



# Hardware Architectures for Computing Eigendecomposition-Based Discrete Fractional Fourier Transforms with Reduced Arithmetic Complexity

Breno C. Bispo<sup>1</sup> · José R. de Oliveira Neto<sup>2</sup> · Juliano B. Lima<sup>1</sup>

Received: 9 January 2023 / Revised: 14 August 2023 / Accepted: 16 August 2023 /

Published online: 1 September 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

## Abstract

The fractional Fourier transform (FrFT) is a useful mathematical tool for signal and image processing. In some applications, the eigendecomposition-based discrete FrFT (DFrFT) is suitable due to its properties of orthogonality, additivity, reversibility and approximation of continuous FrFT. Although recent studies have introduced reduced arithmetic complexity algorithms for DFrFT computation, which are attractive for real-time and low-power consumption practical scenarios, reliable hardware architectures in this context are gaps in the literature. In this paper, we present two hardware architectures based on the referred algorithms to obtain  $N$ -point DFrFT ( $N = 4L$ ,  $L$  is a positive integer). We validate and compare the performance of such architectures by employing field-programmable gate array implementations, co-designed with an embedded hard processor unit. In particular, we carry out computer experiments where synthesis, error and latency analyses are performed, and consider an application related to compact signal representation.

**Keywords** Field programmable gate array · Discrete fractional Fourier transform · Hardware implementation · Arithmetic complexity

---

J. R. de Oliveira Neto and Juliano B. Lima are contributed equally to this work.

---

✉ Juliano B. Lima  
juliano.lima@ufpe.br

Breno C. Bispo  
breno.bispo@ufpe.br

José R. de Oliveira Neto  
joserodrigues.oliveiraneto@ufpe.br

<sup>1</sup> Department of Electronics and Systems, Federal University of Pernambuco, Recife, Pernambuco, Brazil

<sup>2</sup> Department of Mechanical Engineering, Federal University of Pernambuco, Recife, Pernambuco, Brazil

## 1 Introduction

The fractional Fourier transform (FrFT) corresponds to a generalization of the ordinary Fourier transform, where noninteger powers of the respective operator are admitted. The FrFT was first exploited in quantum mechanics [18] and further used in several application scenarios, such as image encryption, optical signal processing, velocity estimation and vibration on SAR imagery, filtering, biomedical signal and image processing [6, 7, 9, 16, 20, 29]. Hardware FrFT implementations are frequently required in some of the aforementioned scenarios. Such implementations are normally designed from discrete versions of the transform (DFrFT), which should satisfy the following requirements: (i) numerical approximation of the continuous FrFT; (ii) unitarity; (iii) additivity; (iv) reduction to discrete Fourier transform (DFT) when the fractional order equals one. Additionally, it is expected that a DFrFT can be computed using some efficient algorithm [27].

Among the DFrFT approaches documented in the literature, the eigendecomposition-based DFrFT (ED-DFrFT) fulfills all the above listed requirements. On the other hand, the computational complexity of a  $N$ -point ED-DFrFT is  $\mathcal{O}(N^2)$ . The sampling-type DFrFT involves less computational complexity than ED-DFrFT, but they cannot satisfy additivity criterion (iii); this is critical, for example, in applications related to signal recovery, as demonstrated in [27]. Therefore, the scope of this work is restricted to ED-DFrFT implementations, due to the preservation of most requirements of a legitimate DFrFT.

Although subquadratic computational complexity algorithms for computing ED-DFrFT are still a shortcoming in the literature, recent studies promote reduced arithmetic complexity approaches [8, 17]; such a reduction can be achieved, for instance, by choosing suitable HGL DFT (Hermite–Gaussian-like) eigenbases that provide significant number of entries equal (or approximately) to zero in the eigenvectors [8]. In particular, HGL DFT eigenbases generated by closed-form expressions can perform the computation of ED-DFrFT with reduced<sup>1</sup> arithmetic complexity [7, 8, 14, 15]. On the other hand, eigenvectors based on matrices commuting with the DFT matrix operator generated by nonclosed-form procedures are not suitable for this task [3, 4, 10, 19, 25].

The scarce number of publications related to hardware architectures for ED-DFrFT computations does not address the development of dedicated hardware for generalized cases. For example, Sinha et al. [26], Acharya and Mukherjee [1] suggested similar architectures for computing the DFrFT of real-valued input vector. Cariow et al. [5] proposed processing unit structures for small size  $N$ -point DFrFTs ( $N < 8$ ). Although Prasad et al. [21] and Ray et al. [22] propose  $N$ -points DFrFT ( $N$  is odd or even positive integer) and  $2^m$ -points DFrFT ( $m$  is a positive integer) architectures, respectively, they do not exploit properly the error analysis of the proposed architectures and the arithmetic hardware complexity scales up as the number of points increase. Moreover, the method in [22] employs a simplified version of the DFrFT matrix, which may

<sup>1</sup> The term “reduced”, in the title of our paper and throughout its text, refers to the algorithms proposed in [8], which are used as a starting point for the hardware architectures to be presented. Therefore, the use of such a term does not indicate that, in this paper, an algorithm with *even lower arithmetic complexity* is being proposed, compared to the cited reference.

compromise the DFrFT accuracy. Therefore, the current literature still lacks of reliable, low complexity and no constrain hardware architectures for general cases of ED-DFrFT. In this paper, we intend to address this issue by introducing two different architectures for computing the DFrFT of a  $N$ -length complex-valued vector  $\mathbf{x}$ , where  $N = 4L$  and  $L$  is a positive integer, based on recent studies of ED-DFrFT with reduced arithmetic complexity [8]. We consider  $N = 4L$  or  $N = 2^m$  cases,  $m \in \{2, 3, 4, \dots\}$ , due to their practical usage in signal processing. However, without loss of generality, the proposed DFrFT architectures can also cover  $N \neq 4L$  cases with slight hardware changes.

We propose an architecture for DFrFT computation and then consider the implementation of this architecture using two different HGL DFT eigenbases. More precisely, we consider a DFrFT computation method that exploits symmetry and sparsity properties of HGL DFT eigenvectors used to define the DFrFT [8]. The considered HGL DFT eigenbases are those proposed in [3] and [15], which are, respectively, identified as HGL<sub>1</sub> and HGL<sub>2</sub>. Thus, we refer to the considered implementation alternatives by means of the acronyms HGL<sub>1</sub> and HGL<sub>2</sub>.

In this context, it is relevant to emphasize that our work does not constitute a *direct* hardware implementation of existing methods, but involves a series of decisions and architecture choices to achieve the results documented throughout our text. In particular, we confirm that the potential computational benefits suggested in the references we consider have repercussions on a practical implementation, which takes into account not only complexity in terms of numbers of arithmetic operations, but memory /registers, logic elements, percentage consumption of reconfigurable hardware resources, best datapath optimization, etc. Moreover, compared to other papers focused on implementing DFrFT in hardware, our proposal has the smallest footprint, whose value is also constant, regardless of the transform length.

The next sections of this paper are summarized as follows:

- In Sect. 2, we describe the basic fixed-point arithmetic modules and the hardware architecture for computing an eigenvalue used in the proposed architectures;
- In Sect. 3, we describe the HGL<sub>1</sub> and HGL<sub>2</sub> architectures for computing a  $N$ -point DFrFT, for  $N = 4L$  ( $L$  is a positive integer);
- In Sect. 4, we validate the proposed hardware architectures by means of several tests for  $N \in \{8, 16, 32, 64, 128, 256\}$ . We carry out a comparative analysis taking into account software/hardware requirements, evaluating implementation complexity, latency and performance of the architectures when applied to signal compaction. In the analyzed scenarios, we verify that the HGL<sub>2</sub> architecture achieved the best memory usage and speed performance among the proposed architectures.

## 2 Basic Modules

In the implemented architectures, we adopt the 32-bit two's complement fixed-point number representation, where the first and the second most significant bits (MSB) are the signed integer part and the 30 lowest significant bits (LSB) are the fractional part of the number. The resolution is given by  $\delta = 2^{-30}$ . In this manner, a real-

valued number  $y$  represented as a 32-bit fixed-point number is limited by the range  $-2.0 \leq y \leq 2.0 - \delta$  and has its fractional part given by the sum of negative powers of 2. This approach aims to reduce the hardware complexity usually associated with floating-point operations. We evaluate the approximation error propagation of each architecture in Sect. 4.

In what follows, we present the fixed-point arithmetic modules and a module to compute an eigenvalue used in the architectures.

## 2.1 Fixed-Point Arithmetic Modules

We describe the fixed-point arithmetic modules as follows:

- *Adder/subtractor*: the sum/subtraction operation between 32-bit two's complement fixed-point numbers is equivalent to sum/subtraction between 32-bit integers;
- *Fixed-point multiplier and bit shifting*: the multiplier module performs the multiplication of two two's complement 32-bit fixed-point numbers, whose output also gives a two's complement 32-bit fixed-point number. Although the multiplication of two 32-bit numbers can generate a 64-bit number, the multiplier output provides a 32-bit two's complement fixed-point number. Right/left-bit shifting represents multiplication/division by two in fixed-point;
- *RCM (real-complex multiplier) module*: this module consists of two fixed-point multipliers, which perform two real multiplications between a real number represented as a 32-bit fixed-point number,  $x$ , and real/imaginary parts of a complex number represented as two 32-bit fixed-point numbers,  $u$  and  $v$ . Let  $z = x(u + jv)$ ,  $j \triangleq \sqrt{-1}$ , the outputs of the *RCM* module are two 32-bit fixed-point numbers,  $\Re(z) = xu$  and  $\Im(z) = xv$ ;
- *CCM (complex-complex multiplier) module*: this module consists of three fixed-point multipliers, two subtractors and three adders. This module is based on [2], where a low complexity complex multiplication algorithm is described; it performs three real multiplications between sums/subtractions of real/imaginary parts of two complex numbers represented as 32-bit fixed-point numbers,  $x$ ,  $y$ ,  $u$  and  $v$ . Let  $z = (x + jy)(u + jv)$ , the outputs of *CCM* module are two 32-bit fixed-point numbers,  $\Re(z) = (x - y)v + x(u - v)$  and  $\Im(z) = (x - y)v + y(u + v)$ .
- *CORDIC module*: this module consists of a lookup table, adders and shift registers (based on Volder's algorithm [11, 28]). It computes cosine and sine of a fixed-point angle  $\theta$ ,  $0 \leq \theta < \pi/2$  rad. In this work, we use the *CORDIC* module imported from Altera IP core library ALTERA\_CORDIC IP core [12].

## 2.2 Computation of an Eigenvalue $\lambda_n^a$

Let  $a \in \mathbb{R}$ ,  $-2.0 \leq a < 2.0$ , be the fractional order and  $\Lambda^a = [\lambda_0^a, \lambda_1^a, \lambda_2^a, \dots, \lambda_{N-1}^a]^T$  be a column vector formed by the eigenvalues of the corresponding DFrFT matrix (the superscript  $T$  denotes row-column transposition of the

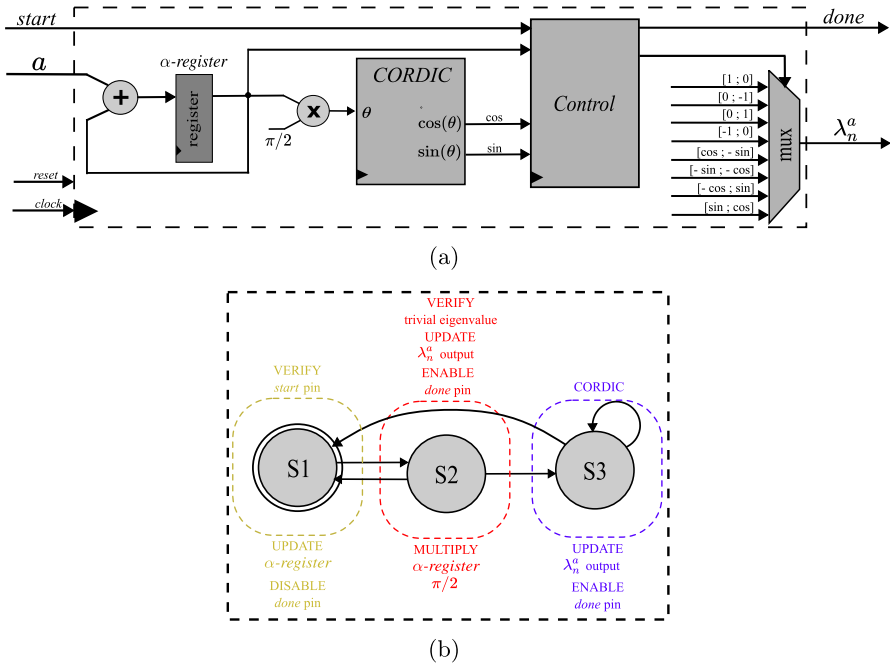


Fig. 1 *Lambda* module that computes an eigenvalue  $\lambda_n^a$ : **a** architecture; **b** FSM

argument). The *Lambda* module computes an eigenvalue

$$\lambda_n^a = e^{-j\frac{\pi}{2}\alpha} = \cos\left(\frac{\pi}{2}\alpha\right) - j \sin\left(\frac{\pi}{2}\alpha\right), \tag{1}$$

$n \in \{0, 1, 2, \dots, N - 1\}$ ,  $\alpha \in \{0, a, 2a, \dots, (N - 2)a, Na\}$  [8], by means of a fixed-point multiplier, one adder, a *CORDIC* module (see Fig. 1a) and a finite state machine (FSM) with 3 states (see Fig. 1b). From this point forward, in order to visually simplify the proposed architectures, in the figures, the labeled arrows containing the same name or symbol are physically connected and the  $[\cdot; \cdot]$  notation means physical bit-array concatenation of two fixed-point numbers. The output data are two fixed-point numbers,  $\Re(\lambda_n^a)$  and  $\Im(\lambda_n^a)$ ,  $n \in \{0, 1, 2, \dots, N - 1\}$ . A controller external to this module loads the input  $a$  and enables/disables the *start* pin. In particular, the following states are specified:

- *State S1*: if *start* pin is enabled, disable *done* pin and accumulate the  $a$  bus value to the  $\alpha$ -register accumulator. After  $(N - 3)$  cycles of the state machine shown in Fig. 1b, the accumulated value will correspond to  $(N - 2)a$ ;
- *State S2*: verify trivial eigenvalue possibilities. If yes, update the  $\lambda_n^a$  bus output according to  $\alpha$ -register value, enable *done* pin and the next state is S1. Otherwise multiply the  $\alpha$ -register value by  $\pi/2$  and the next state is S3;
- *State S3*: after 15 clock cycles, route the *CORDIC* module outputs to  $\lambda_n^a$  bus output. Enable *done* pin and the next state is S1.

### 3 DFrFT Hardware Architectures

In this section, we describe two architectures for computing the  $N$ -point DFrFT ( $N = 4L$ ,  $L$  is a positive integer). We consider the DFrFT computation method proposed in [8] using the HGL eigenbases proposed in [3, 15]: (i) *Architecture HGL<sub>1</sub>*; (ii) *Architecture HGL<sub>2</sub>*, whose memory block structures, data path and FSM of each architecture are presented.

#### 3.1 Architecture HGL<sub>1</sub>

Considering the method proposed in [8], we can compute the DFrFT of a vector  $\mathbf{x} \in \mathbb{C}^N$  through right-to-left matrix multiplications between  $\mathbf{x}$ ,  $\mathbf{V}^T$ ,  $\text{diag}(\Lambda^a)$  and  $\mathbf{V}$ , that is,

$$\mathbf{X}^a = (\mathbf{V} \text{diag}(\Lambda^a) \mathbf{V}^T) \mathbf{x}, \quad (2)$$

with reduced arithmetic complexity by generating intermediate vectors

$$\mathbf{x}' = \mathbf{V}^T \mathbf{x}, \quad (3)$$

$$\mathbf{x}'' = \Lambda^a \circ \mathbf{x}', \quad (4)$$

$$\mathbf{X}^a = \mathbf{V} \mathbf{x}'', \quad (5)$$

where  $\text{diag}(\cdot)$  is a diagonal matrix, whose main diagonal consists of the argument elements [23].

##### 3.1.1 Memory Blocks

In order to synthesize a hardware architecture based on the method described in [8] using the HGL eigenbasis proposed in [3], we design 6 memory blocks aiming to optimize the memory usage, memory access and latency, during the computation of the intermediate vector  $\mathbf{x}'$  and output  $\mathbf{X}^a$ :

- *Two memory blocks store nonzero distinct absolute elements of the matrix  $\mathbf{V}^T$* : (i) the memory block *sram- $\mathbf{V}^T$*  consists of  $(N/2 - 1)N$  memory addresses, where each address stores a fixed-point number,  $\mathbf{V}_{n,i}^T$ ,  $n \in \{0, 1, 2, \dots, N - 1\}$ ,  $i \in \{1, 2, 3, \dots, N/2 - 1\}$ ; and (ii) the memory block *sram- $\mathbf{V}_{0|N/2}^T$*  consists of  $N/2 + 1$  memory addresses, where each address stores two fixed-point numbers,  $\mathbf{V}_{p,0}^T$  and  $\mathbf{V}_{p,N/2}^T$ ,  $p \in \{0, 2, 4, \dots, N - 2, N - 1\}$ ;
- *Two memory blocks store elements of  $\mathbf{x}$  and  $\mathbf{X}^a$* : (i) the memory block *sram- $\mathbf{x}$*  consists of  $N/2$  memory addresses, where the content of the first memory address stores four fixed-point numbers,  $\Re(\mathbf{x}_0)$ ,  $\Im(\mathbf{x}_0)$ ,  $\Re(\mathbf{x}_{N/2})$ ,  $\Im(\mathbf{x}_{N/2})$ , and the remaining addresses store  $\Re(\mathbf{x}_i)$ ,  $\Im(\mathbf{x}_i)$ ,  $\Re(\mathbf{x}_{N-i})$ ,  $\Im(\mathbf{x}_{N-i})$ ,  $i \in \{1, 2, \dots, N/2 - 1\}$ ; and (ii) a memory block with the same structure as *sram- $\mathbf{x}$* , *sram- $\mathbf{X}^a$* , stores the elements of  $\mathbf{X}^a$ , where the content of the first memory address stores four fixed-point numbers,  $\Re(\mathbf{X}_0^a)$ ,  $\Im(\mathbf{X}_0^a)$ ,  $\Re(\mathbf{X}_{N/2}^a)$ ,  $\Im(\mathbf{X}_{N/2}^a)$ , and the remaining addresses store  $\Re(\mathbf{X}_i^a)$ ,  $\Im(\mathbf{X}_i^a)$ ,  $\Re(\mathbf{X}_{N-i}^a)$ ,  $\Im(\mathbf{X}_{N-i}^a)$ ,  $i \in \{1, 2, \dots, N/2 - 1\}$ ;

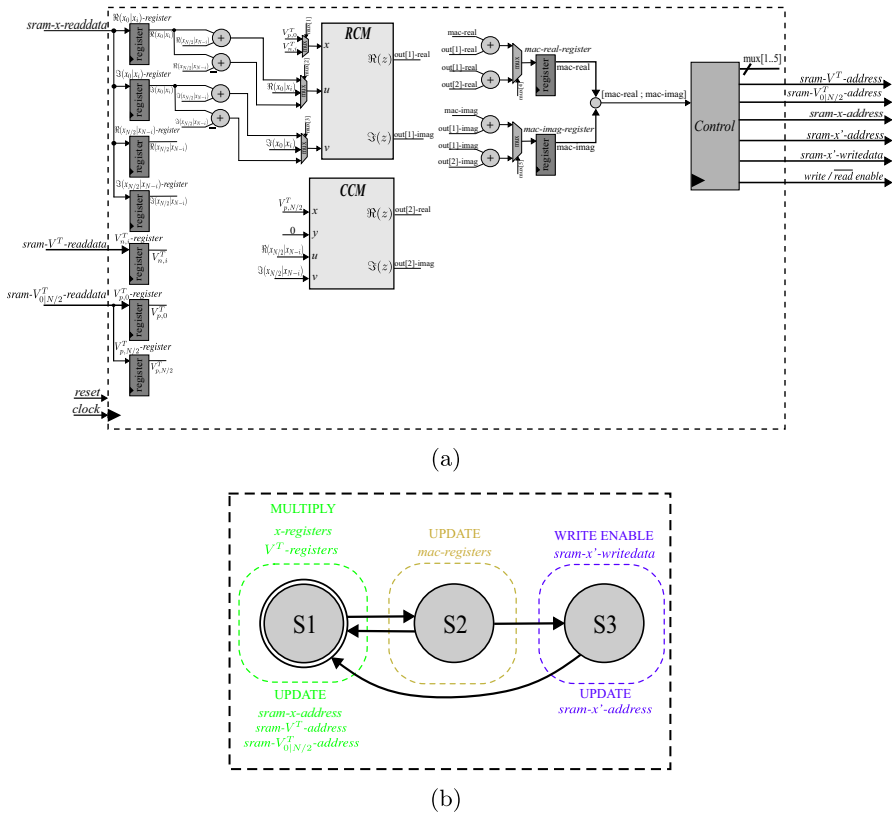


Fig. 2 Module for computing the intermediate vector  $\mathbf{x}'$ : **a** architecture; **b** FSM

- *Two memory blocks store elements of  $\mathbf{x}'$  and  $\mathbf{x}''$* : two memory blocks,  $sram-x'$  and  $sram-x''$ , consist of  $N$  memory addresses, where each memory address stores two fixed-point numbers,  $\Re(\mathbf{x}'_n)$  and  $\Im(\mathbf{x}'_n)$ ,  $\Re(\mathbf{x}''_n)$  and  $\Im(\mathbf{x}''_n)$ ,  $n \in \{0, 1, 2, \dots, N - 1\}$ , respectively.

### 3.1.2 Computation of the Intermediate Vector $\mathbf{x}'$

In this section, we describe the synthesized module that executes the arithmetic operations in (3). For optimization purposes, we take into account the symmetry properties of HGL<sub>1</sub> eigenvectors (rows of matrix  $\mathbf{V}^T$ ). We implement a RCM module, a CCM module and 8 adders, as shown in Fig. 2a. We adopt  $\Re(\cdot | \cdot)$  or  $\Im(\cdot | \cdot)$  notations to label a bus/register that is assigned to either the first or the second argument according to the memory address of the respective memory block structures of each variable as described in Sect. 3.1.1.

A FSM with 3 states, orchestrated by the *Control* module, computes the elements of  $\mathbf{x}'$ , as shown in Fig. 2b. Through the  $sram-V^T$ -address,  $sram-V^T_{0|N/2}$ -address and  $sram-x$ -address output buses, the *Control* module recursively selects the respective memory

block addresses to be read; and they are loaded in the  $sram-V^T$ -readdata,  $sram-V_{0|N/2}^T$ -readdata,  $sram-x$ -readdata input buses according to machine states routines. Moreover, after the recursive computation of each  $\mathbf{x}'$  component, the *Control* module selects the respective memory address,  $sram-x'$ -address, and writes into it through the  $sram-x'$ -writedata bus. In particular, the following states are specified:

- *State S1*: set the  $\overline{\text{write/read enable}}$  pin to low. Update the *RCM* and *CCM* inputs with elements of  $\mathbf{V}^T$  and sum/subtraction of  $\mathbf{x}_0$  or  $\mathbf{x}_{N/2}$ , and  $\mathbf{x}_i$  or  $\mathbf{x}_{N-i}$ ,  $i \in \{1, 2, 3, \dots, N/2 - 1\}$  (according to the current value of  $sram-x'$ -address) stored into their respective registers. Update the  $sram-x$ -address and  $sram-V^T$ -address buses. The next state is *S2*;
- *State S2*: update the  $x$ -registers and  $V^T$ -registers. If the  $sram-x$ -address value is zero, the *mac*-registers accumulators initialize with the sum of *RCM* and *CCM* outputs. Otherwise accumulate the *RCM* outputs in the *mac*-registers. If the *Control* module reads the last  $sram-x$  memory address, then the next state is *S3*. Otherwise the next state is *S1*;
- *State S3*: route the *mac*-registers values to the  $sram-x'$ -writedata buffer of the  $sram-x'$  current address and set the  $\overline{\text{write/read enable}}$  pin to high. Clear the *mac*-registers. If the current value of  $sram-x'$ -address is  $N - 1$ , then the computation of  $\mathbf{x}'$  is finished. Otherwise the next state is *S1*;

### 3.1.3 Computation of the Intermediate Vector $\mathbf{x}''$ and the Elements $X_0^a$ and $X_{N/2}^a$

In this section, we describe the synthesized module that executes the arithmetic operations in (4), where the computation of each element of  $\Lambda^a$  and the element-wise multiplication between the elements of  $\Lambda^a$  and  $\mathbf{x}'$  are performed. Moreover, for latency optimization purpose, this module also computes the elements  $X_0^a$  and  $X_{N/2}^a$  using the elements stored in the memory block  $sram-V_{0|N/2}^T$  described in Sect. 3.1.1. We implement a *Lambda* module, *RCM* and *CCM* modules (shared with the architecture explored in Sect. 3.1.2), 4 adders and a left-bit shift register, as shown in Fig. 3a.

A FSM with 5 states, orchestrated by *Control* module, computes the elements of  $\mathbf{x}''$ ,  $X_0^a$  and  $X_{N/2}^a$ , as shown in Fig. 3b. Through the  $sram-x'$ -address and  $sram-V_{0|N/2}^T$ -address output buses, the *Control* module recursively selects the respective memory block addresses to be read; and they are loaded in the  $sram-x'$ -readdata and  $sram-V_{0|N/2}^T$ -readdata input buses according to machine states routines. Moreover, after the recursive computation of each element of  $\Lambda^a$ , followed by the computation of each element of  $\mathbf{x}''$ , the *Control* module selects the respective memory address,  $sram-x''$ -address, and writes into it through the  $sram-x''$ -writedata bus. Furthermore, the *Control* module selects the first address of the memory block  $sram-X^a$  and write into it through the  $sram-X^a$ -writedata bus. As the selected fractional order  $a$  is constant during the computation of a DFrFT, the value stored in  $a$ -register remains the same. In particular, the following states are specified:

- *State S1*: if the current value of  $sram-x'$ -address is equal to  $N - 1$ , then load  $2a$  to the  $a$  input port of *Lambda* module. Otherwise, load  $a$ . Enable *start* pin. The next state is *S2*;



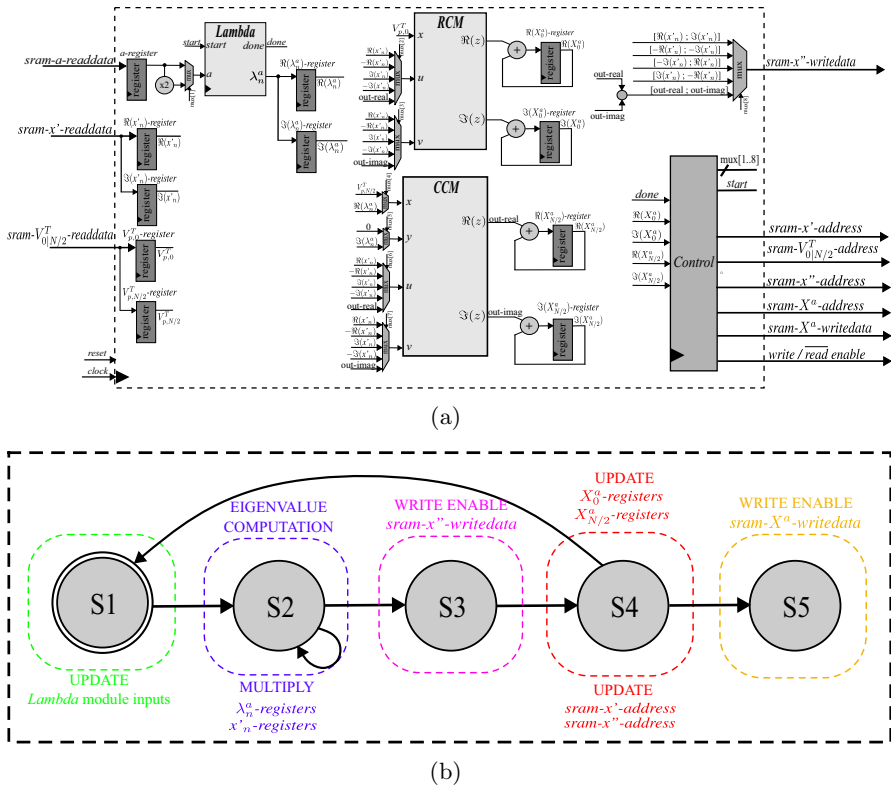


Fig. 3 Module for computing the intermediate vector  $x''$ : **a** architecture; **b** FSM

- State S2: compute an eigenvalue  $\lambda_n^a$  and store it in the  $\lambda_n^a$ -registers. Update the CCM inputs with the values of  $\lambda_n^a$ -registers and  $x'_n$ -registers. The next state is S3;
- State S3: route the values stored in the  $V_{p,0}^T$ -register,  $V_{p,N/2}^T$ -register and CCM outputs to the RCM and CCM inputs. Route the CCM outputs to the  $sram-x''$ -writedata buffer for writing into the current address  $sram-x''$ -address and set the write/read enable pin to high. The next state is S4;
- State S4: set the write/read enable pin to low. Accumulate the RCM and CCM output values in the  $X_0^a$ -registers and  $X_{N/2}^a$ -registers. Update  $sram-x''$ -address and  $sram-x''$ -address. If the current value of  $sram-x''$ -address is  $N - 1$ , then the computation of  $x''$  is finished and the next state is S5. Otherwise the next state is S1;
- State S5: route the  $X_0^a$ -registers and  $X_{N/2}^a$ -registers values to the  $sram-X^a$ -writedata buffer for writing into the first address of  $sram-X^a$  and set the write/read enable pin to high.

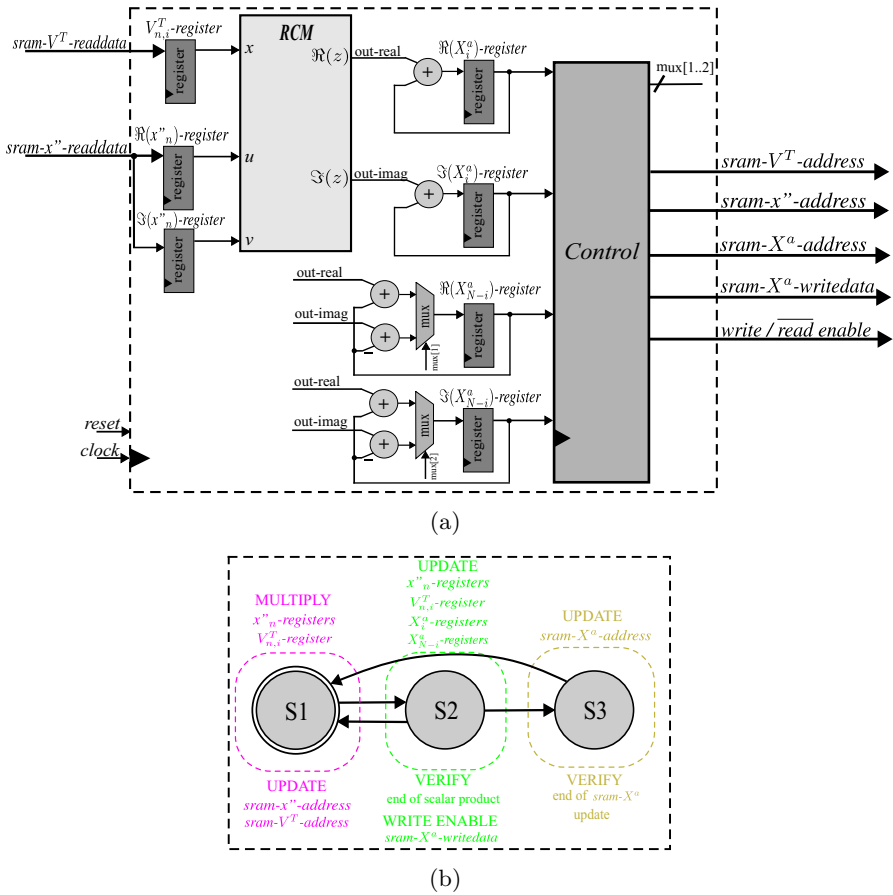


Fig. 4 Module for computing the  $X^a$ : **a** architecture; **b** FSM

### 3.1.4 Computation of Vector $X^a$

In this section, we describe the synthesized module that executes the arithmetic operations in (5), except for the elements  $X_0^a$  and  $X_{N/2}^a$ , which are computed in the module described in Sect. 3.1.3. Thus, this module recursively and simultaneously computes each element of  $X_i^a$  and  $X_{N-i}^a$ ,  $i \in \{1, 2, \dots, N/2 - 1\}$ . We implement 6 adders and a  $RCM$  module (shared with the architectures explored in Sects. 3.1.2 and 3.1.3) for arithmetic operations in (5) regarding the computation of  $X^a$ , as shown in Fig. 4a.

A FSM with 3 states, orchestrated by the  $Control$  module, computes the remaining elements of  $X^a$ , as shown in Fig. 4b. Through the  $sram-V^T$ -address and  $sram-x^a$ -address output buses, the  $Control$  module selects recursively the respective memory block addresses to be read; and they are loaded in the  $sram-V^T$ -readdata,  $sram-x^a$ -readdata input buses according to machine states routines. Moreover, after the recursive computation of each  $X^a$  component, the  $Control$  module selects

the respective memory address,  $sram-X^a$ -address, and writes into it through the  $sram-X^a$ -writedata bus. In particular, the following states are specified:

- **State S1:** update the RCM inputs with the values of  $x''_n$ -registers and  $V_{n,i}^T$ -register. Update the addresses  $sram-x''$ -address and  $sram-V^T$ -address. The next state is S2;
- **State S2:** if the current value of  $sram-x''$ -address is equal to  $N - 1$ , then route the values of  $X_i^a$ -registers and  $X_{N-i}^a$ -registers to the  $sram-X^a$ -writedata bus for writing into the current memory address  $sram-X^a$ -address, set the write/read enable pin to high and the next state is S3. Otherwise accumulate the values provided in the RCM outputs positively or negatively (for odd or even values of  $sram-x''$ -address, respectively) to the  $X_i^a$ -registers and  $X_{N-i}^a$ -registers and the next state is S1;
- **State S3:** set the write/read enable pin to low. If  $sram-X^a$ -address value is equal to  $N/2 - 1$ , then the computation of  $\mathbf{X}^a$  is finished. Otherwise clear the accumulators  $X_i^a$ -registers and  $X_{N-i}^a$ -registers, update the  $sram-X^a$ -address and the next state is S1.

### 3.2 Architecture HGL<sub>2</sub>

The architecture proposed in this section is also based on the DFrFT computation method given in [8] method, which employs intermediate vectors. Therefore, it is similar to the architecture described in Sect. 3.1. However, in the current case, the eigenbasis proposed in [15] is used. The changes resulting from this choice are detailed below.

#### 3.2.1 Memory Blocks

In order to minimize the memory usage and the memory access during the DFrFT computation, we design three memory blocks that store nonzero distinct absolute elements of  $\mathbf{V}^T$ : (i) the memory block  $sram-V^T$  stores nonzero distinct absolute elements of  $\mathbf{V}_{m,i}^T$ ,  $m \in \{0, 1, 2, \dots, N - 3\}$ ,  $i \in \{1, 2, 3, \dots, N/2 - 1\}$ , where each address stores a fixed-point number; (ii) the memory block  $sram-V_{N/2}^T$  consists of  $N/2 - 1$  memory addresses, where each memory address stores a fixed-point number,  $\mathbf{V}_{q,N/2}^T$ ,  $q \in \{0, 2, 4, \dots, N - 4\}$ ; and (iii) the memory block  $sram-V_{\Phi_{N-2}|\Phi_{N-1}}^T$  stores four fixed-point numbers,  $\mathbf{V}_{N-2,0}^T$ ,  $\mathbf{V}_{N-1,0}^T$ ,  $\mathbf{V}_{N-2,1}^T$  and  $\mathbf{V}_{N-1,2}^T$ , that represent the distinct absolute elements of the eigenvectors  $\Phi_{N-2}$  and  $\Phi_{N-1}$  [8].

Furthermore, we implement the same memory block structures described in Sect. 3.1 for  $\mathbf{x}$ ,  $\mathbf{x}'$ ,  $\mathbf{x}''$  and  $\mathbf{X}^a$  storage, but with slight changes of  $\mathbf{x}$  and  $\mathbf{X}^a$  indexes as described in the following steps.

#### 3.2.2 Computation of the Intermediate Vectors $\mathbf{x}'$

The computation of the intermediate vector  $\mathbf{x}'$  employs an architecture and FSM similar to those explored in Sect. 3.1.2, where the elements  $\mathbf{x}_0$ ,  $\mathbf{x}_{N/2}$  become  $\mathbf{x}_{N/2-1}$ ,

$\mathbf{x}_{N-1}$ , respectively, and  $\mathbf{x}_i$ ,  $\mathbf{x}_{N-i}$ ,  $i \in \{1, 2, 3, \dots, N/2 - 1\}$ , become  $\mathbf{x}_i$ ,  $\mathbf{x}_{N-2-i}$ ,  $i \in \{0, 1, 2, \dots, N/2 - 2\}$ , respectively.

Moreover, we increment two accumulators and a second *CCM* module for a fast computation of  $\mathbf{x}'_{N-2}$  and  $\mathbf{x}'_{N-1}$ . Due to repeated components of eigenvectors  $\Phi_{N-2}$  and  $\Phi_{N-1}$  [8], we can compute  $\mathbf{x}'_{N-2}$  and  $\mathbf{x}'_{N-1}$  through positive/negative accumulation of the  $\mathbf{x}$  elements and four multiplications.

### 3.2.3 Computation of the Intermediate Vector $\mathbf{x}''$ and the Elements $\mathbf{X}_{N/2-1}^a$ and $\mathbf{X}_{N-1}^a$

The computation of the intermediate vector  $\mathbf{x}''$  follows the same architecture and FSM explored in Sect. 3.1.3. However, the elements  $\mathbf{X}_0^a$ ,  $\mathbf{X}_{N/2}^a$  become  $\mathbf{X}_{N/2-1}^a$ ,  $\mathbf{X}_{N-1}^a$ , respectively. Using two *CCM* modules, we recursively compute the elements of  $\mathbf{x}''$ ,  $\mathbf{X}_{N/2-1}^a$  and  $\mathbf{X}_{N-1}^a$ .

### 3.2.4 Computation of $\mathbf{X}^a$

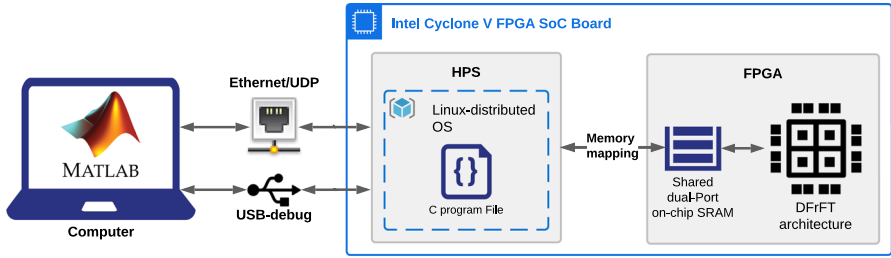
The computation of  $\mathbf{X}^a$  employs an architecture and a FSM similar to those explored in Sect. 3.1.4. However, the elements  $\mathbf{X}_i^a$ ,  $\mathbf{X}_{N-i}^a$ ,  $i \in \{1, 2, 3, \dots, N/2 - 1\}$ , become  $\mathbf{X}_i^a$ ,  $\mathbf{X}_{N-2-i}^a$ ,  $i \in \{0, 1, 2, \dots, N/2 - 2\}$ , respectively.

## 4 Implementation and Results

We design the hardware architectures in Verilog on the Intel Cyclone V field-programmable gate array (FPGA) SoC 5CSEMA5F31C6 [13], using Quartus Prime 18.1 Standard Edition. The FPGA has 32,070 logic elements and 4,065,280 bits of on-chip SRAM. The 5CSEMA5F31C6 chip also includes a hard processor unit (HPS), represented by a Dual-Core ARM Cortex-A9 processor. A Linux-distributed operational system (OS) runs in the HPS, where programs (written in C programming language) establish Ethernet/UDP communication between computer and HPS. The HPS also exchanges data with the FPGA architectures through a shared dual-port on-chip SRAM.

In particular, for test purposes of each architecture, the HPS stores once into the shared on-chip SRAM the elements of the matrix  $\mathbf{V}^T$  of eigenvectors through memory mapping; it receives the complex-valued vector  $\mathbf{x}$  and fractional order  $a$  from the computer; performs vector normalization and floating-point to fixed-point conversions; stores  $a$  and  $\mathbf{x}$  into the shared on-chip SRAM; and enables the DFrFT computation of the respective values of  $a$  and  $\mathbf{x}$ . Later, the HPS performs fixed-point to floating-point conversions; sends the output vector  $\mathbf{X}^a$  and FPGA latency (in clock cycles) to the computer. Additionally, in the application related to compact representation in fractional Fourier domain (see details in Sect. 4.4), the HPS also executes an optimum fractional order searching algorithm; sends the optimum fractional order, measurement of compactness and searching time delay (in seconds) to the computer.

In this way, we can validate the proposed architectures using a high level programming language. In this work, we use the MATLAB to perform such a task. Figure 5



**Fig. 5** Hardware setup for testing the implemented designs and communication between computer, HPS and FPGA

**Table 1** Synthesis results for the proposed DFrFT architectures

Architecture	Number of components			
	LE	Registers	Adder/subtractor	Multiplier
HGL <sub>1</sub>	7848 (24%)	6695	42	6
HGL <sub>2</sub>	9131 (28%)	7773	68	9

**Table 2** Memory usage (bits) for the proposed DFrFT architectures

<i>N</i>	Required SRAM (bits)	
	HGL <sub>1</sub>	HGL <sub>2</sub>
8	3392 (0.1%)	3296 (0.1%)
16	8512 (0.2%)	7520 (0.2%)
32	24,896 (0.6%)	20,576 (0.5%)
64	82,240 (2.0%)	65,120 (1.6%)
128	295,232 (7.3%)	227,936 (5.6%)
256	1,114,432 (27.4%)	848,480 (20.9%)

illustrates the hardware setup designed to validate the architectures and applications. The computer USB terminal performs debugging procedures and sends commands to the HPS.

### 4.1 Synthesis Results

In Table 1, the synthesis results of the architectures described in Sect. 3 are presented; it provides the number of logical elements (LE), number of registers, number of 32-bit Adders/Subtractors and 32-bit fixed-point multipliers used in each architecture. We observe that the HGL<sub>2</sub>-based architecture requires a larger number of hardware elements than the HGL<sub>1</sub>-based architecture. This is in line with the fact that, due to the distinct symmetry and outperforming sparsity of HGL<sub>2</sub> eigenbasis compared to HGL<sub>1</sub> eigenbasis [8], we enhance the memory access and parallel arithmetic operations of HGL<sub>2</sub>-based architecture aiming to optimize the latency performance.

Table 2 provides the number of memory bits used by the architectures in the Cyclone V and their usage percentages (in parentheses) provided by Quartus for

$N = 8, 16, 32, 64, 128, 256$ . We remark that the HGL<sub>2</sub>-based architecture requires smaller memory allocation than the HGL<sub>1</sub>-based architecture due to the smaller number of nonzero distinct absolute elements of HGL<sub>2</sub> eigenbasis matrix  $\mathbf{V}^T$  stored in SRAM and required to compute a DFrFT.

For test purposes, we generate in MATLAB a set of 1000 random input vectors  $\mathbf{x} \in \mathbb{C}^N$ , where each element of  $\mathbf{x}$ ,  $\mathbf{x}_n$ , is bounded by  $-1.0 \leq \Re(\mathbf{x}_n) \leq 1.0$  and  $-1.0 \leq \Im(\mathbf{x}_n) \leq 1.0$ , and a set of 1000 random fractional orders  $a$ ,  $-2.0 \leq a < 2.0$ . In order to compare the architectures in terms of accuracy, we compute 1000  $N$ -DFrFTs of  $\mathbf{x}$  for  $N \in \{8, 16, 32, 64, 128, 256\}$  in MATLAB with double-precision floating-point number representation, using the HGL<sub>1</sub> and HGL<sub>2</sub> eigenbases, and consider the obtained results as a reference.

On the HPS side, a program stores once nonzero distinct absolute elements of eigenvectors matrix  $\mathbf{V}^T$  in 32-bit fixed-point format into the shared SRAM memory blocks, according to the DFrFT architecture. For each DFrFT computation, the computer sends to the HPS a respective input vector  $\mathbf{x}$  and a fractional order  $a$  in double-precision floating-point number format.

Before each DFrFT computation procedure, the HPS normalizes each input vector  $\mathbf{x}$ , converts the 64-bit floating-point elements of  $a$  and  $\mathbf{x}$  to 32-bit fixed-point numbers, stores into the *sram-x* memory block and enables the DFrFT computation according to the respective DFrFT architecture. The normalization approach avoids fixed-point overflows during arithmetic operations performed by the FPGA.

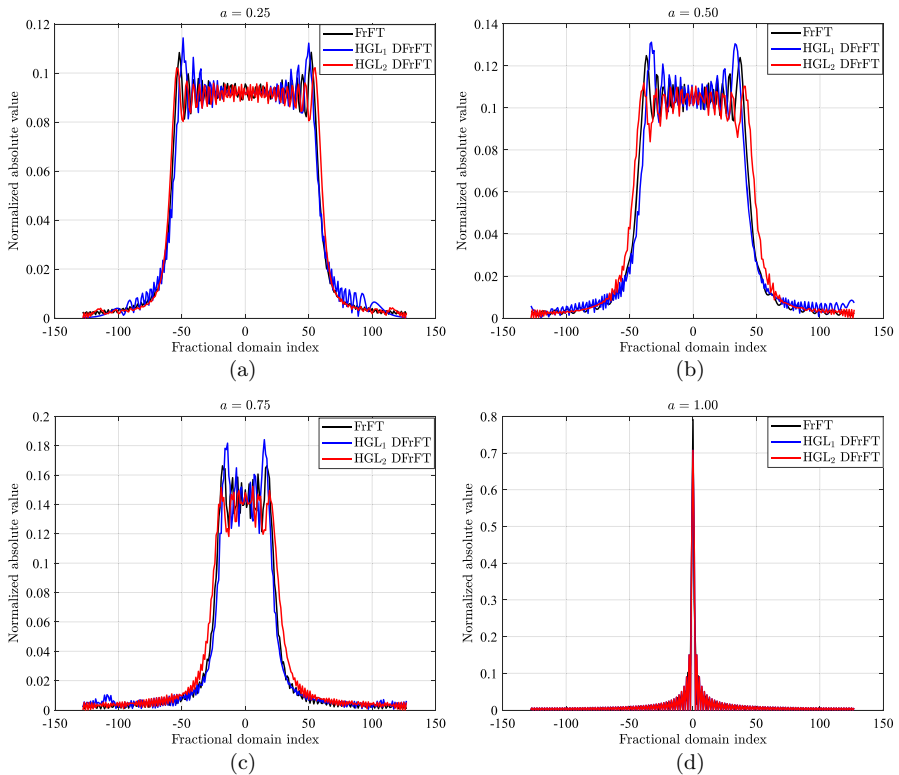
The HPS reads the *sram-X<sup>a</sup>* memory block and converts the  $\mathbf{X}^a$  elements (computed by the FPGA running at a clock frequency of 100 MHz) from fixed to floating-point and multiplies the  $\mathbf{X}^a$  by the norm of the input vector  $\mathbf{x}$ . The HPS sends the DFrFT of a respective input vector  $\mathbf{x}$  and latency in clock cycles (counted by the FPGA) to the computer. The computer then performs the latency and accuracy analysis.

In Table 3, the root mean squared error (RMSE) and latency (in clock cycles) of each DFrFT architecture computed by the HPS-FPGA are shown. The DFrFT latency of integer fractional orders  $a = 0, 1, -1, -2$  is shown in parentheses. We observe that the HGL<sub>2</sub>-based architecture achieved better accuracy and latency than HGL<sub>1</sub>-based architecture to compute a  $N$ -point DFrFT. We suggest that this fact is caused by the lowest required number of arithmetic operations to calculate a DFrFT [8].

During this research, we also synthesized a DFrFT computation architecture based on the method proposed in [17] using the HGL<sub>1</sub> and HGL<sub>2</sub> eigenbases. We observed that such an architecture requires a larger number of hardware elements and is significantly slower than the proposed architecture. This is in line with the fact that, according to the method given in [17], the  $N \times N$  DFrFT matrix can be decomposed into two (for an odd  $N$ ) or three (for an even  $N$ ) matrices. Then, the DFrFT of an input vector can be written as sums of low complexity matrix products. However, the elements of these complex-valued matrices need to be computed. Therefore, the hardware requirements and timing performance for the DFrFT computation are compromised.

**Table 3** RMSE and latency of each proposed DFrFT architecture

$N$	RMSE		Latency (clock cycles)	
	HGL <sub>1</sub>	HGL <sub>2</sub>	HGL <sub>1</sub>	HGL <sub>2</sub>
8	$(1.46 \pm 0.23) \cdot 10^{-8}$	$(1.02 \pm 0.18) \cdot 10^{-8}$	269 (143)	235 (111)
16	$(3.60 \pm 0.41) \cdot 10^{-8}$	$(2.51 \pm 0.27) \cdot 10^{-8}$	809 (539)	687 (419)
32	$(9.37 \pm 0.75) \cdot 10^{-8}$	$(6.86 \pm 0.46) \cdot 10^{-8}$	2657 (2,099)	2167 (1611)
64	$(2.22 \pm 0.15) \cdot 10^{-7}$	$(1.92 \pm 0.08) \cdot 10^{-7}$	9425 (8,291)	7431 (6299)
128	$(5.81 \pm 0.24) \cdot 10^{-7}$	$(5.37 \pm 0.16) \cdot 10^{-7}$	35,249 (32,963)	27,175 (24,891)
256	$(1.49 \pm 0.04) \cdot 10^{-6}$	$(1.50 \pm 0.03) \cdot 10^{-6}$	136,049 (131,459)	103,527 (98,939)



**Fig. 6** Comparison of normalized absolute values of continuous FrFT (computed in MATLAB), HGL<sub>1</sub> and HGL<sub>2</sub> architectures (computed by HPS-FPGA) for  $N = 256$  points: **a**  $a = 0.25$ ; **b**  $a = 0.50$ ; **c**  $a = 0.75$ ; **d**  $a = 1.00$

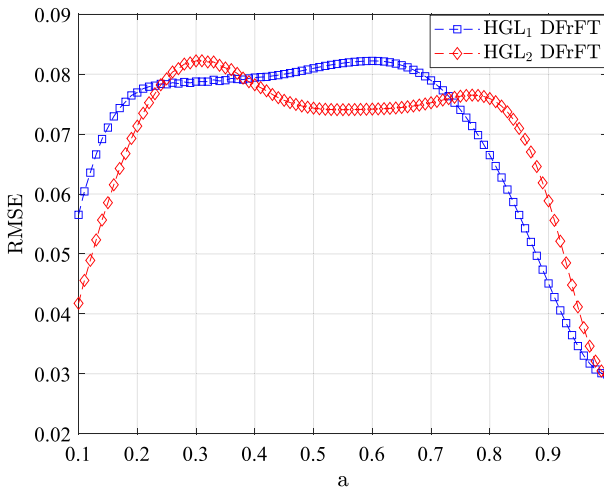
## 4.2 Comparative Analysis

In order to compare the results obtained by the proposed DFrFT architectures, we also compute in MATLAB the continuous FrFT of the rectangular pulse [7, 18]. In Fig. 6, we use  $N = 256$  and plot the DFrFT magnitudes (computed by the HPS-FPGA) and sampled versions of continuous FrFT magnitudes of the rectangular pulse. Inspecting Fig. 6a–c, one can observe that the graphs have similar visual aspects. In Fig. 6d, the graphs overlap each other, assuming, in all cases, the aspect of a sync function.

In order to evaluate the accuracy of DFrFTs compared to continuous FrFT of the rectangular pulse, we vary the fractional order  $a$  at a step of 0.01 and compute the RMSE. The result is shown in Fig. 7, where we observe no significant differences between the DFrFTs computed by the architectures.

Due to similar odd/even symmetry patterns of the matrices  $\mathbf{V}^T$  of eigenvectors, verified in [3, 15], the strategies described in Sect. 3 are fully applicable to DFrFT for  $N \neq 4L$ ,  $L$  is a positive integer. We just have to modify the variable indexes of the memory blocks and FSM to adapt the memory accesses based on strategies mentioned in [8].





**Fig. 7** RMSE of HGL<sub>1</sub> and HGL<sub>2</sub>-based DFrFTs compared to continuous FrFT for  $N = 256$

**Table 4** Comparison of hardware complexity and latency of the proposed architectures with the existing architectures for 256-point DFrFT

Hardware requirement	Number of components		HGL <sub>1</sub>	HGL <sub>2</sub>
	[21]	[22]		
16-bit Multiplier	1029	516	0	0
32-bit Multiplier	0	0	6	9
32-bit Adder/subtrator	1072	606	42	68
Memory usage (bits)	2,097,152	33,024	1,114,432	848,480
1-bit Register	≈ 90,400	≈ 102,614	6695	7773
Latency (clock cycles)	516	667	1,114,432	848,480

### 4.3 Comparison with Related Works

Except the implementations presented in [5], which covers small size DFrFTs ( $N < 8$ ), and [1, 26], which are discussed in [22] and cover DFrFTs of real-valued input vectors, we can compare the proposed architectures with approaches given in [21, 22]. The hardware complexity and latency details of the proposed and existing architectures [21, 22] are presented in Table 4 for 256-point DFrFT case. Other cases covered in this paper can be compared by means of hardware requirement formulas presented in [21, 22]. The arithmetic hardware complexity (number of adders/subtractors and multipliers) of our architectures is constant and significantly lower (at least, a 96% and 88% reduction of multipliers and adders/subtractors, respectively, for 256-point DFrFT computation) than the increasing complexity of [21, 22] architectures (limited to  $\mathcal{O}(4N)$  and  $\mathcal{O}(2N)$ , respectively). Although the memory usage of our architectures is higher than [22] and lower than [21] architectures, the number of registers is significantly lower than [21, 22] methods. Due to pipelined approaches proposed in [21, 22], they achieved higher

throughput. However, the existing ED-DFrFT architectures in the literature do not provide detailed error analysis (similar to those discussed in Sects. 4.1 and 4.2) to compare the accuracy of their methods with ours regarding a paramount feature of a DFrFT, the numerical approximation of the continuous FrFT. They also do not carry out tests with distinct values of  $N$ , complex-valued input vector  $\mathbf{x}$  and fractional order  $\alpha$ . In view of the above, a complete comparison, in terms of accuracy, between our proposal and those presented in the cited papers is not feasible, considering the scope of the current paper.

In summary, comparing our architectures with similar existing architectures (i.e., no input constrains nor limited values of  $N$ ) in the literature [21, 22] regarding the hardware requirements, we conclude that the appropriate architecture depends on the application priority. If timing optimization is required, then [22] reaches the best latency. Otherwise, if a circuit area minimal solution is required, our architectures are more suitable.

We can also compare our architectures with that proposed in [30]. In this context, it is relevant to point out that, in the cited paper, a sampling-type DFrFT is considered (please, see the second paragraph of Sect. 1). Moreover, in [30], the architecture is restricted to the computation of a 256-point DFrFT, without a report of how the hardware complexity behaves for general  $N$ -point transforms, nor numerical error analysis (only a qualitative architecture error analysis for a small set of input signal and parameters is provided). A fair comparison of latency in clock cycles between [30] and our architectures is unfeasible, because, although the cited paper provides the latency in microseconds ( $= 2.819\mu s$ ), it does not inform the running clock frequency. The proposed HGL<sub>2</sub> architecture has memory requirements (in bits) similar to those given in [30],  $\approx 848$  kbits and 828 kbits, respectively [31]. Although the arithmetic hardware complexity is not properly detailed (numbers of adders and multipliers) in [30], according to the FPGA documentation used in the referred work [31], we can deduce that the number of required arithmetic hardware component is significantly higher than that required in our architectures. Finally, the number of registers used in the referred architecture is, at least, 7 times greater than ours. In brief, even taking into account the caveats outlined above and the incompleteness of some experiments and records in [30], our preliminary conclusion is that, in terms of hardware complexity, our architectures outperform that described in the cited paper.

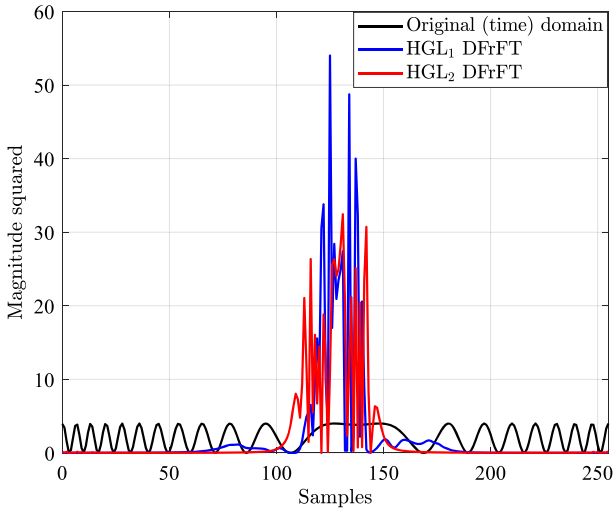
#### 4.4 Compact Fractional Fourier Domains

In order to analyze the searching performance of an optimum fractional order for compact fractional Fourier domain,  $a_{opt}$ , and measurement of compactness,  $l_1$ -norm, of each DFrFT architecture, we apply the minimum norm method (MNM) proposed in [24] considering the same synthetic bi-component chirp signal used in [8] for  $N = 256$ . A program runs the MNM algorithm in the HPS, while the FPGA computes the DFrFTs through the proposed DFrFT architectures.

Based on the method given in [8], we can search the  $a_{opt}$ , using the same intermediate vector  $\mathbf{x}'$ . Therefore, the proposed architectures compute  $\mathbf{x}'$  only once, saving

**Table 5** Application of the MNM to a bi-component chirp signal by using the HGL<sub>1</sub> and HGL<sub>2</sub> architectures

Architecture	$a_{opt.}$	$l_1$ -norm	Time (s)	Number of DFrFTs
HGL <sub>1</sub>	1.181494	206.369750	0.051616	59
HGL <sub>2</sub>	1.110355	194.556843	0.041657	59

**Fig. 8** Magnitude squared of chirp signal in original (time) and fractional Fourier domains encountered by using the MNM algorithm

approximately 40% of computational arithmetic operations of the remaining DFrFTs before the algorithm convergence.

In Table 5, one can observe the respective  $a_{opt.}$ ,  $l_1$ -norm, number of DFrFTs required to find  $a_{opt.}$  and algorithm time-convergence according to each DFrFT architectures. We observe that the optimal fractional orders achieved by the proposed architectures are close to the results obtained in [8]. Additionally, the HGL<sub>2</sub>-based architecture reached the lowest  $l_1$ -norm, i.e., the greatest compactness performance. Due to inherent errors of floating-point to fixed-point conversions (and vice-versa) and fixed-point arithmetic operations used in the DFrFT architectures, we believe that the proposed method requires 4 additional DFrFTs for convergence of the MNM algorithm, when compared to the results given in [8].

The results can be visually compared by using Fig. 8, where the squared magnitude of the original chirp signal (black line) and the corresponding compact representation in fractional Fourier domain computed by the DFrFT architectures are plotted. Although the HGL<sub>1</sub>-based DFrFT (blue line) have a central portion narrower than that of the HGL<sub>2</sub>-based DFrFT (red line), side fluctuations contribute to higher  $l_1$ -norm [8].

## 5 Conclusion

We presented two hardware architectures for computing ED-DFrFTs of  $N$ -length complex-valued vectors, where  $N = 4L$  and  $L$  is a positive integer. The proposed architectures can be generalized for  $N \neq 4L$  with slight changes of variable indexes of the memory blocks. By means of comparative analyses, we demonstrated that reliable DFrFTs can be achieved with low error through fixed-point numerical representations and appropriate conversions, instead of floating-point numerical representations. Among the proposed architectures in this study, the HGL<sub>2</sub>-based architecture achieved better speed performance and memory usage than HGL<sub>1</sub>-based architecture, but additional arithmetic hardware elements are required. Moreover, the HGL<sub>2</sub>-based architecture achieved a smaller  $l_1$ -norm, i.e., a greater compactness performance associated with a lower latency, when compared to HGL<sub>1</sub>-based architecture. Although pipelined architectures presented in the literature outperform HGL<sub>2</sub> architecture regarding latency performance, the proposed architectures have the lowest arithmetic hardware complexity among the existing architectures. Therefore, choosing a specific architecture depends on the application and involves aspects related to area and timing performance.

We are currently investigating the possibility of decreasing the latency of the HGL<sub>2</sub> architecture through pipelined approaches and implementing rounding strategies in HGL<sub>2</sub> eigenbasis, as proposed in [8], in order to reduce the number of arithmetic operations without significantly compromising the accuracy of the DFrFT architecture.

The proposed methods are also suitable for application-specific integrated circuit (ASIC) design, which can be considered to achieve speed performance in real-time applications with low hardware complexity, i.e., minimal circuit area, such as filtering in the fractional domain, time–frequency analysis, chirp-rate estimation, compressive sensing, velocity and vibration estimation on SAR imagery and similar applications.

**Author Contributions** BCB, JRDON and JBL were involved in conceptualization; BCB, JRDON and JBL helped in methodology; BCB and JRDON contributed to formal analysis and investigation; BCB was involved in writing—original draft preparation; BCB, JRDON and JBL helped in writing—review and editing; JRDON and JBL contributed to resources; JRDON and JBL helped in supervision. All authors read and approved the final manuscript.

**Funding** This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grants 310142/2020-2, 409543/2018-7 and 140151/2022-2, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), and Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco (FACEPE) under Grant APQ-1226-3.04/22.

**Data availability** Not applicable.

**Code availability** Not applicable.

## Declarations

**Conflict of interest** The authors declare that there is no conflict of interest.

**Ethics approval** Not applicable.

## References

1. A. Acharya, S. Mukherjee, Designing a re-configurable fractional Fourier transform architecture using systolic array. *Int. J. Comput. Sci. Issues* **7**, 159 (2010)
2. R.E. Blahut, *Fast Algorithms for Signal Processing* (Cambridge University Press, Cambridge, 2010). <https://doi.org/10.1017/cbo9780511760921>
3. C. Candan, M. Kutay, H. Ozaktas, The discrete fractional Fourier transform. *IEEE Trans. Signal Process.* **48**(5), 1329–1337 (2000). <https://doi.org/10.1109/78.839980>
4. C. Candan, On higher order approximations for Hermite–Gaussian functions and discrete fractional Fourier transforms. *IEEE Signal Process. Lett.* **14**(10), 699–702 (2007). <https://doi.org/10.1109/LSP.2007.898354>
5. A. Cariow, J. Papliński, D. Majorkowska-Mech, Some structures of parallel VLSI-oriented processing units for implementation of small size discrete fractional Fourier transforms. *Electronics* **8**(5), 509 (2019). <https://doi.org/10.3390/electronics8050509>
6. G. Cincotti, T. Murakawa, T. Nagashima et al., Enhanced optical communications through joint time-frequency multiplexing strategies. *J. Lightwave Technol.* **38**(2), 346–351 (2020). <https://doi.org/10.1109/JLT.2019.2942452>
7. J.R. de Oliveira Neto, J.B. Lima, Discrete fractional Fourier transforms based on closed-form Hermite–Gaussian-like DFT eigenvectors. *IEEE Trans. Signal Process.* **65**(23), 6171–6184 (2017). <https://doi.org/10.1109/TSP.2017.2750105>
8. J.R. de Oliveira-Neto, J.B. Lima, G.J. da Silva et al., Computation of an eigendecomposition-based discrete fractional Fourier transform with reduced arithmetic complexity. *Signal Process.* **165**, 72–82 (2019). <https://doi.org/10.1016/j.sigpro.2019.06.032>
9. A. Gómez-Echavarría, J.P. Ugarte, C. Tobón, The fractional Fourier transform as a biomedical signal and image processing tool: a review. *Biocybern. Biomed. Eng.* **40**(3), 1081–1093 (2020). <https://doi.org/10.1016/j.bbe.2020.05.004>
10. M.T. Hanna, N.P.A. Seif, W.A. El-Maguid-Ahmed, Discrete fractional Fourier transform based on the eigenvectors of tridiagonal and nearly tridiagonal matrices. *Digit. Signal Process.* **18**(5), 709–727 (2008). <https://doi.org/10.1016/j.dsp.2008.05.003>
11. Y. Hu, CORDIC-based VLSI architectures for digital signal processing. *IEEE Signal Process. Mag.* **9**(3), 16–35 (1992). <https://doi.org/10.1109/79.143467>
12. Intel, *ALTERA\_CORDIC IP core user guide* (Intel, 2017). <https://www.intel.com/content/www/us/en/docs/programmable/683808/current/altera-cordic-ip-core-user-guide.html>
13. Intel, *Cyclone V Device Datasheet* (Intel, 2019). <https://www.intel.com/programmable/technical-pdfs/683801.pdf>
14. A. Kuznetsov, Explicit Hermite-type eigenvectors of the discrete Fourier transform. *SIAM J. Matrix Anal. Appl.* **36**(4), 1443–1464 (2015). <https://doi.org/10.1137/15M1006428>
15. A. Kuznetsov, M. Kwaśnicki, Minimal Hermite-type eigenbasis of the discrete Fourier transform. *J. Fourier Anal. Appl.* **25**, 1053–1079 (2019). <https://doi.org/10.1007/s00041-018-9600-z>
16. J. Lima, L. Novaes, Image encryption based on the fractional Fourier transform over finite fields. *Signal Process.* **94**, 521–530 (2014). <https://doi.org/10.1016/j.sigpro.2013.07.020>
17. D. Majorkowska-Mech, A. Cariow, A low-complexity approach to computation of the discrete fractional Fourier transform. *Circuits Syst. Signal Process.* **36**(10), 4118–4144 (2017). <https://doi.org/10.1007/s00034-017-0503-z>
18. V. Namias, The fractional order Fourier transform and its application to quantum mechanics. *IMA J. Appl. Math.* **25**(3), 241–265 (1980). <https://doi.org/10.1093/imamat/25.3.241>
19. S. Pei, W. Hsue, J. Ding, Discrete fractional Fourier transform based on new nearly tridiagonal commuting matrices. *IEEE Trans. Signal Process.* **54**(10), 3815–3828 (2006). <https://doi.org/10.1109/TSP.2006.879313>
20. R. Pelich, N. Longépé, G. Mercier et al., Vessel refocusing and velocity estimation on SAR imagery using the fractional Fourier transform. *IEEE Trans. Geosci. Remote Sens.* **54**(3), 1670–1684 (2016). <https://doi.org/10.1109/TGRS.2015.2487378>
21. M. V. N. V. Prasad, K. C. Ray, A. S. Dhar, FPGA implementation of discrete fractional Fourier transform. In: 2010 International Conference on Signal Processing and Communications (SPCOM) (2010), pp. 1–5. <https://doi.org/10.1109/SPCOM.2010.5560491>

22. K.C. Ray, M.V.N.V. Prasad, A.S. Dhar, An efficient VLSI architecture for computation of discrete fractional Fourier transform. *J. Signal Process. Syst.* **90**(11), 1569–1580 (2018). <https://doi.org/10.1007/s11265-017-1281-3>
23. G.A.F. Seber, *A Matrix Handbook for Statisticians*, 1st edn. (Wiley, Hoboken, 2007)
24. A. Serbes, Compact fractional Fourier domains. *IEEE Signal Process. Lett.* **24**(4), 427–431 (2017). <https://doi.org/10.1109/LSP.2017.2672860>
25. A. Serbes, L. Durak-Ata, Efficient computation of DFT commuting matrices by a closed-form infinite order approximation to the second differentiation matrix. *Signal Process.* **91**(3), 582–589 (2011). <https://doi.org/10.1016/j.sigpro.2010.05.002>
26. P. Sinha, S. Sarkar, A. Sinha et al., Architecture of a configurable centered discrete fractional Fourier transform processor. In: 2007 50th Midwest Symposium on Circuits and Systems (2007), pp. 329–332. <https://doi.org/10.1109/MWSCAS.2007.4488600>
27. X. Su, R. Tao, X. Kang, Analysis and comparison of discrete fractional Fourier transforms. *Signal Process.* **160**, 284–298 (2019). <https://doi.org/10.1016/j.sigpro.2019.01.019>
28. J.E. Volder, The CORDIC trigonometric computing technique. *IRE Trans. Electron. Comput.* **8**(3), 330–334 (1959). <https://doi.org/10.1109/TEC.1959.5222693>
29. Q. Wang, M. Pepin, R.J. Beach et al., SAR-based vibration estimation using the discrete fractional Fourier transform. *IEEE Trans. Geosci. Remote Sens.* **50**(10), 4145–4156 (2012). <https://doi.org/10.1109/TGRS.2012.2187665>
30. R. Wang, P. Chen, D. Wang, FPGA-based implementation of discrete fractional Fourier transform algorithm. In: 2022 14th International Conference on Wireless Communications and Signal Processing (WCSP) (2022), pp. 511–515. <https://doi.org/10.1109/WCSP55476.2022.10039232>
31. Xilinx, *Xilinx 7 Series FPGAs Data Sheet: Overview* (Xilinx, 2020). [https://docs.xilinx.com/v/u/en-US/ds180\\_7Series\\_Overview](https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.