado 2019.1 with a common 20-ns clock-period constraint (50-MHz target). The same common-condition full-top harness, target device, synthesis and implementation settings, and timing constraint are retained across the four architecture families and three unrolling depths. Consequently, results from earlier Artix-7 experiments, deployment or debug builds, ILA-instrumented systems, different tool versions, or different timing constraints are not mixed with the common-condition PPA dataset.

The common-condition comparison reports LUT, flip-flop (FF), slice, block-RAM (BRAM), and DSP utilization. Post-implementation timing reports are used to verify setup closure under the common 20-ns clock constraint and the absence of hold violations. Timing closure under this imposed constraint is treated as a feasibility condition rather than as an estimate of maximum operating frequency; an accepted routed $F_{\max}$ has not been established for the 12-configuration campaign. Accordingly, the common-condition comparison does not infer $F_{\max}$ from slack values. Separate routed deployment implementations used later for contextual prior-work comparison are reported as a distinct implementation scope and are not mixed with the common-condition resource dataset.

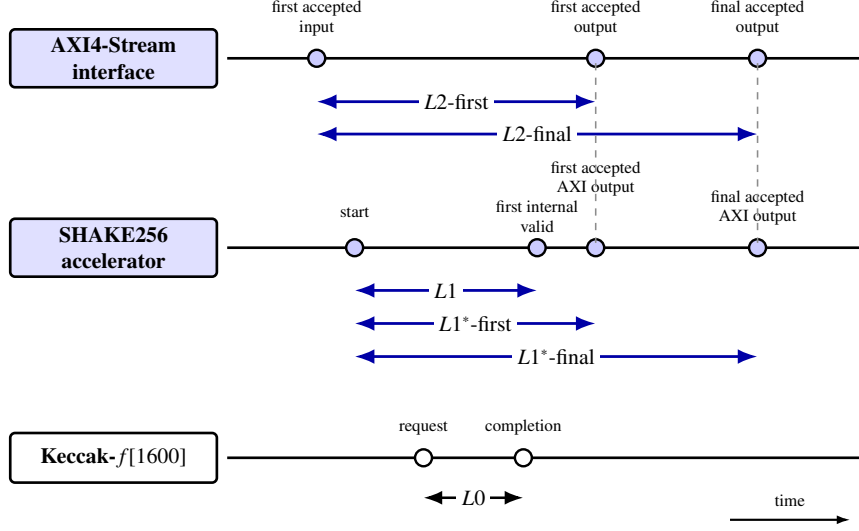


Figure 9: Accelerator-local latency boundaries used in the controlled evaluation. $L0$ is one Keccak- $f[1600]$ request-to-completion interval. $L1$ spans accelerator start to the first internal valid output. $L1^*$ -first and $L1^*$ -final span accelerator start to the first and final accepted AXI4-Stream output beats, respectively. $L2$ -first and $L2$ -final span the first accepted AXI4-Stream input beat to the first and final accepted output beats, respectively. Event positions are schematic and not to scale. System-level boundaries $L3$ – $L6$ are defined separately and are not quantified in this architectural comparison.

5.8. Analytical XMSS Workload Projection Method

To estimate the workload-level significance of the measured primitive latencies, the accelerator results are combined with analytical workload models for XMSS-SHAKE256_10_256. At each unrolling depth, the measured $L2$ -final latency of each relevant SHAKE256 primitive is weighted by its invocation frequency in the corresponding workload model, and the resulting primitive costs are accumulated. To isolate the contribution of SAFE FBC, the projections compare SCD and SAFE FBC at the same unrolling depth, consistent with the controlled comparison defined in Section 5.1. The resulting quantities are serial primitive- $L2$ projections under the stated workload assumptions. They exclude execution costs outside the characterized $L2$ boundary and therefore must not be interpreted as measured end-to-end XMSS latencies or speedsups.

For the reference full-tree key-generation model, the relevant primitive set is $\mathcal{P} = \{F, PRF, H, PRF_{\text{keygen}}\}$. For the selected parameters, the primitive invocation counts follow from

$$\begin{aligned} N_F &= 2^h \text{len} (w - 1), & N_{PRF_{\text{keygen}}} &= 2^h \text{len}, \\ N_H &= 2^h (\text{len} - 1) + (2^h - 1), & N_{PRF} &= 2N_F + 3N_H. \end{aligned}$$

Here, the H count comprises WOTS+ public-key compression and Merkle-tree node hashing. With $h = 10$, $\text{len} = 67$, and $w = 16$, these expressions give $N_F = 1,029,120$, $N_{PRF} = 2,264,061$, $N_H = 68,607$, and $N_{PRF_{\text{keygen}}} = 68,608$ for the selected profile [1, 2]. Let $C_{p,a,R}$ denote the measured $L2$ -final latency of primitive p for architecture a at unrolling depth R . The corresponding summed primitive-level key-generation cost is

$$C_{KG}(a, R) = \sum_{p \in \mathcal{P}} N_p C_{p,a,R}. \quad (3)$$

Using (3), the projected SAFE-FBC saving at a fixed unrolling depth is

$$\Delta C_{KG}(R) = C_{KG}(\text{SCD}, R) - C_{KG}(\text{SAFE FBC}, R), \quad (4)$$

and the corresponding relative reduction is

$$\eta_{KG}(R) = \frac{\Delta C_{KG}(R)}{C_{KG}(\text{SCD}, R)} \times 100\%. \quad (5)$$

Equations (4) and (5) quantify the projected absolute and relative reductions in summed primitive- $L2$ cost under the stated model rather than the complete latency of an implemented XMSS key-generation procedure.

For the WOTS-dominant signing model, the selected profile uses $w = 16$, with $\text{len}_1 = 64$ message digits and $\text{len}_2 = 3$ checksum digits [1, 2, 5]. RFC 8391 maps the 32-byte WOTS+ message digest to 64 base-16 digits and appends the base-16 representation of the checksum before using the resulting 67 digits to determine the individual WOTS+ chain lengths [1]. Let $X_i \in \{0, \dots, 15\}$ denote the i th message digit and define

$$A = \sum_{i=1}^{64} X_i. \quad (6)$$

Using (6), the checksum value before base-16 encoding is

$$C = \sum_{i=1}^{64} (15 - X_i) = 960 - A. \quad (7)$$

Using (7), let $d_{16}(C)$ denote the sum of the three base-16 checksum digits, including a leading zero when required. The total number of WOTS+ chain iterations used in signature generation is then

$$S = A + d_{16}(960 - A). \quad (8)$$

A WOTS+ instance for the selected profile comprises $\text{len} = 67$ independent chains. For $w = 16$, each complete chain spans positions 0 through $w - 1 = 15$, corresponding to at most 15 transitions between successive chain values. During signing, chain i traverses only the number of transitions specified by its corresponding base- w message or checksum digit. Each

transition invokes two ordinary PRF evaluations, one to derive the keyed-hash key and one to derive the bitmask, followed by one F evaluation on the masked chain value [1, 5]. Fig. 10 illustrates the dependencies represented by (8). The arrangement represents the 67 independent chains conceptually and does not imply concurrent execution by the evaluated accelerator.

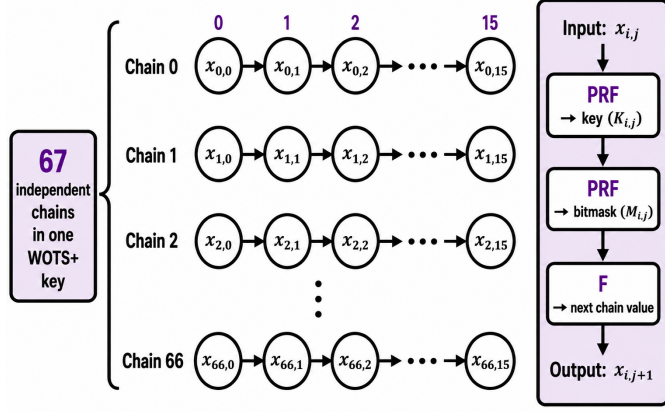


Figure 10: WOTS+ chain organization for XMSS-SHAKE256_10_256. The instance contains 67 independent chains, with sequential dependence within each chain. Each transition performs two ordinary PRF evaluations, for the key and bitmask, followed by one F evaluation.

Assuming a uniformly distributed 256-bit WOTS+ message digest, the 64 variables X_i are independent and uniformly distributed over $\{0, \dots, 15\}$. Therefore, using (8), the expected total number of chain iterations is

$$E[S] = \sum_{a=0}^{960} \Pr(A = a) [a + d_{16}(960 - a)] \approx 499.077. \quad (9)$$

Using the expected iteration count in (9), and because each WOTS+ chain iteration performs two ordinary PRF evaluations and one F evaluation, the expected primitive- $L2$ cost of the modeled chain work is

$$C_{\text{WOTS}}(a, R) = E[S] (2C_{PRF,a,R} + C_{F,a,R}). \quad (10)$$

The WOTS-dominant signing model extends (10) by including the single ordinary PRF evaluation associated with generation of the XMSS message-randomization value [1]. Accordingly,

$$C_{\text{WOTS-SIG}}(a, R) = C_{\text{WOTS}}(a, R) + C_{PRF,a,R}. \quad (11)$$

Using (11), the projected SAFE-FBC saving at a fixed unrolling depth is

$$\Delta C_{\text{WOTS-SIG}}(R) = C_{\text{WOTS-SIG}}(\text{SCD}, R) - C_{\text{WOTS-SIG}}(\text{SAFE FBC}, R). \quad (12)$$

and the corresponding relative reduction is

$$\eta_{\text{WOTS-SIG}}(R) = \frac{\Delta C_{\text{WOTS-SIG}}(R)}{C_{\text{WOTS-SIG}}(\text{SCD}, R)} \times 100\%. \quad (13)$$

Equations (12) and (13) therefore quantify the absolute and relative SAFE-FBC reductions in the modeled WOTS-dominant primitive- $L2$ signing cost at a fixed unrolling depth.

The scope of this WOTS-dominant signing model is intentionally restricted. Authentication-path computation is excluded by assuming that the required path is already available or stored; RFC 8391 permits implementations to trade computation against storage when obtaining these nodes [1]. Generation of the WOTS+ secret elements is also outside the modeled interval, including the corresponding PRF_{keygen} work under the evaluated SP 800-208 semantics [2]. Finally, although XMSS signing requires H_{msg} , its contribution is not included in (11) because the characterized H_{msg} latency depends on the formatted message length [1, 2]. A projection including this term must therefore identify the specific evaluated H_{msg} case used. Accordingly, the model covers the expected WOTS+ chain work and the signing-randomization PRF defined above, and its resulting reductions are analytical primitive- $L2$ projections rather than measured end-to-end XMSS signing speedups.

6. Experimental Results

6.1. Per-Primitive Transaction Latency

Table 2 reports the measured $L2$ -final latency of the nine XMSS-oriented SHAKE256 characterization cases for all 12 evaluated configurations. At every fixed Keccak unrolling depth, SAFE FBC is the only evaluated controller/finalization organization that reduces the nine-case aggregate $L2$ -final latency. Relative to the SCD controlled baseline, the aggregate decreases from 464 to 451 cycles at 1R, from 341 to 328 cycles at 2R, and from 305 to 292 cycles at 3R. SAFE FBC therefore removes exactly 13 aggregate cycles at each fixed depth, corresponding to reductions of 2.80%, 3.81%, and 4.26% for 1R, 2R, and 3R, respectively. It produces lower $L2$ -final latency in five of the nine cases, leaves four unchanged, and introduces no measured latency regression. Because the absolute saving remains 13 cycles while deeper unrolling lowers the corresponding SCD aggregate latency, the SAFE-FBC contribution represents an increasing fraction of the measured transaction latency from 1R to 3R. The component-level measurements in Section 6.2 subsequently localize these saved cycles.

Relative to SCD at the same unrolling depth, the SAFE-FBC per-case saving vector is $[5, 5, 1, 1, 1, 0, 0, 0, 0]$ cycles for $F96$, $PRF96$, $H128$, $PRF_{\text{keygen}}128$, $H_{\text{msg}}128$, $H_{\text{msg}}129$, $H_{\text{msg}}135$, $H_{\text{msg}}136$, and $H_{\text{msg}}137$, respectively. The largest measured reduction occurs for $F96$ and $PRF96$, each of which saves five cycles at every unrolling depth. Their $L2$ -final latency decreases from 46 to 41 cycles at 1R, from 34 to 29 cycles at 2R, and from 30 to 25 cycles at 3R, corresponding to fixed-depth reductions of 10.87%, 14.71%, and 16.67%, respectively. The $H128$, $PRF_{\text{keygen}}128$, and $H_{\text{msg}}128$ cases each save one cycle, whereas $H_{\text{msg}}129$, $H_{\text{msg}}135$, $H_{\text{msg}}136$, and $H_{\text{msg}}137$ show no measured latency change.

This selectivity is consistent with the SAFE-FBC mechanism described in Section 4.4: direct-path eligibility produces a cycle reduction only when the bypass also eliminates sequential

Table 2: Measured $L2$ -final latency for the nine XMSS-oriented SHAKE256 characterization cases

R	Architecture	F_{96}	PRF_{96}	H_{128}	PRF_{keygen}^{128}	H_{msg}^{128}	H_{msg}^{129}	H_{msg}^{135}	H_{msg}^{136}	H_{msg}^{137}	Total
1R	NON-SCD	46	46	46	46	46	46	46	71	71	464
1R	SCD	46	46	46	46	46	46	46	71	71	464
1R	Naive FBC	46	46	46	46	46	46	46	71	71	464
1R	SAFE FBC	41	41	45	45	45	46	46	71	71	451
2R	NON-SCD	34	34	34	34	34	34	34	52	51	341
2R	SCD	34	34	34	34	34	34	34	52	51	341
2R	Naive FBC	34	34	34	34	34	34	34	52	51	341
2R	SAFE FBC	29	29	33	33	33	34	34	52	51	328
3R	NON-SCD	30	30	30	30	30	30	30	48	47	305
3R	SCD	30	30	30	30	30	30	30	48	47	305
3R	Naive FBC	30	30	30	30	30	30	30	48	47	305
3R	SAFE FBC	25	25	29	29	29	30	30	48	47	292

All values are in clock cycles. “Total” is the sum of the nine measured $L2$ -final latencies, not the latency of a single SHAKE256 invocation. Bold values indicate SAFE FBC results lower than the corresponding same-depth SCD baseline.

finalization work that would otherwise remain on the conventional path. A zero measured saving therefore does not, by itself, imply SAFE-FBC ineligibility; fast-path eligibility and elimination of sequential finalization work are distinct conditions. The five-cycle reductions for F and ordinary PRF are particularly relevant to the target XMSS profile because WOTS+ repeatedly invokes these primitives during chain processing [1]. Their cumulative effect is evaluated analytically in Section 6.4.

The controlled ablation sequence separates the measured effects of controller restructuring and final-block handling. NON-SCD and SCD are cycle-identical for every evaluated case at each unrolling depth, showing that SCD does not itself reduce transaction latency for this characterization workload; its role is to provide the controlled restructured baseline. Naive FBC is likewise cycle-identical to SCD across all nine cases, showing that the restrictive direct-final-block condition does not expose a measurable latency saving for the evaluated input shapes. SAFE FBC produces a different outcome: at every depth it reduces five cases and leaves the remaining four unchanged. The ablation therefore shows that controller restructuring alone, or the presence of a restrictive direct-final-block path, is insufficient to obtain the measured benefit. For this characterization workload, the broader guarded, SHAKE256-rate-aware SAFE-FBC condition exposes useful bypass opportunities without introducing a measured latency regression in the remaining cases.

Cross-depth comparisons quantify the latency reached when deeper Keccak unrolling and SAFE FBC are applied together. The lowest aggregate $L2$ -final latency in the evaluated 12-configuration matrix is 292 cycles for 3R SAFE FBC. Relative to the 2R SCD baseline, this configuration reduces the nine-case aggregate from 341 to 292 cycles, corresponding to a 14.37% reduction. Relative to the 1R SCD baseline, the reduction is from 464 to 292 cycles, or 37.07%. These cross-depth values combine the effect of changing the Keccak unrolling depth with the SAFE-FBC finalization optimization and must therefore not be attributed to SAFE FBC alone. The isolated SAFE-FBC contribution is the fixed-depth SCD-to-SAFE comparison reported above; the cross-depth results instead characterize the aggregate latency obtained when the permutation-side and finalization-control optimization dimensions are used together.

6.2. Accelerator-Level and Component Timing

Using the observation boundaries defined in Section 5.5 and the directly instrumented component intervals defined in Section 5.6, Table 3 consolidates the measured timing quantities for SCD and SAFE FBC at each fixed Keccak unrolling depth. SAFE FBC reduces the aggregate $L1$, $L1^*$ -first, $L1^*$ -final, $L2$ -first, and $L2$ -final measurements by 13 cycles relative to SCD at 1R, 2R, and 3R. The same 13-cycle difference is observed in the aggregate absorb/finalization-control interval. In contrast, the AXI4-Stream input and output spans, aggregate squeeze interval, and one-permutation $L0$ latency are unchanged between SCD and SAFE FBC at each corresponding depth. The measured SAFE-FBC latency reduction is therefore localized to the absorb/finalization-control path targeted by the architecture rather than to stream-transfer timing, output squeezing, or Keccak permutation execution.

The timing changes across unrolling depths distinguish the effect of Keccak round unrolling from the fixed-depth effect of SAFE FBC. The $L0$ latency of one Keccak- $f[1600]$ permutation decreases from 24 cycles at 1R to 12 cycles at 2R and 8 cycles at 3R. The aggregate squeeze interval likewise decreases from 261 cycles at 1R to 153 cycles at 2R and 117 cycles at 3R. At any fixed unrolling depth, however, both quantities are identical for SCD and SAFE FBC. The SCD-to-SAFE difference instead remains confined to absorb/finalization control, which decreases from 221 to 208 aggregate cycles at 1R and from 206 to 193 aggregate cycles at both 2R and 3R. These distinct timing signatures separate the two optimization dimensions: round unrolling reduces permutation-side latency, whereas SAFE FBC removes residual absorb/finalization-control cycles outside the permutation. SAFE FBC therefore neither shortens nor eliminates a Keccak permutation.

The aggregate AXI4-Stream input and output spans remain 132 and 27 cycles, respectively, for every SCD and SAFE-FBC configuration. For the nine characterized inputs and their 32-byte outputs, these values correspond to the observed uninterrupted 64-bit stream transfers. The equality of $L1$ and $L1^*$ -first in the measured traces likewise shows that the first internally valid output beat was accepted without an additional wait cycle. To-

Table 3: Measured accelerator-level and internal timing across SCD and SAFE FBC

Timing Quantity	1R SCD	1R SAFE	2R SCD	2R SAFE	3R SCD	3R SAFE
$L0$: permutation request→completion	24	24	12	12	8	8
$L1$: start→first internal valid	455	442	332	319	296	283
$L1^*$ -first: start→first accepted AXI output	455	442	332	319	296	283
$L1^*$ -final: start→final accepted AXI output	482	469	359	346	323	310
$L2$ -first: first accepted input→first accepted output	437	424	314	301	278	265
$L2$ -final: first accepted input→final accepted output	464	451	341	328	305	292
AXI4-Stream input span	132	132	132	132	132	132
AXI4-Stream output span	27	27	27	27	27	27
Absorb/finalization control	221	208	206	193	206	193
Squeeze	261	261	153	153	117	117

$L0$ denotes the latency, in clock cycles, of one Keccak- $f[1600]$ permutation; all other entries are aggregate cycle counts over the nine evaluated XMSS-oriented SHAKE256 cases. AXI4-Stream transfers are accepted only when VALID and READY are asserted simultaneously. Aggregate input and output spans are 132 and 27 cycles, respectively, with no observed stalls. NON-SCD and Naive FBC are omitted because their measured timing is cycle-identical to SCD at the corresponding unrolling depth. DMA, FIFO, SmartConnect, AXI-MM, PS, and software latencies lie outside the reported accelerator-local boundary; they were not separately measured or inferred by subtraction.

gether with the unchanged squeeze interval and $L0$ latency, these observations further isolate the 13-cycle fixed-depth SAFE-FBC saving to absorb/finalization control.

The reported measurements deliberately stop at the accelerator-local boundary. DMA, FIFO, SmartConnect, AXI-MM, PS orchestration, and software-call latencies correspond to the higher $L3$ – $L6$ scopes defined in Section 5.5; they were not separately measured in the present evaluation and are therefore neither included in the reported values nor estimated from lower-level timing differences.

6.3. FPGA Resource Utilization and Timing

Table 4 reports the common-condition FPGA resource utilization for all 12 evaluated configurations. Relative to the matching SCD controlled baseline, SAFE FBC increases the LUT count by 106.66%, 75.86%, and 59.77% at 1R, 2R, and 3R, respectively, with corresponding slice increases of 114.31%, 78.12%, and 61.34%. These relative overheads are distinct from absolute device utilization. The Zynq-7020 provides 53,200 LUTs [25]; the 1R, 2R, and 3R SAFE-FBC configurations occupy 18.19%, 32.45%, and 30.72% of the available LUTs, corresponding to increases of 9.39, 14.00, and 11.49 percentage points over the matching SCD configurations. In contrast, the corresponding FF-count overheads are only 0.13%, 0.38%, and 0.38%. The common-condition results therefore expose a latency–area tradeoff: SAFE FBC requires substantially more LUT and slice resources while adding very little sequential storage.

The architecture sequence also provides an area-side ablation of the design evolution. NON-SCD and SCD remain close in resource utilization at every unrolling depth, indicating that the SCD controller restructuring changes the LUT and slice footprint only marginally under the common implementation condition. Relative to SCD, Naive FBC increases LUT count by 13.18%, 6.08%, and 5.19% at 1R, 2R, and 3R, respectively, with corresponding slice increases of 14.39%, 6.42%, and 5.10%. As shown in Section 6.1, this additional hardware produces no measured $L2$ -final reduction for the evaluated XMSS-oriented SHAKE256 characterization cases. SAFE FBC incurs a larger LUT and slice cost, but unlike the restrictive SAFE FBC variant, it also produces the measured fixed-depth latency reduction. For

this characterization workload, the additional direct-final-block hardware therefore yields a measured latency benefit only when the eligibility condition exposes rate-fit cases that eliminate sequential finalization work.

Fig. 11 pairs the controlled latency and resource measurements for SCD and SAFE FBC at each Keccak unrolling depth. Aggregate $L2$ -final decreases from 464 to 451 cycles at 1R, from 341 to 328 cycles at 2R, and from 305 to 292 cycles at 3R, while the corresponding LUT counts increase from 4,682 to 9,676, from 9,818 to 17,266, and from 10,230 to 16,344. The figure therefore represents a fixed-depth latency–area tradeoff rather than an overall efficiency ranking. At each depth, SAFE FBC provides the same 13-cycle aggregate reduction at the cost of additional combinational area. Cross-depth latency differences additionally include the effect of Keccak round unrolling and are therefore not attributed to SAFE FBC alone.

All 12 configurations meet the common 20-ns implementation constraint. Post-implementation timing reports confirm setup closure at the 50-MHz target and no hold violation for the common-condition flow. Thus, the additional combinational logic introduced by SAFE FBC does not prevent the evaluated 1R, 2R, or 3R configurations from meeting the same imposed timing condition as their corresponding baseline-family configurations. This result establishes timing feasibility at the common comparison point; it does not imply equal maximum operating frequency among the architectures. No routed $F_{\max}$ is inferred from the timing slack of the 12-configuration campaign. The separate routed clock points used later for contextual prior-work comparison are therefore treated as a distinct implementation scope.

6.4. Projected XMSS Workload Impact

The full-tree key-generation projection applies the measured primitive $L2$ -final latencies to the invocation counts and weighting model defined in Section 5.8. At each fixed Keccak unrolling depth, replacing SCD with SAFE FBC reduces the projected summed primitive- $L2$ cost by 16,603,120 cycles. Of this saving, 99.17% arises from the repeatedly invoked F and ordinary PRF primitives, reflecting both their five-cycle per-call reduction and their dominant frequency in the modeled workload. Relative

Table 4: Common-condition FPGA resource utilization for the 12 evaluated configurations

R	Architecture	LUT	LUT (%)	FF	Slices	BRAM	DSP
1R	NON-SCD	4,702	8.84	3,169	1,276	0	0
1R	SCD	4,682	8.80	3,174	1,251	0	0
1R	Naive FBC	5,299	9.96	3,191	1,431	0	0
1R	SAFE FBC	9,676	18.19	3,178	2,681	0	0
2R	NON-SCD	9,831	18.48	3,168	2,607	0	0
2R	SCD	9,818	18.45	3,173	2,587	0	0
2R	Naive FBC	10,415	19.58	3,174	2,753	0	0
2R	SAFE FBC	17,266	32.45	3,185	4,608	0	0
3R	NON-SCD	10,245	19.26	3,208	2,724	0	0
3R	SCD	10,230	19.23	3,174	2,708	0	0
3R	Naive FBC	10,761	20.23	3,196	2,846	0	0
3R	SAFE FBC	16,344	30.72	3,186	4,369	0	0

All values are obtained using a common top-level comparison harness targeting the Zynq-7020 (xc7z020c1g400-1) in Vivado 2019.1 with a 20-ns clock constraint; they do not represent the complete integrated PYNQ-Z2 PL deployment. LUT (%) is relative to the 53,200 LUTs available on the device [25]. Identical implementation conditions are used for all 12 configurations. BRAM and DSP utilization are zero for all configurations.

■ SCD ■ SAFE FBC

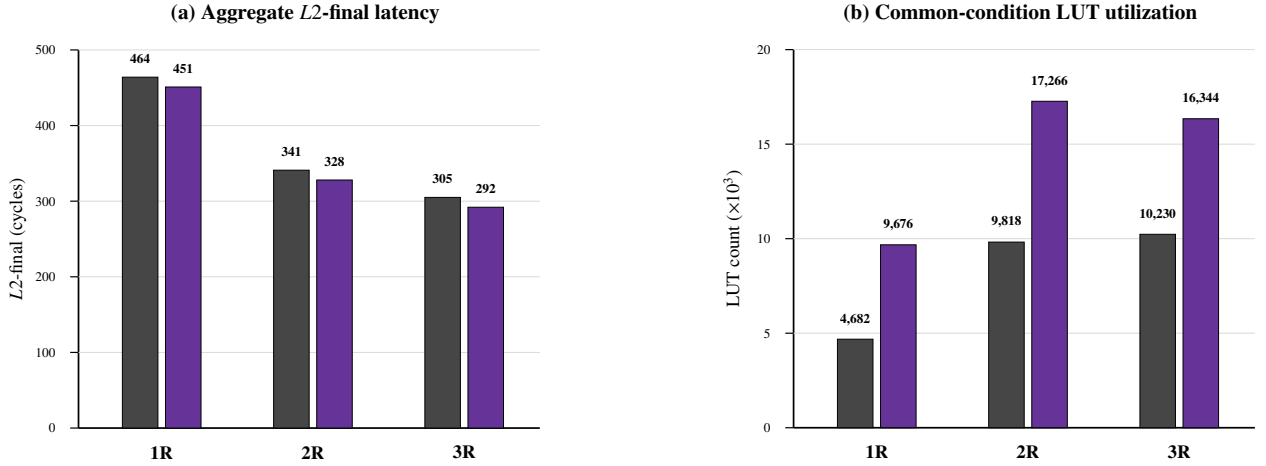


Figure 11: Fixed-depth latency–area comparison of SCD and SAFE FBC under the common implementation condition. (a) Aggregate measured $L2$ -final latency across the nine XMSS-oriented SHAKE256 characterization cases. (b) LUT utilization for the corresponding configurations. At each unrolling depth, SAFE FBC reduces aggregate $L2$ -final latency by 13 cycles while increasing LUT utilization. Cross-depth latency differences also include the effect of Keccak round unrolling and are not attributed to SAFE FBC alone. Complete results are reported in Tables 2 and 4.

to the corresponding SCD primitive- $L2$ cost, the projected reductions are 10.52%, 14.24%, and 16.13% at 1R, 2R, and 3R, respectively. The absolute projected saving is unchanged across the three depths because the measured SAFE-FBC per-call saving vector is identical at each depth; its relative contribution increases as deeper unrolling lowers the corresponding SCD primitive- $L2$ baseline. These values are analytical accumulations of measured primitive latencies under the stated full-tree model and are not measured end-to-end XMSS key-generation speedups.

The WOTS-dominant signing projection uses the measured F and ordinary PRF latencies with the analytical model defined in Section 5.8. Under the stated uniform-message-digest assumption, the expected total number of WOTS+ chain iterations is $E[S] \approx 499.077$. Each iteration performs two ordinary PRF evaluations and one F evaluation [1]. Since all three calls receive the same five-cycle SAFE-FBC reduction at a fixed unrolling depth, the expected WOTS+ chain contribution decreases by approximately 7,486 primitive- $L2$ cycles. Including

the single additional ordinary PRF used to generate the XMSS signing-randomization value increases the projected saving to approximately 7,491 cycles [1]. Relative to the corresponding SCD primitive- $L2$ cost, the resulting reductions are 10.87%, 14.71%, and 16.67% at 1R, 2R, and 3R, respectively. As in the full-tree model, the absolute saving remains unchanged across unrolling depths while its relative contribution grows as the SCD primitive latency decreases. These values follow the restricted WOTS-dominant signing model and its assumptions, including exclusion of an unspecified H_{msg} characterization case; they are analytical primitive- $L2$ projections rather than measured end-to-end XMSS signing speedups.

Table 5 summarizes the two projection models and their resulting fixed-depth reductions.

Fig. 12 visualizes the relative reductions summarized in Table 5. Each plotted point represents a fixed-depth SCD-to-SAFE-FBC comparison, so the Keccak unrolling depth is held constant within each comparison and the reported percentage reflects the incremental reduction obtained by replacing SCD with SAFE

Table 5: Analytical XMSS primitive- $L2$ workload projections

Projection Model	Projected Saved Cycles	1R (%)	2R (%)	3R (%)
Full-tree key generation	16,603,120	10.52	14.24	16.13
WOTS-dominant signing	$\approx 7,491$	10.87	14.71	16.67

Note: “Projected Saved Cycles” is the analytically projected reduction in summed primitive- $L2$ cycles relative to SCD at the same unrolling depth. The absolute saved-cycle total is identical at 1R, 2R, and 3R because the measured SAFE-FBC per-call saving vector is unchanged across depths; only the percentage reduction varies with the corresponding SCD baseline. The WOTS-dominant signing projection includes the expected WOTS+ chain contribution and one signing-randomization ordinary PRF , following the assumptions in Section 5.8. All values are analytical workload projections, not measured end-to-end XMSS speedups.

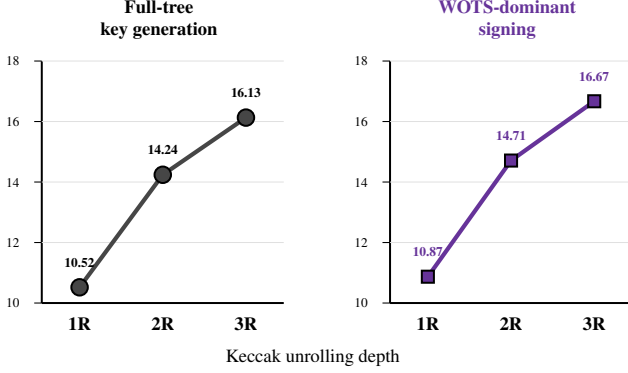


Figure 12: Analytically projected primitive- $L2$ reductions for the two XMSS workload models across the evaluated Keccak unrolling depths. Each point represents a fixed-depth SCD-to-SAFE-FBC comparison. The reported values are analytical workload projections, not measured end-to-end XMSS speedups; absolute projected saved-cycle totals are reported in Table 5.

FBC at that depth. The two projection models exhibit closely matched trends because both are strongly weighted toward repeated F and ordinary PRF evaluations, for which SAFE FBC provides the same five-cycle fixed-depth saving. Within each workload model, the absolute projected SAFE-FBC saved-cycle total remains unchanged across 1R, 2R, and 3R, whereas the corresponding SCD primitive- $L2$ workload baseline decreases with deeper Keccak unrolling. Consequently, the relative SAFE-FBC reduction increases with unrolling depth. These percentages are therefore not cross-depth combined SAFE-FBC-plus-unrolling speedups. The figure reports relative reductions only; the substantially different absolute projected saved-cycle totals of the two workload models remain in Table 5 and are not compared graphically.

The absolute projected savings in Table 5 should not be compared as equivalent workloads because the full-tree key-generation and WOTS-dominant signing models represent substantially different amounts of primitive computation. Their relative reductions, however, are close at each unrolling depth because both models are strongly weighted toward repeated F and ordinary PRF evaluations, the two characterization cases for which SAFE FBC provides the largest measured five-cycle reduction. In the full-tree model, these two primitives account for 99.17% of the projected saved primitive- $L2$ cycles. The slightly lower full-tree reductions of 10.52%, 14.24%, and 16.13%, compared with 10.87%, 14.71%, and 16.67% for the WOTS-dominant model, are consistent with the additional H and PRF_{keygen} calls in the full-tree workload, for which the measured SAFE-FBC saving is one cycle rather than five. The projections therefore provide workload-level context for the measured accelerator-local latency reductions while remaining

limited to the stated primitive- $L2$ models.

A separate derived metric is used to illustrate how the measured aggregate latency changes translate into an effective rate for the nine-case characterization workload. The formatted inputs of the nine XMSS-oriented SHAKE256 characterization cases contain a total of 1,113 bytes, or 8,904 bits. At the common 50-MHz comparison clock, the derived effective $L2$ workload throughput is defined as

$$T_{L2,eff} = \frac{8,904 f_{clk}}{C_{agg}}, \quad (14)$$

where C_{agg} is the measured aggregate $L2$ -final cycle count across the nine cases. Using (14), the derived values are 0.9595 and 0.9871 Gb/s for 1R SCD and SAFE FBC, 1.3056 and 1.3573 Gb/s for 2R, and 1.4597 and 1.5247 Gb/s for 3R, respectively.

Fig. 13 visualizes these derived workload-throughput values. Deeper Keccak round unrolling raises the effective rate while reducing aggregate transaction latency. At each fixed unrolling depth, SAFE FBC provides an additional improvement by reducing aggregate $L2$ -final by 13 cycles: from 464 to 451 cycles at 1R, from 341 to 328 cycles at 2R, and from 305 to 292 cycles at 3R. As established in Section 6.2, these fixed-depth reductions originate in absorb/finalization control rather than in a change to Keccak permutation latency. Cross-depth differences, in contrast, combine round unrolling with SAFE FBC and are not attributed to SAFE FBC alone. The quantity in (14) is a derived characterization-workload metric and should not be interpreted as measured sustained streaming SHAKE256 throughput.

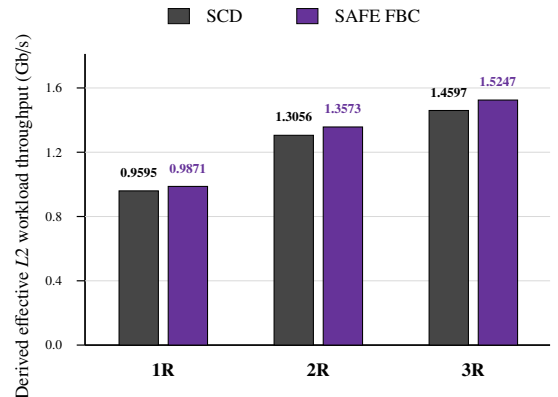


Figure 13: Derived effective $L2$ workload throughput for SCD and SAFE FBC under the common 50-MHz implementation condition across the evaluated Keccak unrolling depths. The metric is computed from the nine XMSS-oriented SHAKE256 characterization cases using (14). The reported values are derived workload-throughput metrics and not measured sustained streaming SHAKE256 throughput.

6.5. Contextual Comparison with Prior FPGA Keccak/SHAKE Architectures

Table 6 places selected routed implementations from this work in the context of three prior FPGA Keccak/SHAKE designs discussed in Section 3. The purpose of this comparison is architectural context rather than a normalized ranking of SAFE FBC against prior permutation-side optimizations. Deshpande *et al.* provide the open-source SHAKE/Keccak RTL from which the NON-SCD reference organization is derived [6]; Sideris *et al.* provide a representative FPGA implementation of established Keccak round unrolling [9]; and Saarinen provides processor-coupled Keccak/SHAKE accelerator context under a different system boundary [12]. These works therefore establish relevant permutation-side and implementation context for the present design, whereas SAFE FBC targets a different latency component: residual absorb/finalization-control work outside the Keccak- $f[1600]$ permutation.

FPGA device, implementation flow, function, interface scope, clocking basis, and resource-reporting boundary differ among the compared studies. Accordingly, the values in Table 6 are not used to claim normalized superiority in area, throughput, or end-to-end SHAKE256 performance. In particular, the permutation-side metric T_{perm} characterizes the Keccak implementation and its clock basis; it does not quantify the controller-side latency reduction introduced by SAFE FBC. The present-work entries in the table are selected routed deployment implementations. Controlled quantitative comparisons among NON-SCD, SCD, Naive FBC, and SAFE FBC remain based on the common-condition 12-configuration campaign reported in Section 6.3.

For the architectural context in Table 6, the permutation-side rate is expressed as

$$T_{\text{perm}} = \frac{1088 f_{\text{clk}}}{C_{\text{perm}}},$$

where 1088 bits is the SHAKE256 rate and C_{perm} is the Keccak- $f[1600]$ permutation cycle count. This metric describes only the rate associated with the permutation organization and clock basis. It does not include absorb/finalization-control latency and therefore is not a direct metric of the SAFE-FBC contribution. Likewise, T_{perm} is not an end-to-end SHAKE256 throughput measurement.

This distinction is central to the comparison. Prior round-unrolled architectures such as that of Sideris *et al.* target permutation-side latency by evaluating multiple Keccak rounds per cycle. SAFE FBC instead leaves Keccak- $f[1600]$ unchanged and targets residual controller work required to construct and deliver an eligible final SHAKE256 rate block before the final absorb permutation. The two approaches therefore operate on different latency components. The comparison with prior FPGA Keccak/SHAKE implementations is included to establish the permutation-side architectural context in which SAFE FBC operates as a complementary optimization, not to claim that SAFE FBC itself improves T_{perm} .

The values in Table 6 illustrate this distinction directly. Sideris *et al.* have the highest listed T_{perm} , 23.071 Gb/s, reflecting their permutation-side implementation and 254.46-MHz reported

clock basis. The selected 3R SAFE-FBC routed implementation has a derived T_{perm} of 7.771 Gb/s. This difference does not constitute a comparison of the SAFE-FBC controller optimization against the Sideris round-unrolling technique, because T_{perm} does not measure the latency component modified by SAFE FBC. No implementation of SAFE FBC within the exact Sideris architecture is evaluated in this work, and no numerical performance result is claimed for such a combined implementation.

The complementarity with round unrolling is established most directly by the fixed-depth comparisons within the present implementation rather than by a cross-paper latency comparison.

Prior work such as Sideris *et al.* demonstrates the permutation-side benefit of evaluating multiple Keccak rounds per cycle [9]. In the present design, the corresponding question is whether SAFE FBC continues to provide an incremental benefit after such permutation-side unrolling has already been applied. The measured 2R and 3R SCD-to-SAFE comparisons answer this affirmatively: with the Keccak datapath organization held fixed at each depth, SAFE FBC reduces the nine-case aggregate $L2$ -final latency from 341 to 328 cycles at 2R and from 305 to 292 cycles at 3R. These fixed-depth reductions demonstrate that residual absorb/finalization-control latency remains after round unrolling and can be reduced independently by SAFE FBC.

The fixed-depth measurements within the present design provide the controlled evidence for the SAFE-FBC contribution. At 1R, SCD and SAFE FBC both retain a 24-cycle Keccak- $f[1600]$ permutation. Their selected routed clock points are 125.00 and 111.11 MHz, respectively, yielding derived T_{perm} values of 5.667 and 5.037 Gb/s. Despite the lower routed T_{perm} of the selected 1R SAFE-FBC implementation, SAFE FBC reduces the measured nine-case aggregate $L2$ -final latency from 464 to 451 cycles relative to 1R SCD. The latency reduction therefore cannot be attributed to a faster permutation. As established in Section 6.2, the complete fixed-depth saving is localized to absorb/finalization control, while the one-permutation Keccak latency, aggregate squeeze interval, and AXI4-Stream transfer spans remain unchanged at fixed depth.

The same separation applies at the other unrolling depths. At a given depth, the SCD-to-SAFE comparison holds the Keccak datapath organization fixed and isolates the incremental controller-side effect of SAFE FBC. Across different depths, however, the comparison also changes the permutation organization. Among the selected SAFE-FBC routed implementations, increasing the unrolling depth from 1R to 2R and 3R reduces the permutation cycle count from 24 to 12 and 8 cycles. With the corresponding routed clock points of 111.11, 74.07, and 57.14 MHz, the derived T_{perm} values are 5.037, 6.716, and 7.771 Gb/s, while the nine-case aggregate $L2$ -final latencies are 451, 328, and 292 cycles. These cross-depth changes combine the effect of Keccak round unrolling with SAFE FBC and are therefore not attributed to SAFE FBC alone.

Accordingly, the two optimization dimensions should be interpreted separately. Round unrolling reduces the cycles required by Keccak- $f[1600]$, whereas SAFE FBC reduces eligible absorb/finalization-control work outside the permutation. Their complementarity is demonstrated within the controlled 1R/2R/3R design matrix by the persistence of the SAFE-FBC

Table 6: Contextual comparison with relevant FPGA Keccak/SHAKE implementations

Work / Config.	Function / Scope	FPGA	R/ cycle	Perm. cycles	LUT	FF	BRAM / Slices	Clock basis (MHz)	T_{perm} (Gb/s)
<i>* This work: selected routed SHAKE256 hierarchies</i>									
1R SCD	SHAKE256 hierarchy	XC7Z020-1	1	24	4,756	3,001	0 BRAM	125.00	5.667
1R SAFE FBC	SHAKE256 hierarchy	XC7Z020-1	1	24	13,034	3,012	0 BRAM	111.11	5.037
2R SAFE FBC	SHAKE256 hierarchy	XC7Z020-1	2	12	16,957	3,142	0 BRAM	74.07	6.716
3R SAFE FBC	SHAKE256 hierarchy	XC7Z020-1	3	8	16,035	3,143	0 BRAM	57.14	7.771
<i>* Relevant prior FPGA Keccak/SHAKE implementations</i>									
Deshpande <i>et al.</i> [6]	SHAKE256 module	XC7A200T-3	1	24	4,383	2,708	0 BRAM	250	11.333
Saarinen [12]	Keccak/SHAKE + control	XC7A100T-2	1	24	+5,582	–	–	100 (constraint)	4.533
Sideris <i>et al.</i> [9]	SHA3-256 core	Artix-7	2	12	–	–	1,184 slices	254.46	23.071

Notes: Present-work area values refer to the SHAKE256 accelerator hierarchy of the selected routed PYNQ-Z2 deployments and therefore differ from the common-condition comparison-harness values in Table 4. The listed clocks are routed timing-closure points, not claimed F_{max} values. T_{perm} is a derived permutation-side metric included only for Keccak implementation context; it is neither a measure of the SAFE-FBC controller-side latency reduction nor sustained SHAKE256 throughput. Values for Deshpande *et al.* and Saarinen are derived from their reported clock and permutation-cycle data, whereas the Sideris *et al.* value is reported directly by the authors. Saarinen reports incremental LUT cost, while Sideris *et al.* report occupied slices. Unreported resource types are shown as “–” and are excluded from normalized area comparisons.

fixed-depth saving after the Keccak datapath has already been unrolled. The cross-depth reduction from 1R SCD to 3R SAFE FBC reported elsewhere in this work consequently reflects both optimization dimensions, whereas the fixed-depth SCD-to-SAFE comparisons quantify the incremental SAFE-FBC contribution.

To make the resource boundary of the routed comparison explicit, Table 7 decomposes the corresponding PYNQ-Z2 deployments into the SHAKE256 accelerator hierarchy and the surrounding PL infrastructure. This decomposition is reported separately from the common-condition PPA campaign because the two sets of results serve different purposes: Table 4 supports controlled comparison among all 12 configurations, whereas Table 7 identifies the resource boundary of the selected integrated routed deployments used for cross-work architectural context.

Across the four selected routed deployments, non-SHAKE PL infrastructure remains within 4,568–4,582 LUTs and uses 6,554 FFs, whereas the SHAKE256 accelerator hierarchy ranges from 4,756 to 16,957 LUTs and from 3,001 to 3,143 FFs. Most of the configuration-dependent resource variation therefore resides within the SHAKE256 accelerator rather than in the surrounding PL infrastructure. The accelerator hierarchy uses no BRAM in any of the selected configurations, while the complete integrated PL uses six BRAM tiles in every build for communication and buffering infrastructure.

Among the SAFE-FBC routed deployments, the SHAKE256 hierarchy uses 13,034, 16,957, and 16,035 LUTs at 1R, 2R, and 3R, respectively. The routed LUT mapping is therefore non-monotonic: the selected 3R implementation uses fewer LUTs than the selected 2R implementation. The available implementation reports establish this outcome but do not isolate a specific synthesis, packing, or placement cause, so no causal interpretation is assigned to the non-monotonic mapping.

The routed deployment results serve a different purpose

from the common-condition 12-configuration evaluation in Section 6.3. Cross-work resource context is therefore based on the axis_shake_0 accelerator hierarchy rather than on the complete integrated-PL totals. Direct cross-device area percentages are not reported because FPGA device, implementation flow, timing condition, interface scope, and resource-reporting boundary differ among the compared studies. Accordingly, Table 6 should be interpreted as architectural and permutation-side context rather than evidence of a normalized area, throughput, or overall- performance ranking.

7. Conclusion

This work investigated residual transaction latency in an AXI4-Stream SHAKE256 accelerator while treating Keccak round unrolling and controller-side finalization as separate optimization dimensions. Starting from the inherited NON-SCD organization of the adopted open-source SHAKE RTL, the design progression introduced SCD as a restructured controlled baseline, Naive FBC as a restrictive direct-final-block ablation, and SAFE FBC as the proposed guarded final-block mechanism. The controlled comparison shows that neither SCD restructuring alone nor the Naive FBC path reduces measured transaction latency for the evaluated characterization cases. SAFE FBC instead applies SHAKE256-rate-aware eligibility and direct construction of eligible final rate-block content to bypass residual absorb/finalization-control work without modifying the Keccak- $f[1600]$ permutation, making it a controller-side complement to permutation-level round unrolling.

Across the nine XMSS-oriented SHAKE256 characterization cases, SAFE FBC reduces aggregate $L2$ -final latency by exactly 13 cycles relative to SCD at every fixed unrolling depth: from 464 to 451 cycles at 1R, from 341 to 328 cycles at 2R, and from 305 to 292 cycles at 3R, corresponding to reductions of 2.80%,

Table 7: Routed deployment-layer resource decomposition of selected integrated SHAKE256 configurations

System Layer	1R SCD		1R SAFE		2R SAFE		3R SAFE		BRAM
	LUT	FF	LUT	FF	LUT	FF	LUT	FF	All builds
SHAKE256 accelerator	4,756	3,001	13,034	3,012	16,957	3,142	16,035	3,143	0
AXI DMA	1,525	2,166	1,527	2,166	1,527	2,166	1,527	2,166	3
AXI-MM data fabric (SmartConnect)	2,297	3,242	2,294	3,242	2,291	3,242	2,298	3,242	0
AXI4-Stream transport	113	198	113	198	113	198	113	198	3
PS AXI interconnect / protocol conversion	527	657	527	657	524	657	527	657	0
SHAKE AXI-Lite control	101	258	101	258	101	258	101	258	0
Reset infrastructure	16	33	16	33	16	33	16	33	0
Non-SHAKE PL infrastructure	4,579	6,554	4,577	6,554	4,568	6,554	4,582	6,554	6
Integrated PL total	9,335	9,555	17,611	9,566	21,525	9,696	20,617	9,697	6

Notes: “AXI4-Stream transport” includes the two data FIFOs and the data-width converter. “Non-SHAKE PL infrastructure” is calculated as integrated-top utilization minus the corresponding SHAKE256 accelerator hierarchy and should not be interpreted as the sum of the individual hierarchical rows because hierarchical utilization may overlap. BRAM is reported as Vivado Block RAM Tiles and is identical across all four deployments: the SHAKE256 accelerator uses no BRAM, while the complete integrated PL uses six tiles. No ILA or debug-hub core is instantiated in these builds.

3.81%, and 4.26%. The largest per-call benefit occurs for F and ordinary PRF , which each save five cycles, giving fixed-depth reductions of 10.87%, 14.71%, and 16.67% at 1R, 2R, and 3R. Component-level measurements localize the complete 13-cycle aggregate saving to absorb/finalization control; the one-permutation Keccak- $f[1600]$ latency, aggregate squeeze interval, and aggregate AXI4-Stream input and output spans remain unchanged between SCD and SAFE FBC at a fixed depth. The lowest aggregate $L2$ -final latency in the 12-configuration matrix is 292 cycles for 3R SAFE FBC, a 37.07% reduction relative to 1R SCD; this cross-depth value reflects the combined effects of round unrolling and SAFE FBC rather than SAFE FBC alone.

The latency reduction is obtained with a clear area cost. Under the common-condition FPGA comparison, SAFE FBC increases LUT count relative to SCD by 106.66%, 75.86%, and 59.77% at 1R, 2R, and 3R, with corresponding slice increases of 114.31%, 78.12%, and 61.34%, while FF overhead remains only 0.13%, 0.38%, and 0.38%. The common-condition accelerator comparison uses neither BRAM nor DSP resources, and all 12 configurations meet the same 20-ns clock-period constraint. These results establish the feasibility of the measured latency–area tradeoff at the common 50-MHz comparison point without implying equal maximum operating frequency or an inferred $F_{\max}$.

Applying the measured primitive latencies to the stated XMSS workload models projects the same fixed-depth saving of 16,603,120 summed primitive- $L2$ cycles at 1R, 2R, and 3R for full-tree key generation, corresponding to reductions of 10.52%, 14.24%, and 16.13%, respectively; 99.17% of the saved cycles arise from repeated F and ordinary PRF evaluations. Under the WOTS-dominant signing model, the projected saving is approximately 7,491 primitive- $L2$ cycles, corresponding to 10.87%, 14.71%, and 16.67% at 1R, 2R, and 3R, respectively. These workload-level values are analytical projections, not measured end-to-end XMSS speedups. The experimental scope is limited to the SHAKE256 accelerator, with measurement boundaries directly characterized through $L2$; complete XMSS execution on the processing system and the defined $L3$ – $L6$ system- and software-level latencies were not measured. Extending the same

measurement discipline to those integration layers, together with separate fault and side-channel evaluation, remains future work.

Data availability

The data supporting the findings of this study are reported in the article. Additional FPGA implementation reports, measurement logs, and derived analysis files are available from the corresponding author upon reasonable request.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the author used OpenAI ChatGPT for manuscript organization, language refinement, sentence restructuring, consistency checking, editorial discussion, and refinement of an explanatory system-integration schematic. The tool was not used to generate or modify experimental data, measurement results, RTL implementations, or reported hardware results. Technical analyses, mathematical derivations, hardware design decisions, interpretation of results, and scientific conclusions were developed and verified by the author. All AI-assisted text and artwork were critically reviewed, edited, and technically verified by the author, who takes full responsibility for the content of the published article.

References

- [1] A. Hülsing, D. Butin, S.-L. Gazdag, J. Rijneveld, A. Mohaisen, XMSS: extended merkle signature scheme, RFC 8391, Internet Research Task Force (May 2018). doi:10.17487/RFC8391.

- [2] D. A. Cooper, D. C. Apon, Q. H. Dang, M. S. Davidson, M. J. Dworkin, C. A. Miller, Recommendation for stateful hash-based signature schemes, NIST Special Publication 800-208, National Institute of Standards and Technology (Oct. 2020). doi:10.6028/NIST.SP.800-208.
- [3] J. Buchmann, E. Dahmen, A. Hülsing, XMSS—a practical forward secure signature scheme based on minimal security assumptions, in: Proc. 4th Int. Workshop Post-Quantum Cryptogr. (PQCrypto), Vol. 7071 of Lecture Notes Comput. Sci., 2011, pp. 117–129. doi:10.1007/978-3-642-25405-5_8. URL: <https://eprint.iacr.org/2011/484>
- [4] National Institute of Standards and Technology, SHA-3 standard: Permutation-based hash and extendable-output functions, Federal Information Processing Standards Publication 202, National Institute of Standards and Technology (Aug. 2015). doi:10.6028/NIST.FIPS.202.
- [5] A. Hülsing, W-OTS+—shorter signatures for hash-based signature schemes, in: Progress in Cryptology—AFRICACRYPT 2013, Vol. 7918 of Lecture Notes Comput. Sci., Springer, 2013, pp. 173–188. doi:10.1007/978-3-642-38553-7_10.
- [6] S. Deshpande, Y. Lee, C. Karakuzu, J. Szefer, Y. Paek, SPHINC-SLET: An area-efficient accelerator for the full SPHINCS+ digital signature algorithm, ACM Transactions on Embedded Computing Systems 24 (5) (2025) 69:1–69:19. doi:10.1145/3728469.
- [7] TUL Corporation, PYNQ-Z2 Reference Manual, TUL Corporation, version 1.0 (May 2018).
- [8] Xilinx, Inc., Zynq-7000 SoC Technical Reference Manual, Xilinx, Inc., version 1.12.2 (Jul. 2018).
- [9] A. Sideris, T. Sanida, M. Dasygenis, Hardware acceleration design of the SHA-3 for high throughput and low area on FPGA, Journal of Cryptographic Engineering 14 (2) (2024) 193–205. doi:10.1007/s13389-023-00334-0.
- [10] E. Homsirikamol, M. Rogawski, K. Gaj, Throughput vs. area trade-offs in high-speed architectures of five round 3 SHA-3 candidates implemented using Xilinx and Altera FPGAs, in: Proc. Int. Workshop Cryptographic Hardw. Embedded Syst. (CHES), Vol. 6917 of Lecture Notes Comput. Sci., 2011, pp. 491–506. doi:10.1007/978-3-642-23951-9_32.
- [11] B. Jungk, M. Stöttinger, Serialized lightweight SHA-3 FPGA implementations, Microprocessors and Microsystems 71 (2019) 102857. doi:10.1016/j.micpro.2019.102857.
- [12] M.-J. O. Saarinen, Accelerating SLH-DSA by two orders of magnitude with a single hash unit, in: Advances in Cryptology – CRYPTO 2024, Vol. 14920 of Lecture Notes in Computer Science, Springer, 2024, pp. 276–304. doi:10.1007/978-3-031-68376-3_9.
- [13] M. Imran, A. Aikata, S. S. Roy, S. Pagliarini, High-speed design of post quantum cryptography with optimized hashing and multiplication, IEEE Transactions on Circuits and Systems II: Express Briefs 71 (2) (2024) 847–851. doi:10.1109/TCSII.2023.3273821.
- [14] Z. Ni, A. Khalid, Z. Zhang, Y. Cui, W. Liu, M. O’Neill, LightHD: A lightweight and high-performance hardware accelerator of CRYSTALS-Dilithium, IEEE Transactions on Computers 75 (5) (2026) 2007–2019. doi:10.1109/TC.2026.3666457.
- [15] N. Pandey, S. Deshpande, D. D. Bohra, D. B. Roy, D. S. Banerjee, J. Szefer, Performance analysis of parameterizable HQC hardware architecture, in: Selected Areas in Cryptography – SAC 2026, 2026, accepted paper; pre-proceedings version. URL: <https://ia.cr/2026/592>
- [16] W. Wang, B. Jungk, J. Wälde, S. Deng, N. Gupta, J. Szefer, R. Niederhagen, XMSS and embedded systems: XMSS hardware accelerators for RISC-V, in: Selected Areas in Cryptography – SAC 2019, Vol. 11959 of Lecture Notes in Computer Science, Springer, 2020, pp. 523–550. doi:10.1007/978-3-030-38471-5_21.
- [17] J. Lopez-Valdivieso, R. Cumplido, Design and implementation of hardware-software architecture based on hashes for SPHINCS+, ACM Transactions on Reconfigurable Technology and Systems 17 (4) (2024) 1–22. doi:10.1145/3653459.
- [18] Y. Cao, Y. Wu, W. Wang, X. Lu, S. Chen, J. Ye, C.-H. Chang, An efficient full hardware implementation of extended Merkle signature scheme, IEEE Trans. Circuits Syst. I, Reg. Papers 69 (2) (2022) 682–693. doi:10.1109/TCSI.2021.3115786.
- [19] Y. Cao, Y. Wu, L. Qin, S. Chen, C.-H. Chang, Area, time and energy efficient multicore hardware accelerators for extended Merkle signature scheme, IEEE Trans. Circuits Syst. I, Reg. Papers 69 (12) (2022) 4908–4921. doi:10.1109/TCSI.2022.3200987.
- [20] J. P. Thoma, D. Hartlief, T. Güneysu, Agile acceleration of stateful hash-based signatures in hardware, ACM Transactions on Embedded Computing Systems 23 (2) (2024) 29:1–29:29. doi:10.1145/3567426.
- [21] D. Amiet, L. Leuenberger, A. Curiger, P. Zbinden, FPGA-based SPHINCS+ implementations: Mind the glitch, in: 2020 23rd Euromicro Conference on Digital System Design (DSD), IEEE, 2020, pp. 229–237. doi:10.1109/DSD51259.2020.00046.
- [22] S. Das, S. P. Khatri, A timing-driven approach to synthesize fast barrel shifters, IEEE Transactions on Circuits and Systems II: Express Briefs 55 (1) (2008) 31–35. doi:10.1109/TCSII.2007.908951.
- [23] ARM, AMBA 4 AXI4-Stream Protocol Specification, ARM, version 1.0, ID030510 (2010).
- [24] Xilinx, Inc., AXI DMA v7.1 LogiCORE IP Product Guide, Xilinx, Inc., June 14, 2019 (Jun. 2019).
- [25] Xilinx, Inc., Zynq-7000 SoC Data Sheet: Overview, Xilinx, Inc., version 1.11.1 (Jul. 2018).