SW/HW Codesign of the Post-Quantum Cryptography Algorithm NTRUEncrypt Using HLS and RTL Design Methodologies

Farnoud Farahmand, Duc Tri Nguyen, Viet B. Dang*, Ahmed Ferozpuri and Kris Gaj

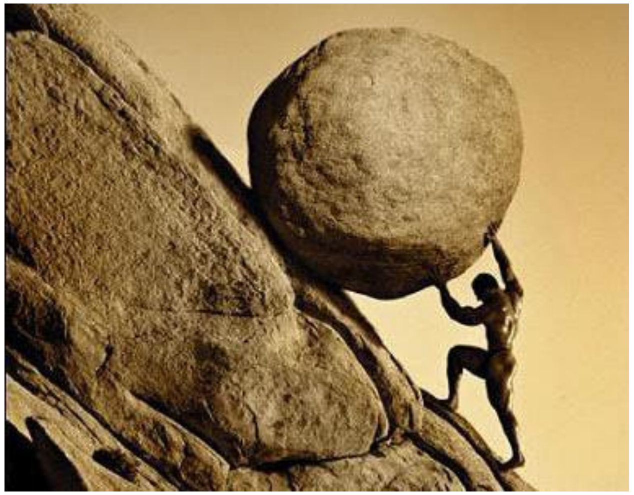
George Mason University

Post-Quantum Cryptography (PQC)

- Ongoing NIST PQC standardization process
- Total 69 submissions in Round 1 and

  26 submissions qualified to Round 2

Challenges

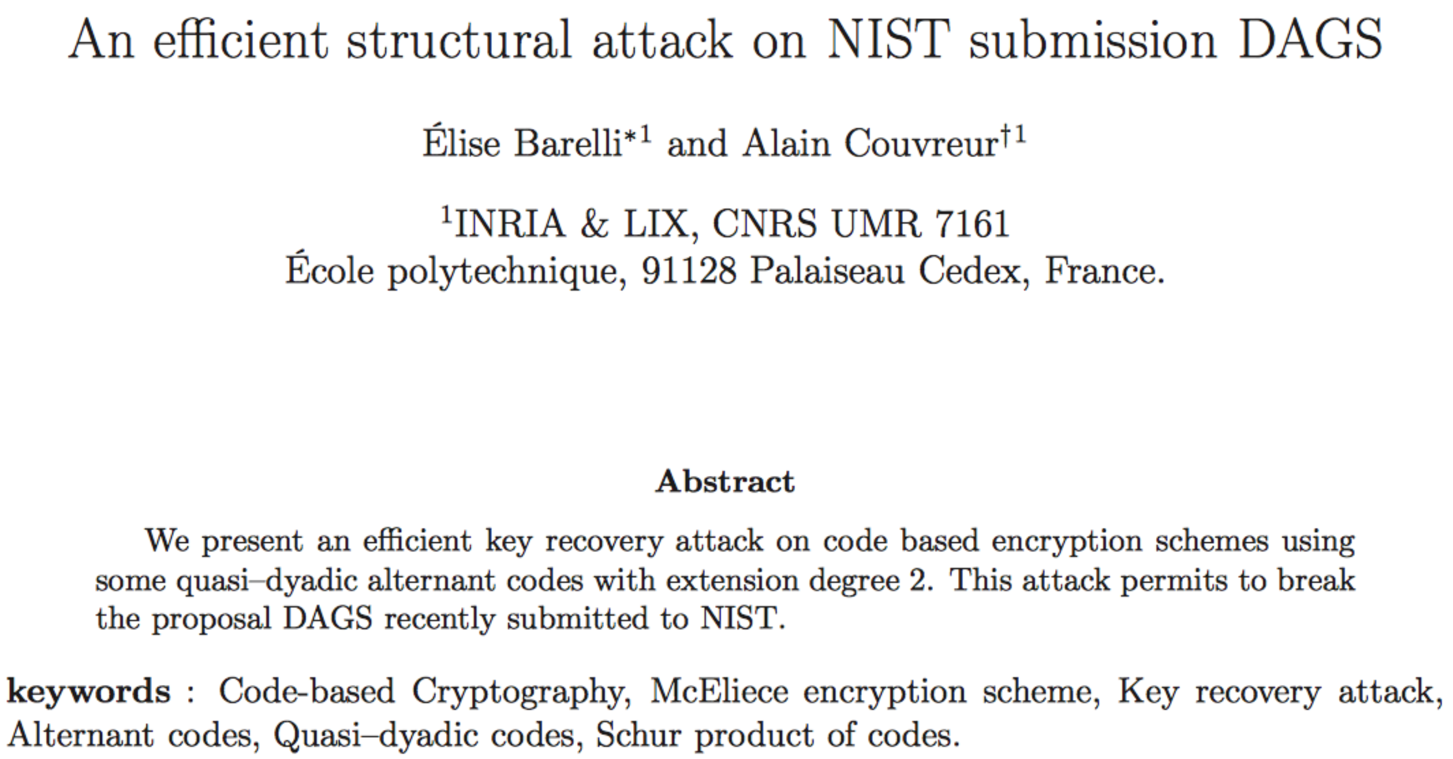


- Mathematical complexity
- Large amount of man-power
- New types of basic operations
- Constant-time implementations
- Need for new SCA (Side-Channel Attack) countermeasures against power and electromagnetic analysis

Risks of Early Hardware Implementations

GMU implementation of DAGS developed in Fall 2017-Spring 2018. Preliminary results presented at the Code-Based Cryptography (CBC) workshop in April 2018.

Attack against DAGS announced on May 16, 2018.
DAGS not qualified to Round 2



An efficient structural attack on NIST submission DAGS

Élise Barelli*[1] and Alain Couvreur†[1]

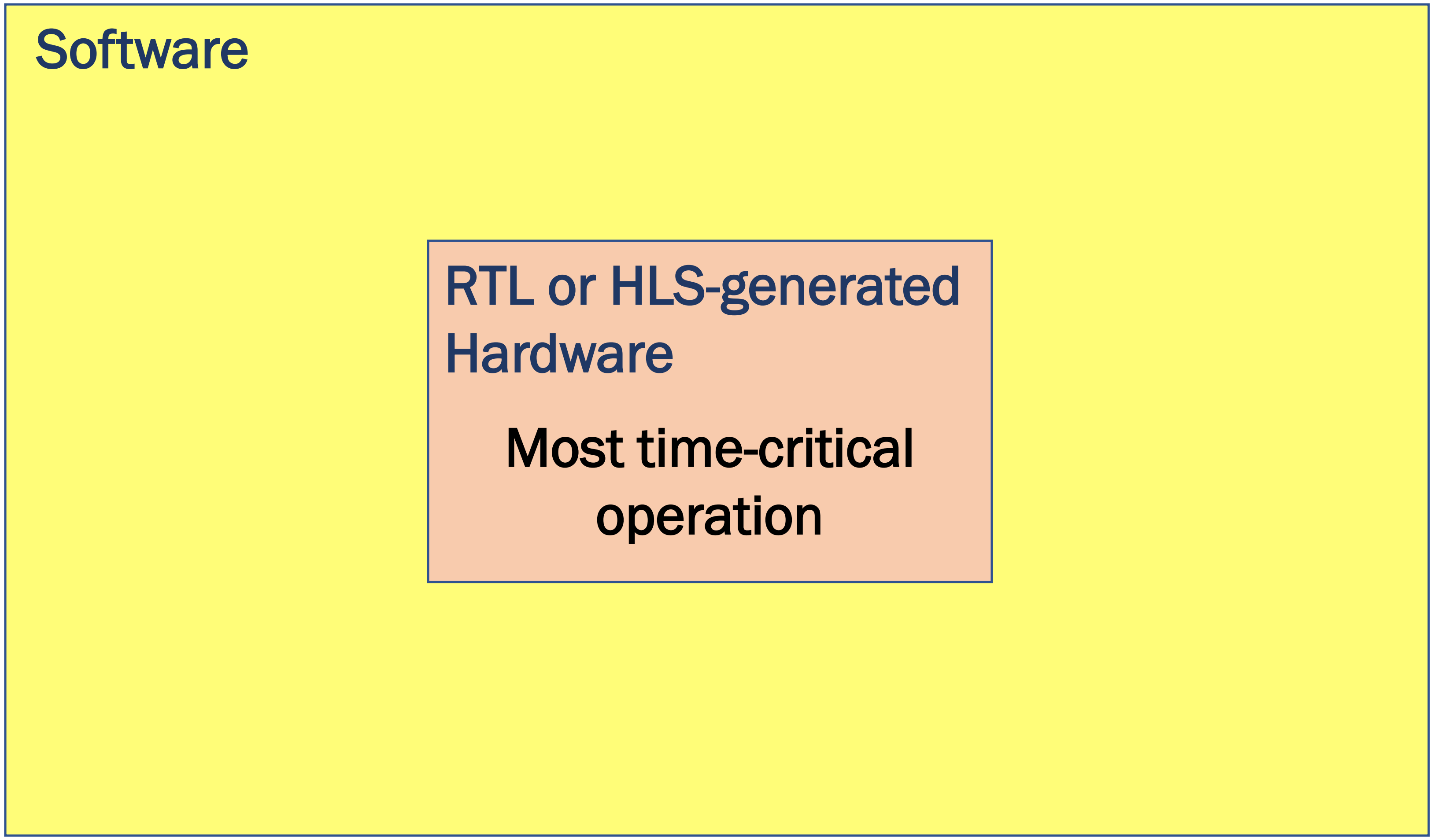
[1]INRIA & LIX, CNRS UMR 7161
École polytechnique, 91128 Palaiseau Cedex, France.

Abstract

We present an efficient key recovery attack on code based encryption schemes using some quasi–dyadic alternant codes with extension degree 2. This attack permits to break the proposal DAGS recently submitted to NIST.

keywords : Code-based Cryptography, McEliece encryption scheme, Key recovery attack, Alternant codes, Quasi–dyadic codes, Schur product of codes.

Software/Hardware Codesign

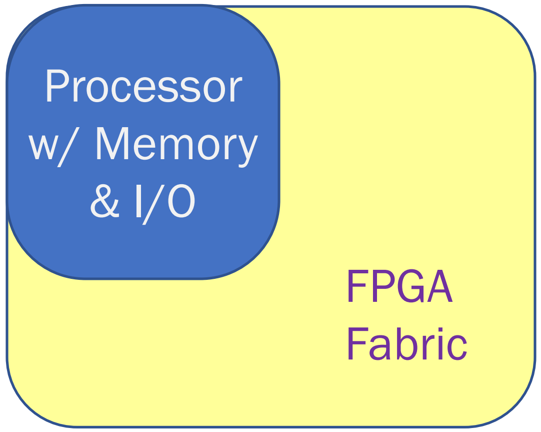


SW/HW Codesign for PQC: Advantages

- Focus on a few (typically 1-3) major operations, known to be easily parallelizable
  - much shorter development time (at least by a factor of 10)
  - guaranteed substantial speed-up
- Insight regarding performance of future instruction set extensions of modern microprocessors
- Possibility of implementing multiple candidates by the same research group, eliminating the influence of different
  - design skills
  - operation subset (e.g., including or excluding key generation)
  - interface & protocol
  - optimization target
  - platform

Two Major Types of Platforms

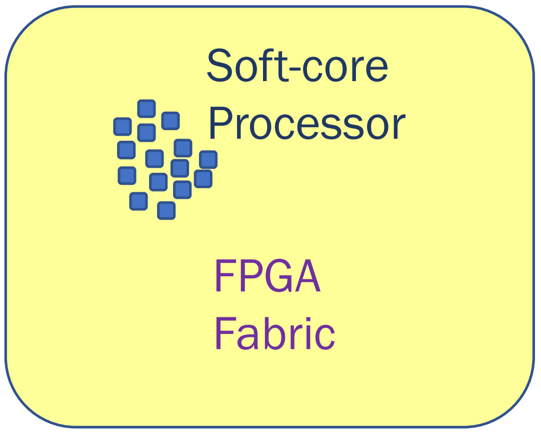
FPGA Fabric & Hard-core Processors



Examples:

- Xilinx Zynq 7000 System on Chip (SoC)
- Xilinx Zynq UltraScale+ MPSoC
- Intel Arria 10 SoC FPGAs
- Intel Stratix 10 SoC FPGAs

FPGA Fabric, including Soft-core Processors



Examples:

Xilinx Virtex UltraScale+ FPGAs
Intel Stratix 10 FPGAs, including

- Xilinx MicroBlaze
- Intel Nios II
- RISC-V, originally UC Berkeley

Selected Platform

FPGA Family:	Xilinx Zynq UltraScale+ MPSoC
Device:	XCZU9EG-2FFVB1156E
Prototyping Board:	ZCU102 Evaluation Kit from Xilinx

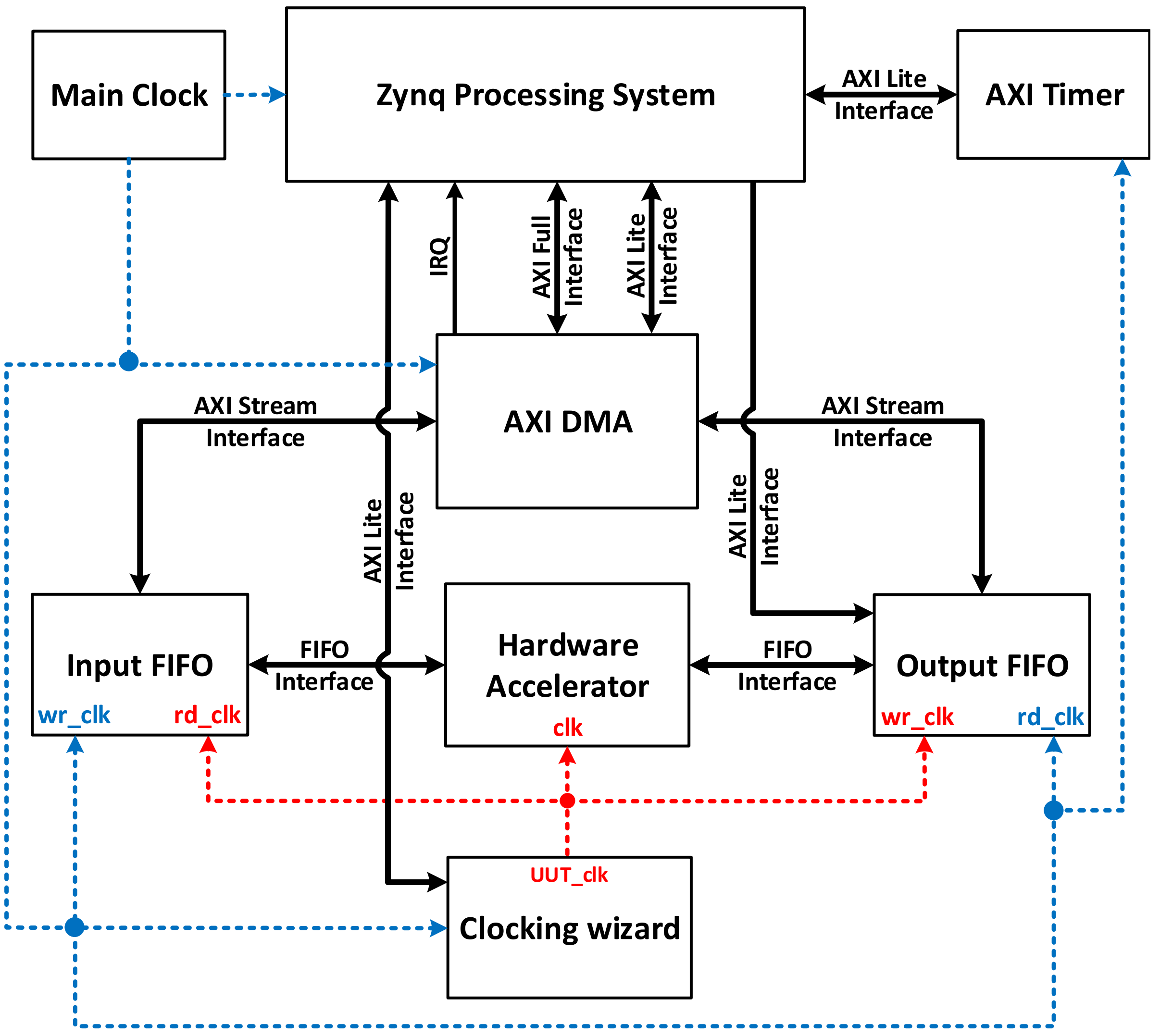
Processing System:

- **Quad-core ARM Cortex-A53 Application Processing Unit,** running at the frequency of 1.2 GHz (only one core used for benchmarking)

Programmable Logic:

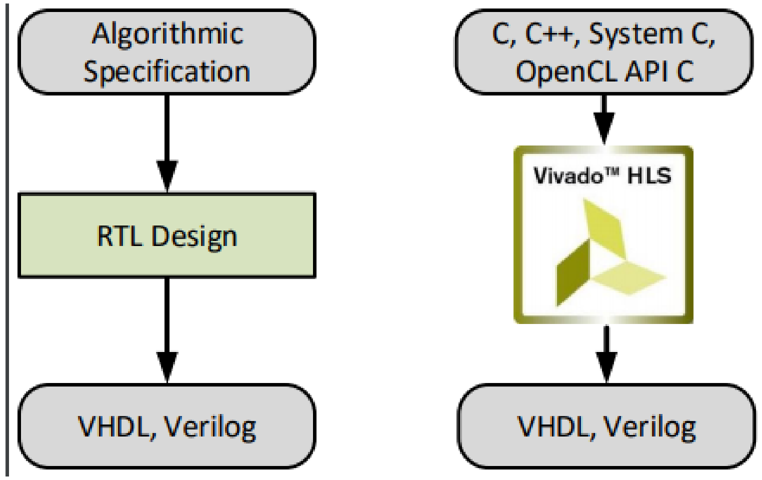
- **Configurable Logic Blocks (CLB), Block RAMs, DSP units**

Experimental Setup



Selected Algorithm

- NTRUEncrypt is one of the most well-known PQC algorithms that has withstood cryptanalysis.
- The speed of NTRUEncrypt in software, especially on embedded software platforms, is limited by the long execution time of polynomial multiplication.
- We implement two variants of the NIST Round 1 PQC candidate NTRUEncrypt: ntru-pke-443 and ntru-pke-743 in bare-metal mode.
- Polynomial multiplication is implemented in the Programmable Logic (PL) of Zynq using two approaches RTL and HLS



Accelerator Design

Target: Minimum Execution Time

- Register-Transfer Level methodology with VHDL
  - ☆ Block diagram of the Datapath and Algorithmic State Machine (ASM) chart of the Controller
- High Level Synthesis methodology with C

  Goal: The same or comparable number of clock cycles as in the Register-Transfer Level (manual) implementation in VHDL

  - ☆ Attempt 1: Reference implementation based on the grade school algorithm for multiplication (a.k.a. schoolbook, paper-and-pencil, etc.)
  - ☆ Attempt 2: Optimized implementation based on rotation
  - ☆ Multiple attempts at optimization using Vivado HLS directives (pragmas) and minor code changes

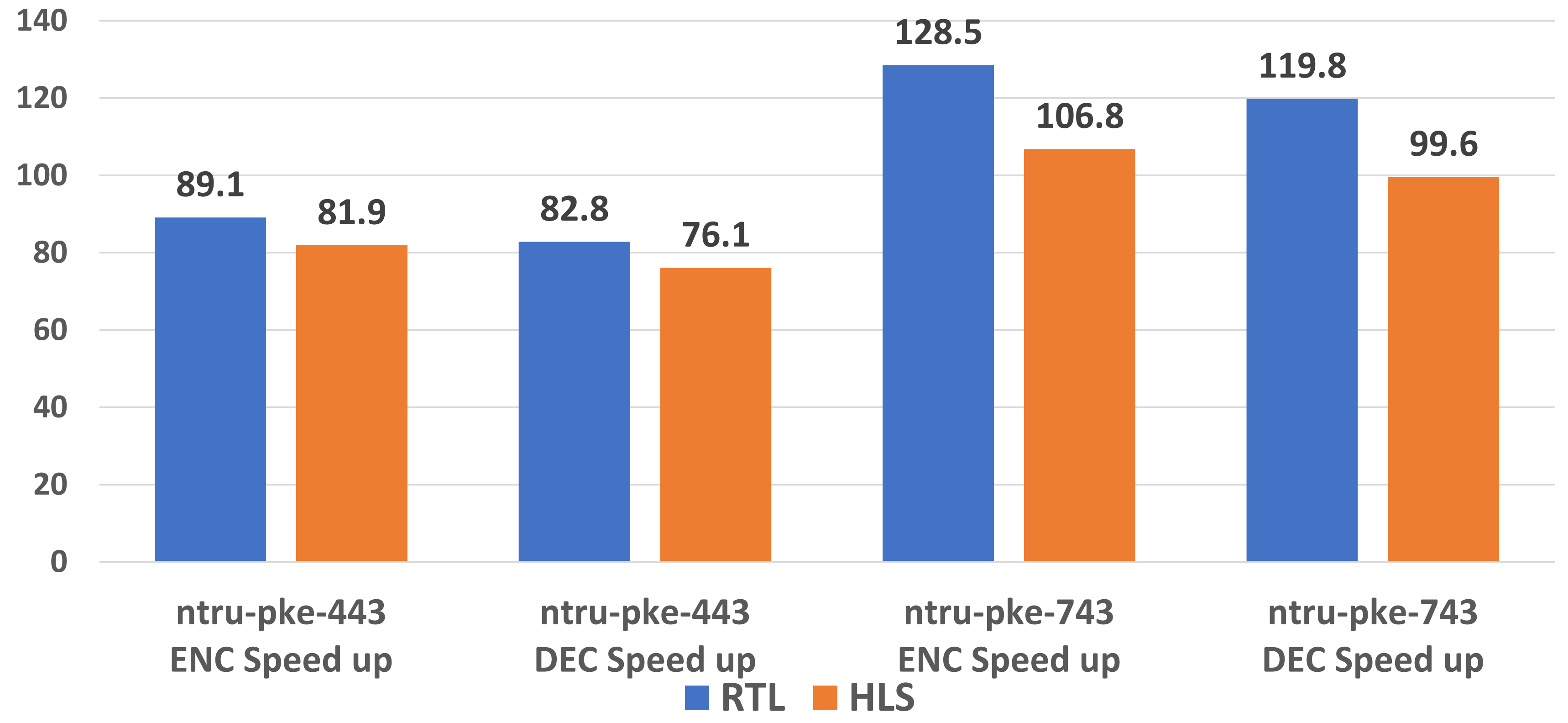
  Outcome 1: **Tens of thousands of clock cycles**, compared to the expected n=743 clock cycles

  Solution: Rewriting the code in C in such a way to match the block diagram used to generate VHDL code

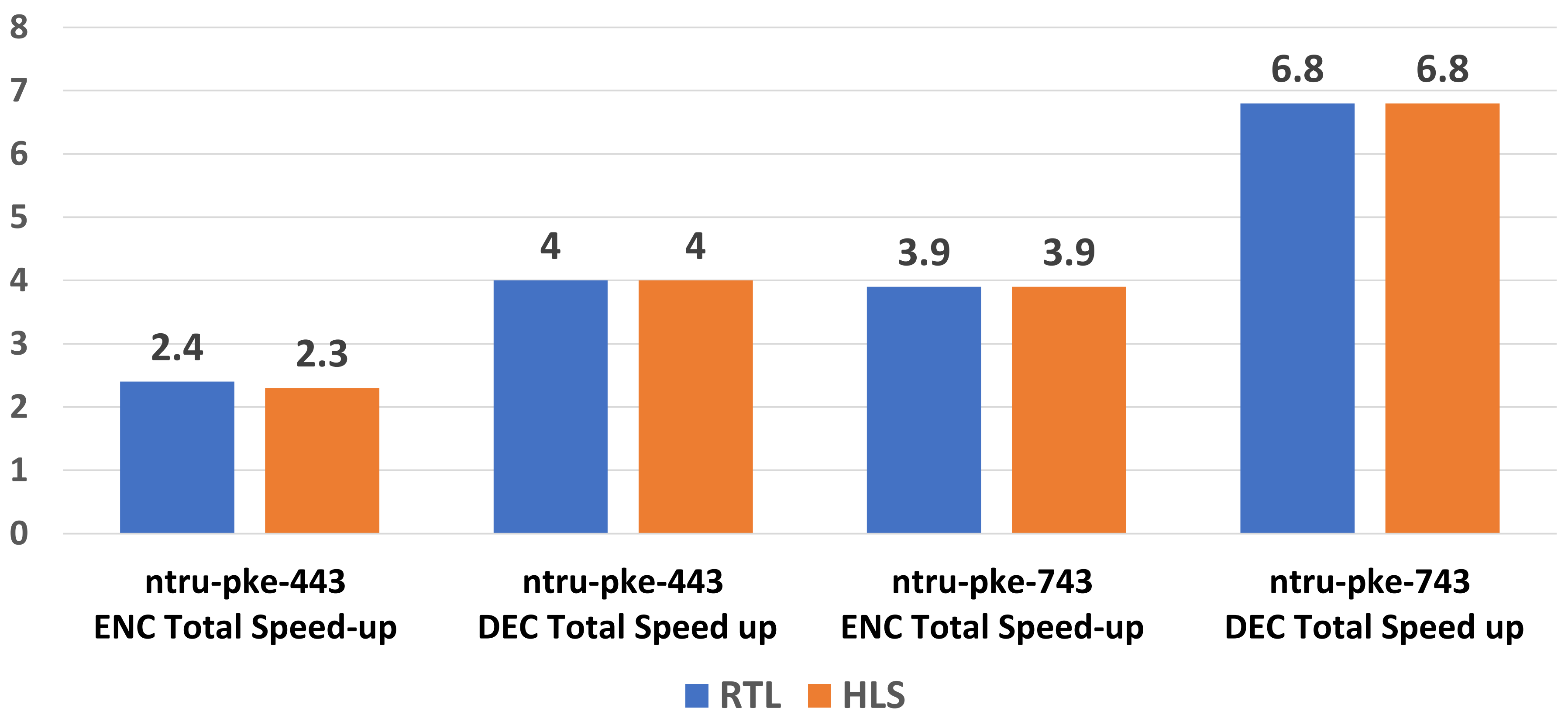
  Outcome 2:

  - ☆ Expected functionality
  - ☆ **Around n clock cycles** of the execution time

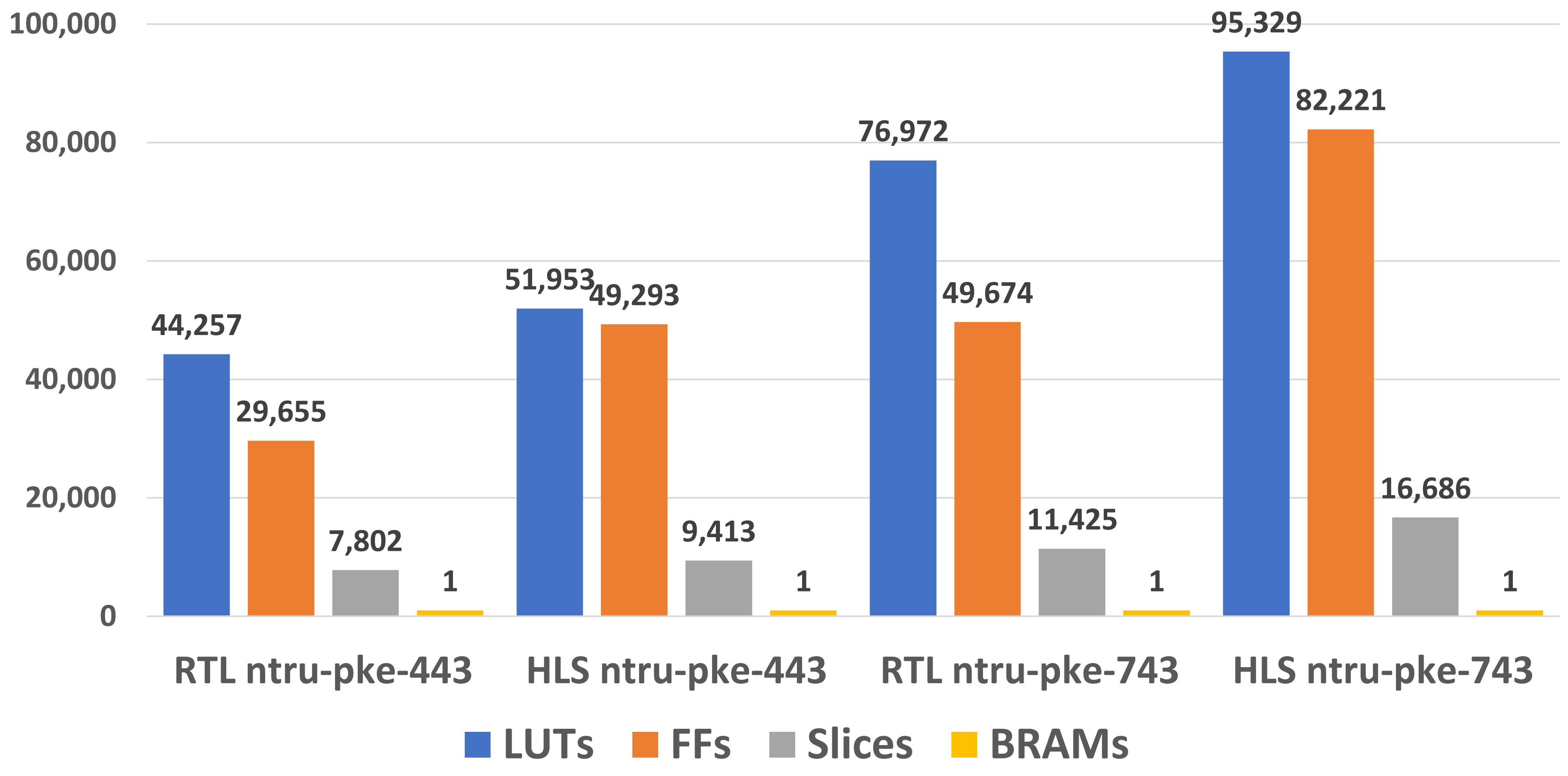
Speed-up achieved for Polynomial Multiplication



Total Speed-up achieved for entire ENC/DEC



Resource Utilization



Q&A

Thank You!

Questions?

Comments?

Suggestions?

CERG: http://cryptography.gmu.edu

ATHENa: http://cryptography.gmu.edu/athena