Benchmarking RSA on Softcore Processors with Pynq-Z2

December 9, 2025

California State Polytechnic University, Pomona
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
Caleb Pai, Winson Zhu, Paul Kim, Hyukjin Jeong
Emails: {ckpai,paulkim,winsonzhu,hyukjinjeong}@cpp.edu

Dr. Mohamed El-Hadedy Aly

Abstract

This project evaluates the performance benefits of accelerating Montgomery modular multiplication on the PYNQ-Z2 SoC by comparing software execution on the ARM Cortex-A9 processing system with a custom hardware accelerator synthesized onto the Zynq-7020 programmable logic. Montgomery multiplication is a key primitive in public-key cryptography and dominates the execution time of RSA for large operands. To analyze the advantages of hardware acceleration, a 2048-bit iterative, word-serial Montgomery multiplier was designed in Verilog with support for large operand sizes and integrated into the Zynq-7020 FPGA fabric. The design includes an internal finite-state machine, iterative accumulation data path, and a dedicated AXI4-Lite register interface exposed to the processing system.

A matching software implementation was developed in C and executed under PetaLinux on the ARM Cortex-A9. Both hardware and software outputs were validated against OpenSSL-generated test vectors to ensure bit-accurate correctness. A C based benchmark harness measured execution latency using clock_gettime() for RSA-1024 and RSA-2048 encryption and decryption workloads. The experimental results demonstrate that the hardware-accelerated design achieves substantial speedup over software execution, confirming the effectiveness of offloading Montgomery multiplication to FPGA logic for improving RSA performance on resource-constrained embedded platforms.

Introduction

Public-key cryptography algorithms such as RSA rely heavily on large-integer modular arithmetic, and the performance of modular multiplication becomes a significant performance limitation for large operands. Executing these operations efficiently is especially challenging for general processors in embedded platforms, where resources and clock speeds are constrained. Montgomery modular multiplication is used to improve the efficiency of modular arithmetic by replacing division with simpler shift- and add-based operations, making it well-suited for hardware implementation. Prior IEEE studies have demonstrated that custom Montgomery multiplier architectures can achieve higher throughput and reduced latency compared to software-based implementations. However, much of the existing work focuses on ASIC designs or complete RSA processors, and fewer studies evaluate the performance gap between software execution and hardware acceleration on an SoC platform.

The Xilinx PYNQ-Z2 board provides an ideal environment for such analysis, as it integrates an ARM Cortex-A9 Processing System with FPGA Programmable Logic, enabling the same algorithm to be executed on both hardware and software within a unified system. This project implements Montgomery modular multiplication in both domains: a C version running under PetaLinux on the PS, and a Verilog-based hardware accelerator synthesized onto the PL and accessed through an AXI4-Lte interface. Correctness is verified using OpenSSL-generated test vectors, and performance is benchmarked across 512-,1024-, and 2048-bit operands. The goal of this work is to quantify the benefits of FPGA-based acceleration for large-integer modular arithmetic and to characterize the performance trade-offs between CPU-based and hardware-based computation on embedded SoC platforms.

Contribution

This work develops a hardware implementation of Montgomery multiplication on the PYNQ-Z2 FPGA to support efficient large-integer arithmetic used in RSA-2048. The design focuses solely on accelerating the Montgomery reduction step, which is the most time-consuming component of RSA arithmetic on embedded processors. The hardware unit performs iterative word-level operations in parallel logic, reducing computation latency compared to software execution on the ARM Cortex-A9. An AXI4-Lite interface connects the multiplier to the processing system, enabling operand transfers, control signaling, and result of retrieval through a simple memory-mapped structure. A Python-based control and measurement framework coordinates data movement and records execution times for comparison with software performance. Results show that offloading Montgomery multiplication to hardware produces clear latency improvements, illustrating how targeted acceleration of a single arithmetic primitive can enhance the feasibility of cryptographic workloads on resource-constrained platforms. The design demonstrates how FPGA-based parallelism benefits modular arithmetic, reduces processor load, and provides a scalable foundation for future extensions such as full modular exponentiation or complete RSA data paths.

Related Work

Prior research on Montgomery multiplication provides several architectural strategies that directly influence lightweight FPGA implementations, such as the one developed in this project. Yang *et al.* propose a modified Montgomery multiplication framework that separates multiplication and reduction stages to prevent over-large residues and shorten the critical path. Their design demonstrates that simplifying intermediate correction steps can improve modular multiplication throughput without significantly increasing hardware cost, making their approach well aligned for resource-efficient RSA accelerators [1].

Beyond algorithmic refinement, scalable hardware architectures have also been explored to support large operand sizes with limited resources. Tenca and Koç introduce a word-based radix-2 Montgomery architecture in which operands are processed at the word level, allowing designers to tune the level of parallelism depending on area, timing, and hardware constraints. Their architecture supports variable precision through reuse-in-time of processing elements, offering a useful design space for FPGA implementations where resource limits and flexibility are critical considerations [2]. This word-serial perspective is especially relevant for embedded SoC platforms such as the Pynq-Z2, where balancing area and performance is essential.

Practical FPGA-oriented designs further examine iterative data paths for handling large RSA operand sizes. Renardy *et al.* present a 2048-bit Montgomery multiplier that adopts an iterative architecture supported by SIPO/PISO interfaces to overcome I/O width limitations. Their implementation achieves a fixed latency of 2048 cycles with moderate logic utilization, demonstrating that sequential, per-cycle update architectures provide an excellent area-time trade-off for RSA-2048 arithmetic on mid-range FPGA devices [3].

These works indicate that high-performance RSA acceleration does not require fully parallel architectures. Instead, algorithmic simplification, word-based processing, and iterative hardware structures can provide efficient and scalable solutions. These principles directly inform the design presented in this work, which implements a standalone Montgomery multiplication accelerator connected to the processing system through an AXI4-Lite interface, enabling efficient hardware assisted RSA-2048 modular arithmetic on a resource-constrained FPGA platform.

System Architecture

The system integrates a Montgomery multiplication hardware unit implemented on the programmable logic of the PYNQ-Z2 FPGA with control software executing on the embedded ARM Cortex-A9 processor. This design focuses exclusively on accelerating the Montgomery reduction step used in RSA-2048 arithmetic. The hardware unit operates on 2048-bit operands represented as sixty-four 32-bit words. Internally, the multiplier uses an iterative, word-serial data path that performs one reduction step per clock cycle. This structure minimizes resource usage while maintaining the ability to process large operands.

Communication between the processing system and programmable logic is achieved through an AXI4-Lite interface. The multiplier exposes memory-mapped registers for the loading of operand A, operand B, the modulus N., and the precomputed constant n'. A control register initiates the Montgomery multiplication, and a status register indicates completion. The design follows a simple sequence: write operands, assert start, poll done, read result. This approach reduces software overhead and simplifies synchronization between the processor and hardware.

A Python-based control framework running on the PYNQ Linux environment manages to register transfers and execution timing. The framework provides functions for loading operands, invoking the hardware multiplier, and retrieving 2048-bit results for verification. This software layer also enables cycle-accurate measurements and comparisons with a software-only Montgomery multiplication routine.

This architecture emphasizes modularity and reproducibility. By isolating Montgomery multiplication as the sole hardware-accelerated component, the system provides a clear foundation for analyzing FPGA efficiency in large-integer modular arithmetic without introducing the complexity of full RSA exponentiation hardware. This structure can also expand into future extensions, such as pipelined multipliers or complete RSA data paths, without redesigning the AXI interface.

Evaluation

The evaluation measures the performance and correctness of the hardware Montgomery multiplication implemented on the PYNQ-Z2 FPGA. Results are compared against a software-only implementation executing on the ARM Cortex-A9 processor. Performance is evaluated for RSA-1024 and RSA-2048 operations using average cycle count, execution time, throughput, and relative speedup.

Tables I and II summarize the performance results. For RSA-2048, the hardware implementation achieves approximately 7.5× speedup for encryption and 12.5× speedup for decryption compared to software execution. Similar performance trends are observed for RSA-1024, demonstrating consistent acceleration across different key sizes. The higher speedup observed during decryption is attributed to the increased number of Montgomery multiplications required in the decryption process.

TABLE I
RSA-2048 PERFORMANCE RESULTS

Operation	Implementation	Avg. Cycles	Avg, Time	Throughput (ops/s)	Speedup

		(M-cycle)	(ms)		
Encrypt	**Software**	**608.4**	**923.1**	**10**	
Encrypt	**Hardware**	**7.977**	**12.31**	**81**	**7.5x**
Decrypt	**Software**	**148.9**	**230.1**	**4**	
Decrypt	**Hardware**	**12.02**	**18.46**	**54**	**12.5x**

TABLE II
RSA-1024 PERFORMANCE RESULTS

Operation	Implementation	Avg. Cycles (M-cycle)	Avg, Time (ms)	Throughput (ops/s)	Speedup
Encrypt	**Software**	**106.1**	**23.08**	**43**	
Encrypt	**Hardware**	**1.966**	**3.076**	**325**	**7.5x**
Decrypt	**Software**	**406.7**	**61.54**	**16**	
Decrypt	**Hardware**	**3.609**	**5.384**	**185**	**11.43x**

Correctness is verified by comparing hardware and software outputs. In all test cases, decrypted ciphertexts match the original plaintext, confirming the correct operation of the hardware multiplier within the RSA computation flow. These results show that accelerating Montgomery multiplication alone provides meaningful performance improvements for RSA operations on resource-constrained FPGA platforms.

Conclusion

This project evaluated the impact of offloading Montgomery modular multiplication from a general-purpose processor to FPGA programmable logic on the PYNQ-Z2 SoC platform. A 2048-bit word-serial, iterative Montgomery multiplier was implemented in Verilog on the Zynq-7020 PL and connected to the ARM Cortex-A9 processing system through an AXI4-Lite memory-mapped interface. A functionally equivalent C implementation ran under PetaLinux on the Cortex-A9. Both hardware and software paths were validated against OpenSSL-generated test vectors for RSA-1024 and RSA-2048 to ensure bit-accurate correctness.

The performance results demonstrate that accelerating only the Montgomery multiplication kernel yields substantial end-to-end speedup for RSA operations. For RSA-2048 (Table I), software encryption required approximately 6.08×10^8 cycles, whereas the hardware-assisted design reduced this to about 7.98×10^6 cycles, corresponding to a speedup of roughly 7.5x. Decryption showed an even larger improvement, from about 1.49×10^8 cycles in software to 1.20×10^7 cycles in hardware, yielding a speedup of 12.5x. Similar trends were observed for RSA-1024 (Table II), where hardware acceleration achieved 7.5x speedup for encryption and 11.4x for decryption. These consistent gains across key sizes confirm that the iterative hardware datapath effectively accelerates large-integer modular arithmetic on a mid-range FPGA.

At the same time, the overall system architecture remains modular and extensible. By isolating Montgomery multiplication as a standalone accelerator with a simple AXI4-Lite register interface, the design can be integrated into different RSA implementations or other public-key cryptosystems without redesigning the software stack. This structure also provides a clear path for future work, including pipelined or higher-throughput multiplier designs, support for larger key sizes, or comparative studies with alternative softcore or RISC-V processors on the same FPGA.

Overall, the results show that targeted hardware acceleration of Montgomery modular multiplication significantly improves the practicality and

performance of RSA-1024/2048 workloads on the PYNQ-Z2, while preserving flexibility for further architectural exploration.

Reference

[1] Ching-Chao Yang, Tian-Sheuan Chang and Chien-Wei Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 7, pp. 908-913, July 1998, doi: 10.1109/82.700944.

[2] A. F. Tenca and C. K. Koc, "A scalable architecture for modular multiplication based on Montgomery's algorithm," in *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1215-1221, Sept. 2003, doi: 10.1109/TC.2003.1228516.

[3] A. P. Renardy, N. Ahmadi, A. A. Fadila, N. Shidqi and T. Adiono, "Hardware implementation of montgomery modular multiplication algorithm using iterative architecture," *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Surabaya, Indonesia, 2015, pp. 99-102, doi: 10.1109/ISITIA.2015.7219961.