Low-Latency (i)FFT RTL Implementation for the FALCON Post-Quantum Signature Algorithm

Alexandre Ortega, Lilian Bossuet and Brice Colombier
Université Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France
{alexandre.ortega; lilian.bossuet; b.colombier}@univ-st-etienne.fr

I. Introduction

FALCON [1] is one of the three post-quantum digital signature schemes that have been recently standardized by NIST due to the future threat that quantum computers pose to classical cryptographic schemes [2]. Despite this, there is currently no full hardware register-transfer level (RTL) implementation of FALCON. One possible explanation is the rather unusual requirement for a double-precision floating-point Fast Fourier Transform (FFT) [3], which is used in FALCON to speed up polynomial multiplication. In this work, we propose a full RTL constant-time implementation of the FFT and its inverse (iFFT), on FPGA, tailored for the specific context of FALCON. Section II presents the FFT in the context of FALCON before Section III describes the proposed architecture. Afterwards, the performances of the proposed implementation are detailed and compared with previous works in Section IV. Section V concludes.

II. The Fast Fourier Transform in FALCON

In FALCON, the FFT over the ring $\mathbb{Q}[x]/(\phi)$ is used with $\phi = x^N + 1$ and $N = 2^k$ a power of two. N is a security parameter of FALCON that can be equal to either 512 or 1024. Due to FALCON security requirements, IEEE-754 compliant double-precision floating-point arithmetic is being used [1]. Using the fact that FALCON polynomials are in $\mathbb{Z}[x]/(\phi)$, as well as the roots of unity symmetry in $\mathbb{Z}[x]/(\phi)$, the storage requirements can be halved and more than half of the computations can be omitted [1]. Before the optimizations, the amount of computations to perform is:

$$\#Ops = \log_2(N) \times \frac{N}{2} \tag{1}$$

After applying the optimizations, (1) becomes:

$$\#Ops = (\log_2(N) - 1) \times \frac{N}{4} \tag{2}$$

III. Description of the Proposed Hardware Architecture

Fig. 1 shows the proposed hardware architecture. On top of a reset and clock signals, the input signals are:

- `start` is used to make the component start the computation of either the FFT or the iFFT.
- `inv` is used to choose between performing the FFT or the iFFT.

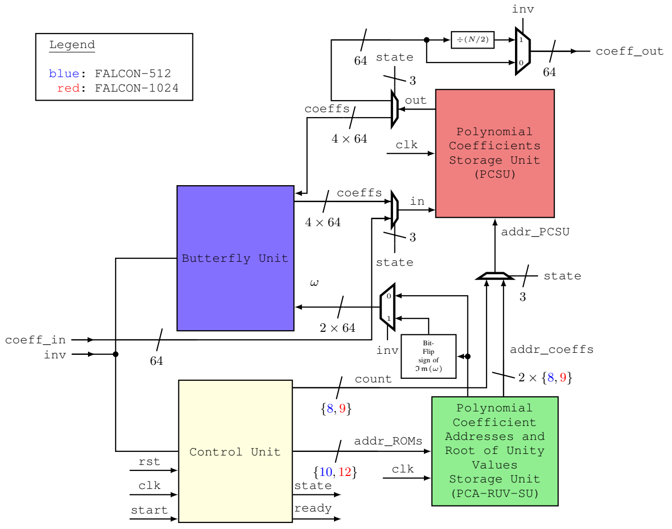

Fig. 1. Block diagram of the proposed hardware architecture of the (i)FFT for FALCON

- `coeff_in` is a 64-bit bus used to stream in the coefficients of the input polynomial to transform.

The outputs are:

- `ready` indicates that the component has finished performing the computations and is streaming out the result.
- `coeff_out` is a 64-bit bus used to stream out the coefficients of the result.

The proposed design is divided in four main blocks.

1) The butterfly unit: Made with three complex double-precision floating-point operators, an adder as well as a subtractor and a multiplier, it can be reconfigured dynamically to perform either the radix-2 decimation-in-time FFT or the radix-2 decimation-in-frequency iFFT.

2) The Polynomial coefficients storage unit: Two true dual port RAMs are used to store the polynomial coefficients. One RAM stores the real parts of the polynomial coefficients and the other RAM stores their imaginary parts. For FALCON-512, polynomials have 512 coefficients. Only the real and imaginary parts for the first half of these coefficients are stored as explained in Section II, so the two RAMs will each store 256 double-precision floating-point values with 8-bit addresses. A double-precision floating-point value is stored on 64 bits (i.e. 8 bytes). Hence, the RAMs will each store

$256 \times 8 = 2.048\,\mathrm{kB}$. An identical reasoning with FALCON-1024 gives a 9-bit address bus and two RAMs each storing $512 \times 8 = 4.096\,\mathrm{kB}$.

3) The Polynomial Coefficient Addresses and Root of Unity Values Storage Unit: Four single-port ROMs and one dual-port ROM are used to store the pre-computed coefficient addresses in RAM and root of unity values. For each butterfly operation, two complex coefficients and one complex root of unity are used. Two single-port ROMs are used to store the pre-computed coefficient addresses in RAM. Using (2), it is determined that 1024 butterfly operations are required for the 512-coefficient (i)FFT, and 2304 for the 1024-coefficient (i)FFT. As the RAMs each store 256 values for the 512-coefficient (i)FFT and 512 values for the 1024-coefficient (i)FFT, an 8-bit wide adress bus, for the two ROMs, is required for the 512-coefficient (i)FFT and an 9-bit wide address bus is required for the 1024-coefficient (i)FFT. This gives a storage requirement of $1024 \times (8/8) = 1024 \times 1 = 1.024\,\mathrm{kB}$ and $2304 \times (9/8) = 2.592\,\mathrm{kB}$ respectively for the 512-coefficient and the 1024-coefficient (i)FFT.

The choice was made to only store once the 64-bit values that can be used for either the real part or the imaginary part in one dual-port ROM and to store the sequence in which those values are used in two single-port ROMs. The reason for that choice was to reduce the amount of memory required to store the values needed for the root of unity. If the root of unity values to be used are stored consecutively in a straightforward manner, which means that repetitions are possible in the dual-port ROM, $1024 \times 8 = 8.192\,\mathrm{kB}$ are needed for the 512-coefficient (i)FFT and $2304 \times 8 = 18.432\,\mathrm{kB}$ for the 1024-coefficient (i)FFT. If the root of unity values are stored only once along with the order in which they are accessed, 382 values need to be stored in the dual-port ROM which corresponds to $382 \times (64/8) = 3.056\,\mathrm{kB}$, and 1024 addresses coded on 9 bits in both single-port ROMs which corresponds to $2 \times 1024 \times (9/8) = 2 \times 1152 = 2.304\,\mathrm{kB}$. This means that this solution requires $3056 + 2304 = 5.36\,\mathrm{kB}$ of storage capacity for the 512-coefficient (i)FFT, which is a reduction of $34.6\,\%$ compared with the straightforward solution. An identical reasoning, gives a storage requirement of $11.888\,\mathrm{kB}$ for the 1024-coefficient (i)FFT, which represents a reduction of $35.5\,\%$ compared with the straightforward solution. Hence, instead of only one dual-port ROM, two single-port and one dual-port ROMs are used to store the root of unity values.

Concerning the root of unity values for the iFFT, their real parts are the same as the root of unity for the FFT and their imaginary parts are of opposite sign. Hence, only the values for the FFT are stored. When performing the iFFT the same values are read from the ROM, but not in the same order. Indeed, to perform the FFT operations, the addresses are read consecutively starting from the highest value. To perform the iFFT operations it is the opposite, the addresses are read consecutively starting from zero. Additionally, the sign bit of the imaginary part of the root of unity value is flipped.

4) The Control Unit: The control unit purpose is to manage the dataflow of the (i)FFT.

TABLE I
(I)FFT-512/(I)FFT-1024 IMPLEMENTATION RESULTS

	This work		[4]	Vivado 2023.2	
Floating-point precision	**Double**		Double	Single	
FFT length	**512**	**1024**	512	512	1024
LUT	**9658**	**9677**	8396	1741	1793
FF	**369**	**374**	2526	3468	3508
DSP	**36**	**36**	9	10	10
BRAM	**8**	**11**	9.5	4	5
Latency (cycles)	**3074**	**6658**	19800	4589	9474

IV. RESULTS AND COMPARISONS WITH PREVIOUS WORKS

The proposed hardware implementation is described in VHDL and synthetised using AMD-Xilinx Vivado 2023.2. Table I reports the implementation results of the proposed design and compares it with the Vivado 2023.2 (i)FFT IP, and a co-design implementation of FALCON (i)FFT for the security parameter $N = 512$ [4]. The FPGA targetted, in all the reported results in Table I, is the AMD-Xilinx ZCU104+ (xczu7ev-ffvc1156-2-e) FPGA. The fairest comparison is with Mandal et al [4]. design as both design use double-precision. Both have similar metrics for the LUTs and the BRAMs. The proposed design uses $4\times$ more DSP blocks but around $6.5\times$ less FFs and clock cycles. As expected when comparing the proposed design to Vivado's IP, which uses single-precision, it uses around $2\times$ more BRAMs, more LUTs and DSPs. However, the proposed design uses around $10\times$ less FFs and achieves a lower latency.

V. CONCLUSION

A low-latency full hardware constant-time RTL implementation of the (i)FFT, tailored for FALCON parameters, was presented. It achieves the best latency of the literature among FPGA-based implementations. This work addresses one of the major difficulties reported concerning the full hardware implementation of FALCON. This work can be used as an essential building block for future hardware implementation works on FALCON.

ACKNOWLEDGMENT

This work received funding from the France 2030 program, managed by the French National Research Agency under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

REFERENCES

[1] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, *et al.*, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, pp. 1–75, 2018.

[2] NIST, "Nist Announces First Four Quantum-Resistant Cryptographic Algorithms," *https://nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms*, 2022.

[3] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-Performance Hardware Implementation of Lattice-Based Digital Signatures." Cryptology ePrint Archive, Paper 2022/217, 2022.

[4] S. Mandal and D. Roy, "Design of a Lightweight Fast Fourier Transformation for FALCON using Hardware-Software Co-Design," in *GLSVLSI'24 Proceedings*, pp. 228–232, 06 2024.