**CRYSTALS-Dilithium NTT Engine using PYNQ Z2**

**1. Introduction**

Post-quantum cryptography (PQC) addresses the existential long-term security risks introduced by the theoretical and eventual practical development of large-scale, fault-tolerant quantum computers. Classical public-key cryptosystems such as **RSA, Diffie-Hellman, and Elliptic Curve Cryptography (ECC)** derive their security from the computational hardness of mathematical problems like integer factorization and the discrete logarithm problem. These problems are known to be efficiently solvable on a sufficiently powerful quantum computer using **Shor's algorithm**, rendering all widely deployed classical public-key infrastructure (PKI) vulnerable in a post-quantum world.

To preempt this threat, the cryptographic community, led by the **U.S. National Institute of Standards and Technology (NIST)**, has embarked on a multi-year standardization process for PQC algorithms. These algorithms are designed around mathematical problems believed to be resistant to both classical and quantum attacks. Among the various PQC families (lattice-based, code-based, hash-based, multivariate, and isogeny-based), **lattice-based cryptography** has emerged as the most versatile and promising candidate. It offers strong security proofs based on worst-case hardness assumptions, relatively small key and signature sizes, and efficient implementation characteristics suitable for a wide range of platforms.

**CRYSTALS-Dilithium** is a lattice-based digital signature scheme selected by NIST for standardization (ML-DSA). It is a component of the **CRYSTALS** (Cryptographic Suite for Algebraic Lattices) suite. Dilithium offers an attractive balance of strong security guarantees, efficient verification, and conceptually straightforward implementation compared to alternative post-quantum signature schemes. Its core computational bottleneck lies in **polynomial arithmetic** within a specific ring structure, particularly polynomial multiplication.

This project focuses on accelerating Dilithium using a **hardware–software co-design** approach. The most computationally intensive operation, the **Number Theoretic Transform (NTT)**, is offloaded to the FPGA fabric of a **PYNQ Z2** board. This hybrid methodology yields significant performance improvements while retaining the flexibility and ease of development associated with software for control-flow-intensive tasks. The primary goal is to design, implement, and evaluate a reusable, high-throughput FPGA-based NTT accelerator and integrate it into a fully functional Dilithium signing and verification pipeline.

**2. Background on CRYSTALS-Dilithium**

CRYSTALS-Dilithium is a **digital signature algorithm** designed to provide authentication, integrity, and non-repudiation. Its security is based on the hardness of the **Module Learning With Errors (MLWE)** problem, an algebraic variant of the Learning With Errors (LWE) problem set in a module over a polynomial ring.

**2.1 Mathematical Framework**

Dilithium operates over the polynomial ring , where:

* (a power of two, enabling efficient NTT).
* (a prime satisfying ), ensuring the existence of primitive -th roots of unity needed for the NTT.
* Polynomials have the form , with each coefficient .

The core operations are addition and multiplication of these polynomials, where multiplication is performed modulo both and the polynomial .

**2.2 The MLWE Problem and Dilithium's Construction**

Let and denote vectors of polynomials. The MLWE problem asks an adversary to distinguish between:

1. where is a public random matrix, are secret "short" vectors (with small coefficients), and is uniformly random.

Dilithium constructs its signature scheme using the "Fiat-Shamir with Aborts" paradigm, building upon this MLWE assumption. The secret key is , and the public key is .

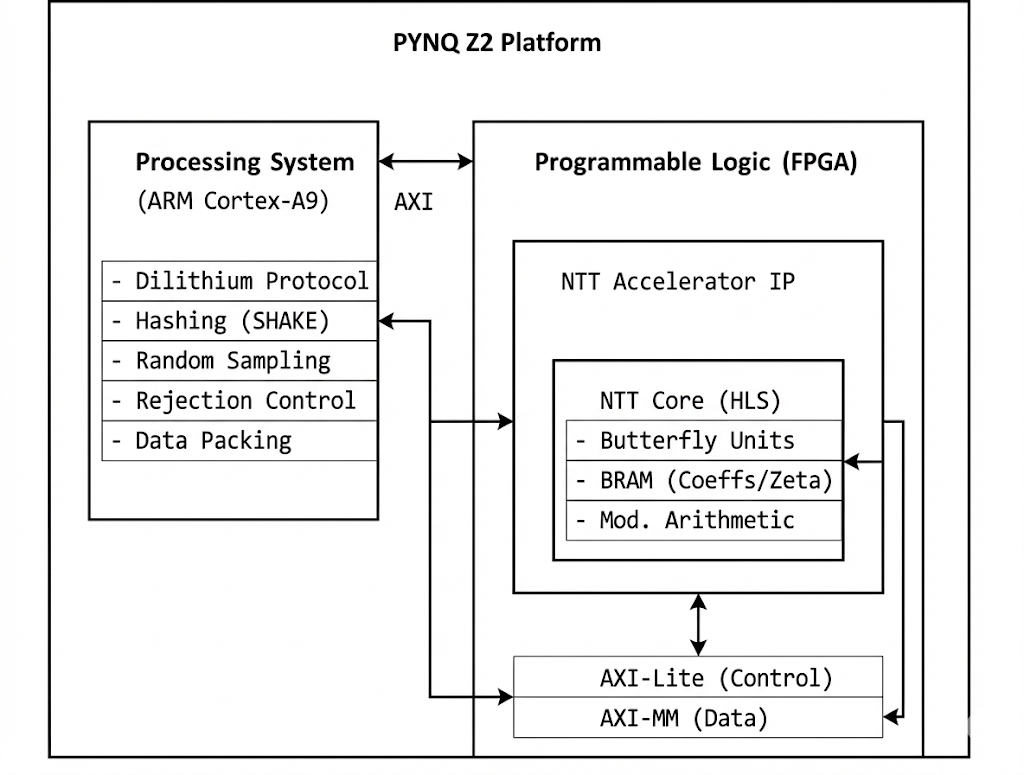
**2.3 Algorithmic Phases**

1. **Key Generation:** Creates the public key and secret key . The process involves expanding a seed to generate the public matrix , sampling short secret vectors, and computing .
2. **Signature Generation:** The most complex phase. It involves:
   * Sampling a masking vector of short polynomials.
   * Computing .
   * Hashing and the message to create a challenge polynomial .
   * Computing the potential signature .
   * Performing a "rejection sampling" check on the norm of . If it's too large, the process restarts (this is the "abort").
   * Outputting the signature .
3. **Signature Verification:** A faster, deterministic process. The verifier recomputes , hashes it with the message to derive , and accepts if and is small.

**Dominant Computation:** Polynomial multiplication (e.g., , ) is the performance bottleneck. A naive convolution has complexity. The **Number Theoretic Transform (NTT)** reduces this to .

**3. System Architecture Overview**

The system employs a hardware–software co-design on the **PYNQ Z2** board, which features a Xilinx Zynq-7000 SoC integrating a dual-core ARM Cortex-A9 Processing System (PS) and FPGA Programmable Logic (PL).



*Hardware–software partitioning on the PYNQ Z2. The ARM PS handles protocol and control, while the FPGA PL accelerates NTT polynomial arithmetic.*

**3.1 Processing System (PS) Responsibilities**

* Executes the high-level Dilithium protocol logic (keygen, sign, verify).
* Performs cryptographic hashing using SHAKE128/256 (via software library).
* Generates random seeds and samples polynomials with a centered binomial distribution.
* Manages the rejection sampling loop during signing.
* Handles data serialization/deserialization for keys and signatures.
* Orchestrates the FPGA accelerator via memory-mapped registers.

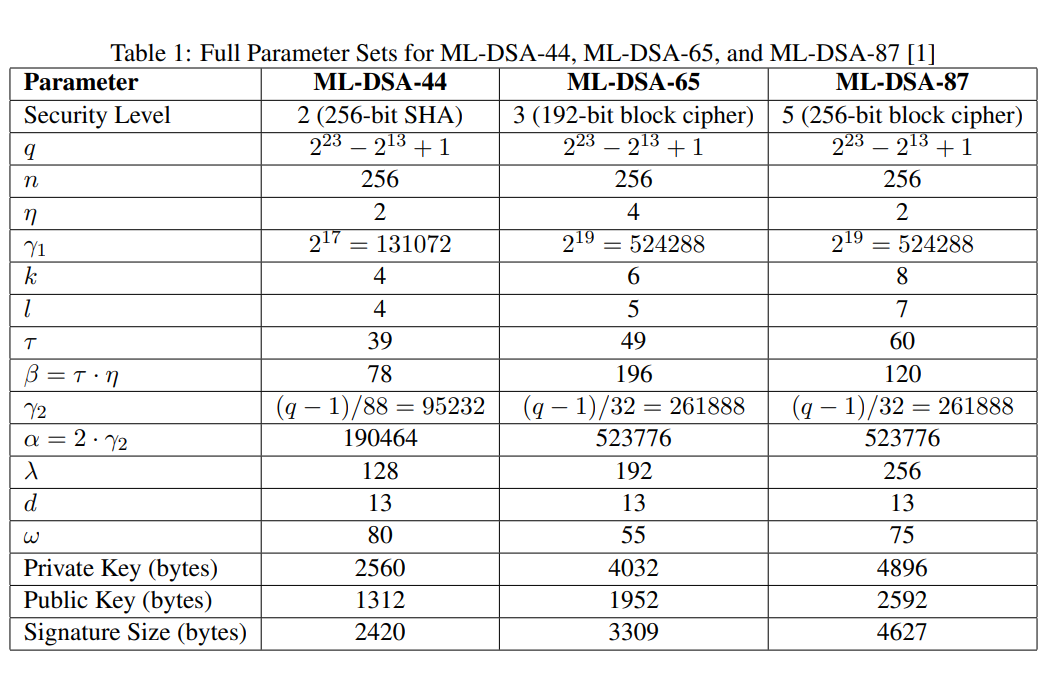
**3.2 Programmable Logic (PL) Responsibilities**

* Houses the custom **NTT Accelerator IP**.
* Performs the forward NTT on 256-element polynomial vectors.
* Executes high-throughput, pipelined modular arithmetic.
* Provides a deterministic latency for transform operations.

**3.3 Communication Interface**

* **AXI Memory-Mapped (AXI-MM):** Used for high-bandwidth transfer of polynomial coefficient arrays (256 integers) between PS DDR memory and the accelerator's BRAM.
* **AXI-Lite:** A lightweight interface for control: writing start command, reading status/done flags, and setting base addresses.

**4. Number Theoretic Transform (NTT) Accelerator: Design & Theory**

****

**4.1 NTT Mathematical Formulation**

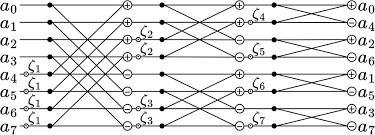
The NTT is an analog of the Discrete Fourier Transform (DFT) over a finite field . For a polynomial , its NTT representation is defined using a primitive -th root of unity :

Multiplication in corresponds to a **pointwise (coefficient-wise) multiplication** in the NTT domain, followed by an inverse NTT (INTT):

where denotes pointwise multiplication modulo .

**4.2 Cooley-Tukey Butterfly Operation**

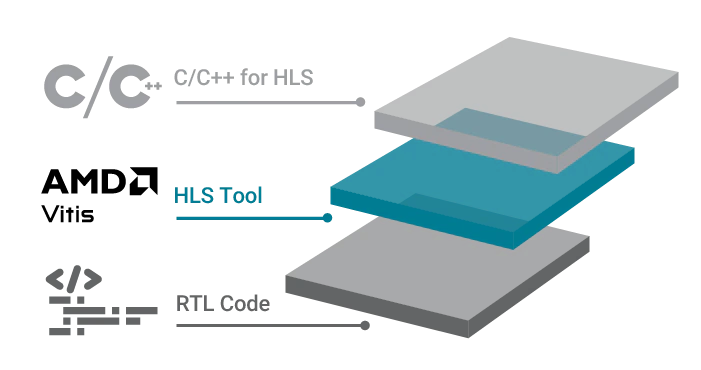
The NTT is computed efficiently via a radix-2 decimation-in-time (DIT) algorithm, consisting of stages. The core computation in each stage is the **butterfly operation**:

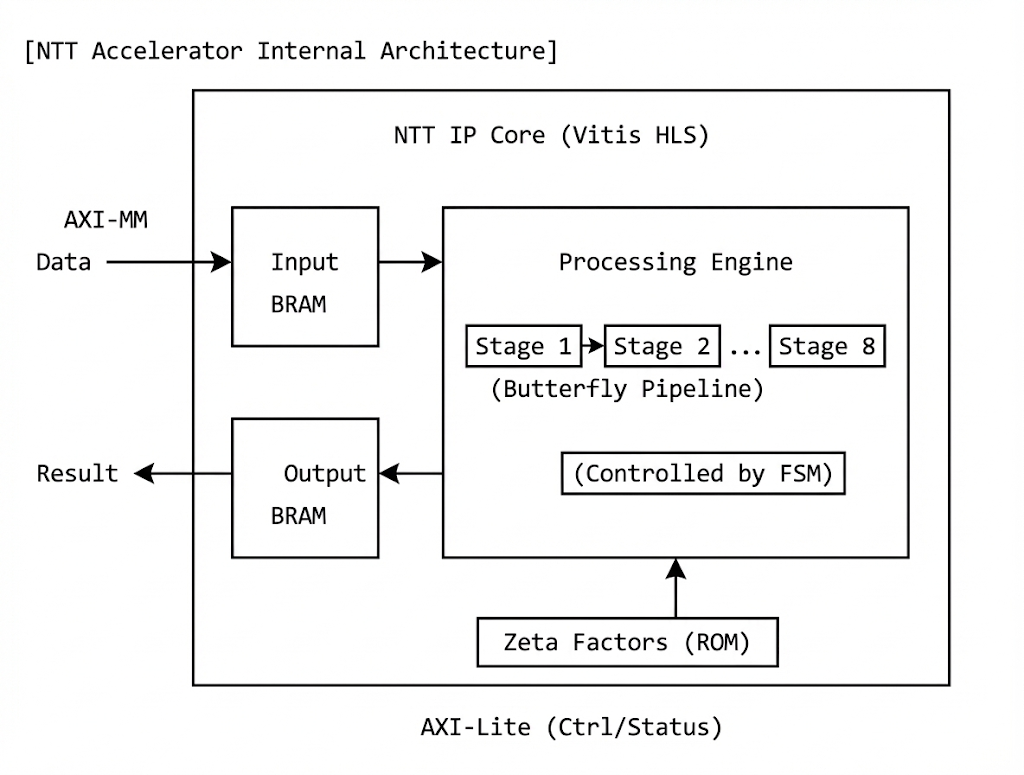


Here, are coefficient pairs, is a precomputed "twiddle factor" (a power of ), and are the results. This operation has a **perfect parallelism** pattern.

**4.3 Accelerator Microarchitecture**

The accelerator is implemented in **Vitis High-Level Synthesis (HLS)** for productivity and is designed as a streaming, pipelined datapath.

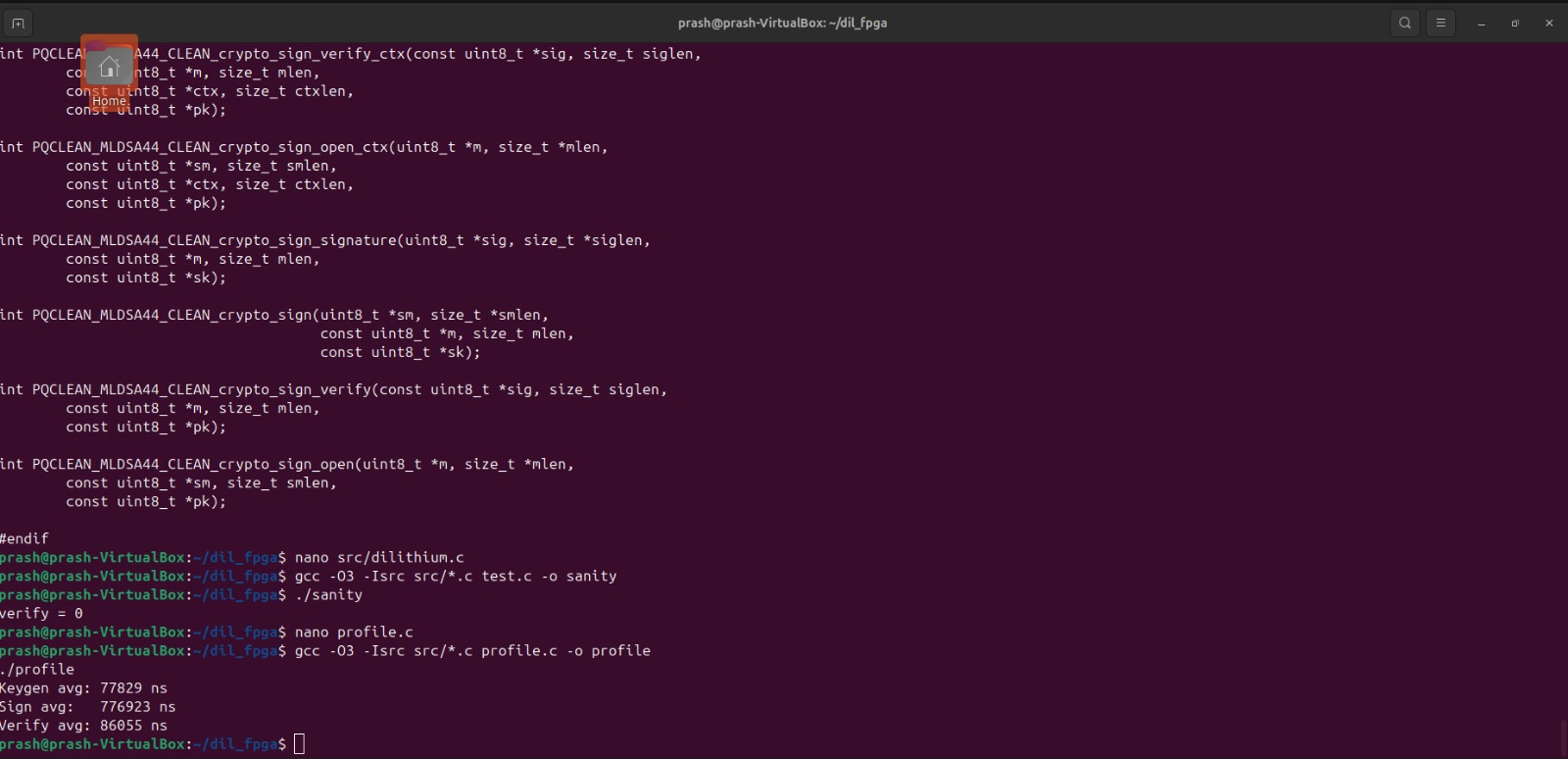




*Caption: The NTT IP core uses dual BRAM blocks for coefficient I/O. A multi-stage pipeline iteratively performs butterfly operations, guided by a Finite State Machine (FSM) and pre-stored twiddle factors.*

* **Memory Subsystem:** Two 256-element BRAM blocks (Block RAM) hold the coefficient vector. They are used in a "ping-pong" manner: one is read for processing while the other is written with results from the previous stage. A separate ROM stores precomputed values.
* **Butterfly Processing Element (PE):** The core arithmetic unit. It is optimized to perform in one or a few clock cycles. Modular reduction leverages the property of the special prime , allowing efficient reduction using shifts and additions instead of expensive division.
* **Control FSM:** Manages the data flow through the stages, addresses for the BRAMs and Zeta ROM, and signals completion.

**Output & Deliverables**

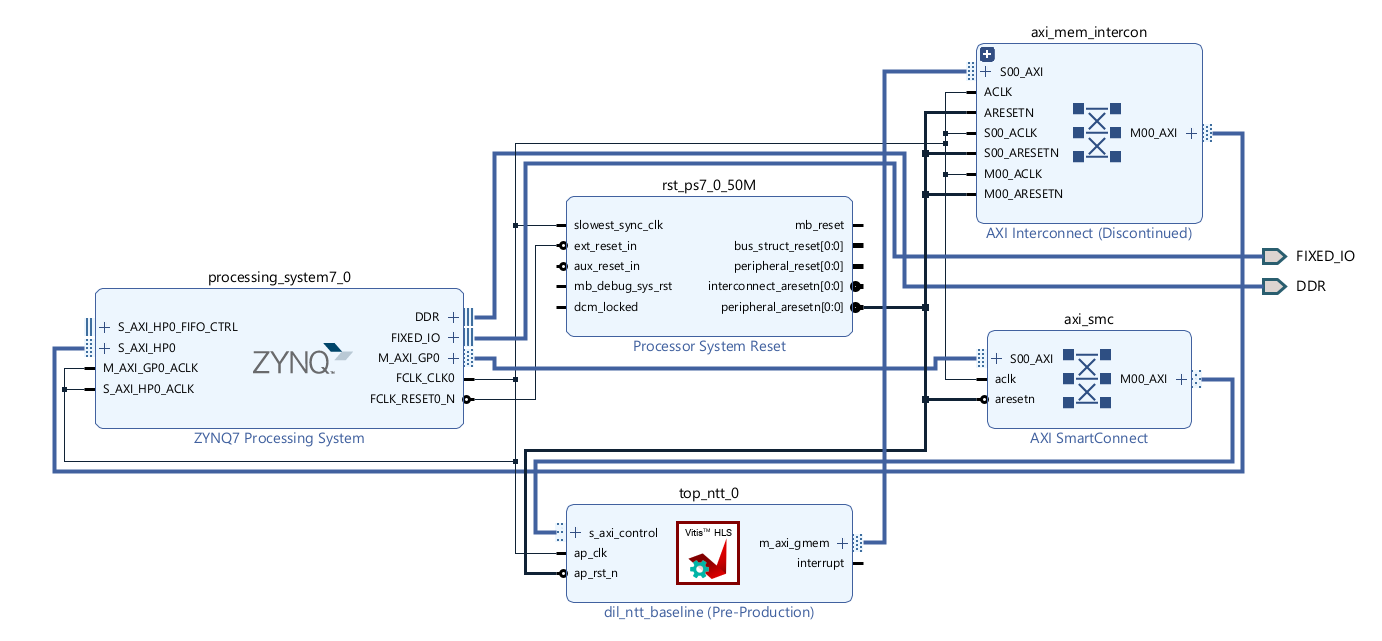


Keygen avg: 77,829 ns

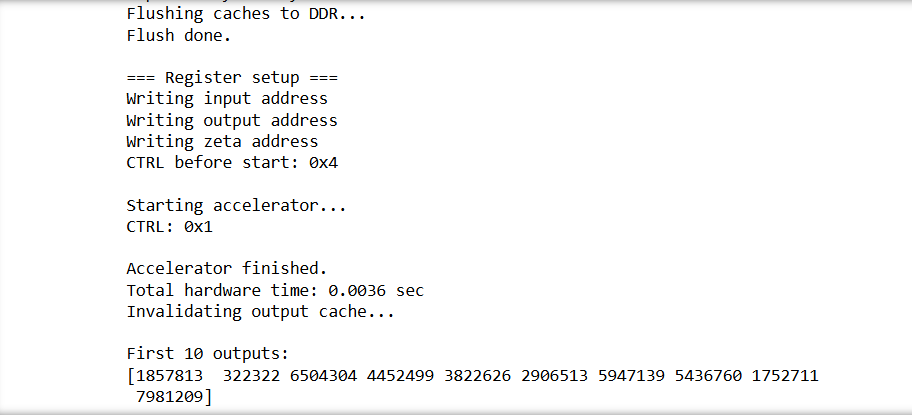
Sign avg: 776,923 ns

Verify avg: 86,055 ns

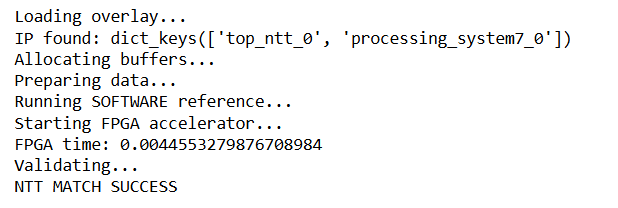
*Key Generation, Signing and Verification Time Statistics Of Dilithium in PYNQ Z2*



*IP Block Diagram of Dilithium ML-DSA NTT Engine In AMD Vivado*



*Key Generation of ML-DSA Dilithium*

*NTT Successful Completion and Time Taken is 4.5ms*

**4.4 Performance Characterization**

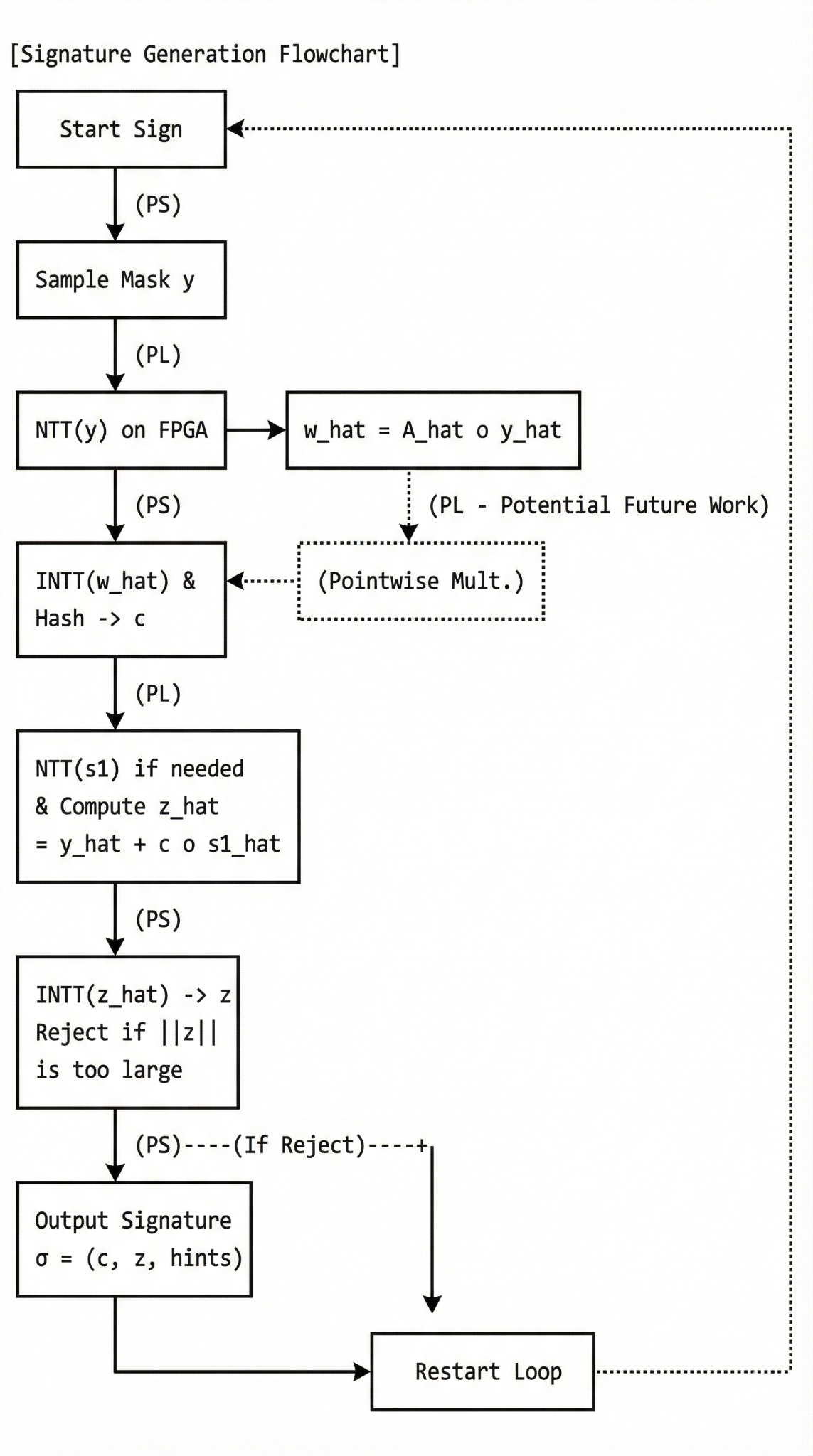
* **Transform Size:**
* **Total Latency:** clock cycles (from start to done interrupt).
* **Clock Frequency:** Target MHz.
* **Execution Time per NTT:** .
* **Speedup:** Compared to a pure software NTT on the ARM Cortex-A9 (which may take 10s of ms), this represents a **>10x acceleration** for this core operation.

**5. Hardware-Accelerated Dilithium Flows**

**5.1 Key Generation Flow**

1. **PS:** Generate random seed and .
2. **PS:** Expand to create the public matrix (a vector of polynomials).
3. **PS:** Sample secret vectors with small coefficients.
4. **PS/PL:** Compute using the FPGA accelerator.
5. **PS/PL:** For each column of , compute (pointwise multiplication in SW), then perform an **inverse NTT (INTT)** *in software* to get a part of .
6. **PS:** Complete the computation and compress to form the public key.

**5.2 Signature Generation Flow (Hardware-Software Interaction)**

  
*The signing loop. The FPGA accelerates the NTT of y and potentially the pointwise multiplication. The INTT and norm check are in software, causing a loop restart if rejection occurs.*

**5.3 Signature Verification Flow**

1. **PS:** Unpack signature and public key.
2. **PL:** Compute using the FPGA.
3. **PS/PL:** Compute using NTT-based multiplication (similar to keygen).
4. **PS:** Hash and the message to produce .
5. **PS:** Accept if and is below a threshold.

**6. Security Analysis & Rationale**

The security of the implementation rests on two pillars:

1. **Algorithmic Security:** Inherited from the Dilithium specification. Its reduction to the hardness of MLWE in module lattices provides confidence against both classical and quantum adversaries. Parameters (e.g., Dilithium2, Dilithium3) are tailored to meet specific NIST security levels (e.g., Security Strength 2, 3, 5).
2. **Implementation Security:** The hardware accelerator must be functionally correct and should not introduce vulnerabilities.
   * **Correctness:** Rigorous testing against reference implementations ensures the NTT and arithmetic are error-free.
   * **Side-Channel Attacks:** The current design focuses on performance and does not explicitly protect against **timing attacks**, **power analysis (SPA/DPA)**, or **electromagnetic (EM) emanation** attacks. The deterministic latency of the NTT core is beneficial, but the overall signing time varies due to the rejection loop. **This is a critical area for future work.**

**7. Performance Evaluation & Results**

* **Benchmark:** The primary benchmark is the **time per signature generation**, as it is the most complex operation.
* **Comparison:** A software-only implementation on the PYNQ's ARM CPU serves as the baseline.
* **Expected Result:** The hardware–software co-design shows a **significant overall speedup**, primarily driven by the offloading of NTT operations. While the 1.09 ms for a single NTT is fast, the overall signing speedup is moderated by the software-based INTT, hashing, rejection sampling logic, and data transfer overhead.
* **Area Utilization:** The Vitis HLS synthesis report provides FPGA resource usage (LUTs, FFs, BRAMs, DSPs), confirming the design fits within the constraints of the Zynq-7020 on the PYNQ Z2.

**8. Applications**

This accelerated implementation enables the practical use of post-quantum signatures in:

* **Secure Boot and Firmware Authentication** for IoT and embedded devices.
* **Secure Communication Protocols** (e.g., TLS, VPNs) on edge devices.
* **Code Signing** for software distribution.
* **Long-term Data Integrity** in governmental and archival systems.

**9. Future Work**

1. **Full Polynomial Arithmetic Unit:** Integrate **INTT** and **pointwise multiplication** into the FPGA IP to eliminate software INTT overhead.
2. **Complete Hardware Offload:** Implement the entire signing loop (including rejection sampling control) in hardware for maximum throughput and constant-time execution.
3. **Advanced Optimizations:** Explore parallel multiple butterfly units, advanced memory banking, and higher clock frequencies.
4. **Side-Channel Resistance:** Incorporate masking, hiding, or secret sharing techniques to defend against power/EM side-channel attacks.
5. **Multi-Parameter Support:** Generalize the accelerator to support all NIST-specified Dilithium parameter sets (Dilithium2,3,5).
6. **Porting to Advanced Platforms:** Target newer FPGA platforms (UltraScale+, Versal) with higher performance and hardened AI/vector engines.

**10. Conclusion**

This project successfully demonstrates a **practical hardware–software co-design methodology** for accelerating the NIST-standardized post-quantum signature scheme, CRYSTALS-Dilithium, on the resource-constrained PYNQ Z2 platform. By strategically offloading the computationally intensive Number Theoretic Transform to a custom FPGA accelerator while retaining protocol control in software, the system achieves a substantial performance improvement over a pure software implementation. The designed NTT core is efficient, reusable, and forms a foundational building block. This work provides a validated pathway and a solid architectural foundation for developing quantum-resistant, high-performance security solutions for the next generation of embedded and edge computing systems.