Hw/Sw Co-Design technique for 2D fast fourier transform algorithm on Zynq SoC

Kortli Yassin, Souhir Gabsi, Maher Jridi, Ayman Alfalou, Mohamed Atri

Version of Record: https://www.sciencedirect.com/science/article/pii/S0167926021001085
Manuscript_4766b7a4445b4d4af2bcf181232c3c89

Hw/Sw Co-Design Technique for 2D Fast Fourier Transform Algorithm on Zynq SoC

Yassin Kortli [1,2], Souhir Gabsi [2], Maher Jridi [1], Ayman Alfalou [1], and Mohamed Atri [2,3]

[1] L@bisen, AI-ED Lab., Yncrea ouest, 20 rue du Cuirassé de Bretagne, 29200 Brest, France
[2] Electronic and Micro-electronic Laboratory, Faculty of Sciences of Monastir, University of Monastir, Tunisia
[3] College of Computer Science, King Khalid University, Abha 61421, Saudi Arabia

Email: yassin.kortli@isen-ouest.yncrea.fr, souhir.gabsi@fsm.rnu.tn , maher.jridi@isen-bretagne.fr, matri@kku.edu.sa, ayman.al-falou@isen-ouest.yncrea.fr

Abstract- The Two-Dimensional Fast Fourier Transform (2D-FFT) algorithm is used for the study of many modern systems applied for security and biometrics. The adoption of this algorithm, which is a compute intensive task, is limited due to its hardware design complexity. The first objective of this paper is to underline the effect of the hardware/software co-design (Hw/Sw co-design) for the reduction of the processing time and power consumption. Secondly, we propose an innovative architecture for the 2D-FFT algorithm tested on Zynq Soc, which requires less processing time and memory compared to the traditional algorithm. Three implementations (one software and two Hw/Sw co-designs) of the 2D-FFT algorithm using the Zynq SoC are presented in this paper. The first is based on ARM processor. A speedup of 29x is obtained compared to the original implementation thanks to many optimizations. The second is a Hw/Sw co-design solution of the traditional 2D-FFT algorithm introduced on a hybrid platform combining an ARM Cortex-A9 processor with an FPGA. The third is also a Hw/Sw co-design solution using our optimized 2D-FFT algorithm to reach the real-time contraints for high-resolution images(1920×1080). It provides a speedup of 1.13x, 3.31x and 96.21x faster than the Hw/Sw co-design implementation of the traditional RC algorithm, the pure software implementations with and without optimizations, respectively.

Keywords: 2D-FFT algorithm; Embedded systems; Hw/Sw Co-design; High-level synthesis; Complexity.

1. Introduction

The Fast Fourier Transform (FFT) algorithm was initially introduced by J. Cooley

and J.Tukey [1] in 1965 as a useful algorithm for image processing. It has become the most effective algorithm used for computer vision to obtain the frequency or the spectrum content of an image for many applications such as correlation filters and discrete convolution [2][3][4]. It is necessary to improve their performance to respect real-time and memory consumption constraints. However, the FFT algorithm is costly in terms of processing time and memory consumption [4][5][6]. More specifically, the traditional 2D-FFT algorithm is achieved by performing two 1D-FFT algorithms along rows or columns and transpose operation[1]. This affects the overall latency and could be designed differently. The use of embedded systems such as Graphics Processing Units (GPUs)[4], a combination of Hw/Sw Co-Design, and Field Programmable Gate Arrays (FPGAs)[7] are expected to solve these challenges. In this work, we have used the Zynq SoC FPGA-based platform, which presents better performance in terms of energy efficiency compared to GPUs[4]. Zynq SoC FPGAs are a good candidate for computer vision system development for their ability to exploit parallelism. Additionally, the design flow of current FPGAs synthesis tools supports high-level abstraction languages (C/C++) as input descriptions, in contrast to hardware description languages (HDL: Verilog and VHDL). High-level synthesis (HLS) is used to simplify the design process by transforming the algorithmic description into hardware while satisfying the design constraints.

Given the importance of using the 2D-FFT algorithm in many applications, it could be interesting to propose a different implementation of this algorithm. In this paper, three implementations (one software and two Hw/Sw co-designs) of the 2D-FFT algorithm using the Zynq SoC are proposed. The first is based on ARM Cortex-A9 processor to perform the software implementation using OpenCV functions. The second is based on a hybrid platform combining an ARM Cortex-A9 processor with an FPGA to perform a Hw/Sw co-design implementation. The Hw/Sw co-design implementation provides less processing time and less power, which is the first contribution of this paper. The second contribution is an innovative architecture for the 2D-FFT algorithm using the Zynq SoC. It requires less processing time and less dedicated hardware blocks (DSP, LUT, and FF) compared to the traditional algorithm. Particularly, we present two different Hw/Sw co-design implementations of the 2D-FFT algorithm using the Zynq SoC: (i) traditional Row-Column (RC) algorithm with transpose operation and (ii) optimized RC algorithm without transpose operation. Both 2D-FFT algorithms were compared in terms of processing time, memory, and dedicated hardware blocks.

Generally, the Radix-2 and Radix-4 architectures are the basic architectures used to implement the FFT IP core. The decimation-in-frequency and non-discrimination are often used for the pipelined streaming and burst I/O architectures, respectively. Moreover, it is well known that N-point FFT using Radix-2 introduces log2 (N) steps, each step containing N/2 Radix-2 butterflies. However, N-point FFT using Radix-4 introduces log4 (N) steps, each step containing N/4 Radix-4 butterflies. The butterfly architecture used and the number of inputs and outputs are the main difference between these architectures[8]. In this work, four architectures have been assessed in order to select the best one. These architectures include Radix-4 Burst I/O, Radix-2 Burst I/O, Pipelined Streaming I/O, and Radix-2 Lite Burst I/O. Vivado HLS tool is used to generate the hardware description (at RTL level) from software-coded functions and targeting Zynq SoC. This description serves to exchange data between the software part and the hardware part. Many optimizations are introduced in order to develop suitable hardware implementation. The contributions of this paper are as follows:

- Proposing a Hw/Sw co-design solution of the 2D-FFT algorithm, which provides less processing time, and less power consumption using the Zynq SoC.
- Innovative architecture for the 2D-FFT algorithm tested on Zynq Soc that requires less processing time, memory, and less number of computational blocks compared to the traditional algorithm.
- A comparative study between four existing architectures to select the best one.
- Study the impact of directives on resource usage and performance.

The rest of the paper is organized as follows: Section 2, presents the different applications of the 2D-FFT algorithm and platforms used for these implementations. Section 3 describes our innovative architecture for the 2D-FFT algorithm using the Zynq SoC. Section 4 describes the software implementation based on ARM Cortex-A9 processor. Section 5 presents the Hw/Sw co-design solution using the Zynq SoC and the different optimizations for both 2D-FFT algorithms performed on ARM Cortex-A9 processor and the FPGA. Section 6 presents and discusses all implementations. The last section concludes the work and gives future research directions.

2. RELATED WORK

Computer vision techniques are getting more and more popular in a variety of areas

in scientific research, such as identification [2][9] [7], classification [10][3][11][12], and marking images [13]. Fast Fourier Transform (FFT) has been introduced in many of the above-listed applications. Several applications for image processing are presented and discussed to prove the importance of the 2D-FFT algorithm. For example, Lamas-Seco et al. [14] implement a novel algorithm based on the Fourier Transform (FT) to extract some spectral features of inductive signatures used in traffic management systems. In addition, a variant of the 2D-FFT algorithm named Enhanced Partial Discrete Fourier Transform (EP-DFT) is used to implement a multi-biometric system to improve recognition accuracy and security [15]. A simplified correlation method based on the 2D-FFT is implemented to simplify the correlation setup[10]. Recognition with this method is done without resorting to inverse 2D-FFT compared to the traditional correlation application. In addition, some research work focuses on image preprocessing to improve performance. Zhang et al.[16] proposes a system for face recognition with Principal Component Analysis (PCA) approach, which applied the Fast Fourier Transform (FFT) to combine the phase spectrum of one image with the amplitude spectrum of another image as a mixed image. In addition, Khan et al.[17] develops a computational method based on Particle Swarm Optimization (PSO) and the Discrete Fourier Transform algorithm.

The different applications mentioned previously prove that either a combination between 2D-FFT algorithm and other algorithms is used for objects recognition tasks. The FFT algorithm is costly in terms of processing time and power consumption, many researchers have studied efficient acceleration based on FPGAs and GPUs by exploiting their high parallelism capabilities. Indeed, the heterogeneous multicore SoC includes CPU, GPU and FPGA are proposed in many types of research. An example of the heterogeneous multicore platform used to accelerate the vehicle detection process was implemented by Cheng et al. [5]. This platform includes an Intel i5-2400 processor and an AMD HD6670 GPU. Zhang et al.[18], proposes a novel method based on the CPU-FPGA platform with coherent shared memory to accelerate state-of-art Convolutional Neural Networks (CNNs). Ouerhani et al.[4] proposes an implementation of the correlation technique based on the Nvidia Geforce 8400 GS GPU for facial recognition. This technique is based on the 2D-FFT filter and the phase-only filter (POF). A mixed radix FFT and hierarchical FFT algorithms for both power-of-two and non-power-of-two sizes using GPUs are presented in [6]. The work in [19] exploits the similarity between adjacent inputs and sparse input samples in stream processing to enhance the

efficiency of sparse FFT algorithm. The sparse FFT implementation is evaluated based on Intel i7 CPU and three NVIDIA GPUs, i.e., NVIDIA GeForce GTX480, Tesla C2070, and Tesla C2075. Hyun et al. [20] developed a Virtex-5 FPGA implementation of signal processing of the vehicle FMCW radar systems with a two-step FFT. Smach et al. [7] presented a hardware implementation using FPGA technology in order to accelerate the computation of Fourier Descriptors (FD). The Support Vector Machine (SVM) is used for classification. To optimize the radix-2 FFT FPGA implementation, a novel algorithm is proposed to simplify the computation by Walid et al [21]. Raju et al. [22] present two implementations of DFT/IDFT architectures on the FPGA. These architectures are based on radix 2 butterfly because of its reduced computation time. Yu et al. [26] develop a Multi-dimensional (MD) Discrete Fourier Transform (DFT) algorithm based on Xilinx Virtex-5 FPGA. The proposed algorithm can support 2D, 3D, and even higher dimensional DFT. Li and Wyrwicz [27] report the design and implementation of a parallel 2D-FFT) algorithm on FPGA for real-time MR image processing. The results indicate that the image-reconstruction acceleration is primarily limited by the speed of the data transfer between the FPGA device and external sensors. Ouerhani et al. [29] allow to optimize existing FFT algorithms for low-cost FPGA implementations.

Finally, this paper focuses on FPGA-based platforms, which presents a better performance in terms of energy efficiency compared to GPU platforms. The FPGA shows better results in terms of performance with low power consumption. GPU platforms, also, provide high-performance results but consume more power [4]. Unlike the different implementations introduced in relation to the FPGAs, this work contributes with a Hw/Sw co-design of the 2D-FFT algorithm using the Zynq SoC respecting real-time execution and memory consumption constraints. As a result, the FPGA is widely used as an accelerator for image processing due to the possibility of reducing execution time and extracting parallel computations. However, this accelerator requires a thorough knowledge of hardware description languages (HDL) such as VHDL and Verilog. These languages are used to describe and synthesize at the Register-Transfer Level (RTL) levels. Today, high-level synthesis tools (HLS) is used to synthesize hardware description (RTL) from a high-level language such as C/C ++ and SystemC. In addition, HLS tools can not directly transform high-level language into RTL levels. As a result, it needs to be optimized and restructured to be suitable and synthesizable for a specific hardware platform.

Vivado HLS reduces design time and makes complex algorithms much easier than to use HDL. However, Vivado HLS does not support the compilation of all C/C ++ command structures. So, Xilinx provides many popular libraries and model templates for programming users. The *hls_fft.h* and *ap_fixed.h* libraries are used to implement our proposed FFT IP core. The float data type is used for the default implementation of FFT IP core. Then, to reduce the processing time and the resources number, it is better to transfer the input data to fixed-point data. Once compiled into HDL, modules on Vivado HLS produces an FFT IP core, which can be instantiated and simulated with other software such as Matlab, ISE. Besides, this FFT IP core subject to various optimizations in order to improve performance and reduce memory requirements. The design process using Vivado HLS is shown in Fig. 1.

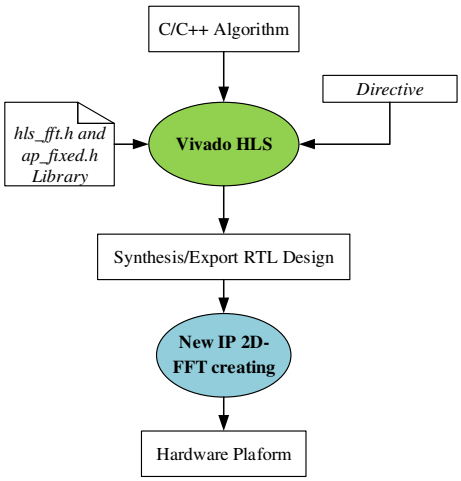


Fig.1: Design Process on Vivado HLS

3. 2D-FFT Algorithms

The 2D-FFT algorithm can be used to identify any object by extracting in the transform domain features that can be used for identification or recognition[10]. It considered as a local descriptor to represent an object instead of the global descriptor. The 2D-FFT and its inverse are defined as follows:

$$\text{2D- FFT} \quad F(u,v) = \sum_{x=0}^{N} \sum_{y=0}^{N} f(x,y) W_N^{ux} W_N^{vy} \tag{1}$$

$$\text{2D- IFFT} \quad f(x,y) = \frac{1}{N^2} \sum_{u=0}^{N} \sum_{v=0}^{N} F(u,v) W_N^{-ux} W_N^{-vy} \tag{2}$$

when N indicates the number of sample points, f(x, y) indicates the pixel value of the input image, F (μ, v) indicates coefficients of the output image, and $W_N = e^{-2\pi i/N}$ are

the twiddle factor, and x, y, u and v represent the coordinate of the input and transformed image. Hence, to compute N-point FFT, we need respectively $N\ log2(N)$ and $(N/2)\ log2(N)$ complex additions and complex multiplications. The traditional RC algorithm is computed with three steps. Firstly, the input 2D frame is initially saved in the local memory. After that, the 1D-FFT transform is applied along the columns. Secondly, the result is saved in the local memory and transposed (columns to rows). Thirdly, the second 1D-FFT transform is applied to columns another time.

However, the traditional RC algorithm is not efficient and has poor performance for large image processing because the transpose operation requires more resources and time to transfer data to and from memories [23]. To solve this problem, we introduce an optimized RC algorithm without transpose operation, which computes with two steps:

1. In the first step, we perform the 1D-FFT algorithm on columns of the input image. The intermediate results are written and stored inversely (i.e. stored in rows not in columns) in two memories for real and imaginary parts.
2. In the second step, the 1D-FFT algorithm is reactivated on columns of the 1D-FFT image obtained previously. The final results are stored as real and imaginary data parts in two separate memories.

The optimized RC algorithm needs two reading and writing operations, while the traditional RC algorithm requires three operations. Both implementations are depicted in Fig. 2.

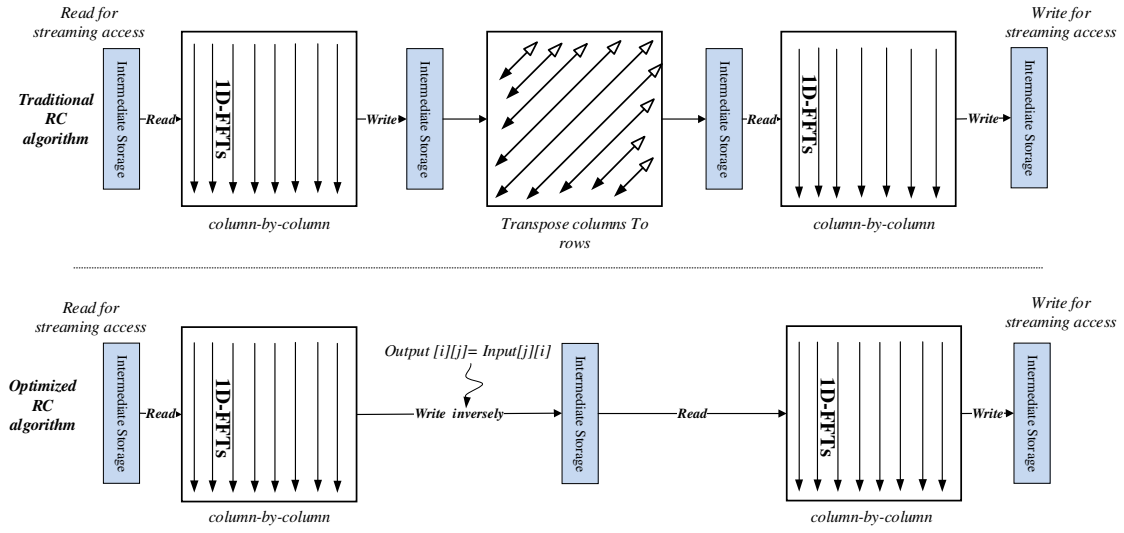


Fig.2. Traditional and optimized RC algorithms using Zynq SOC.

4. Software Implementation of the 2D-FFT Algorithm

The OpenCV library provides an implementation for many interesting applications dedicated to software platforms. In addition, the 2D-FFT algorithm applied to image reconstruction can be used for the validation of the software implementation based on ARM Cortex 9 processor using the Zynq device. A SIMD architecture integrated into the ARM Cortex 9 processor is used to accelerate the execution. The 2D-FFT algorithm provided by the OpenCV library applied on the input frames stored in the DDR memory. The data transfer between the ARM Cortex 9 processor and the DDR memory is controlled with a Video Direct Memory Access (VDMA) core based on the High Performance (AXI HP) port of the Zynq SoC[24]. Finally, the output result is forwarded to the VGA output. The software implementation of the 2D-FFT algorithm is configured to forward direction. We should notice that for a single high-resolution frame 1920x1080, this implementation requires 229 ms, which is inefficient for real-time applications. Two pre-processing steps are introduced to reduce the computation time of the 2D-FFT algorithm based on OpenCV functions: grayscale conversion and resizing. Fig. 3 shows the design of the software implementation based on ARM Cortex 9 processor.

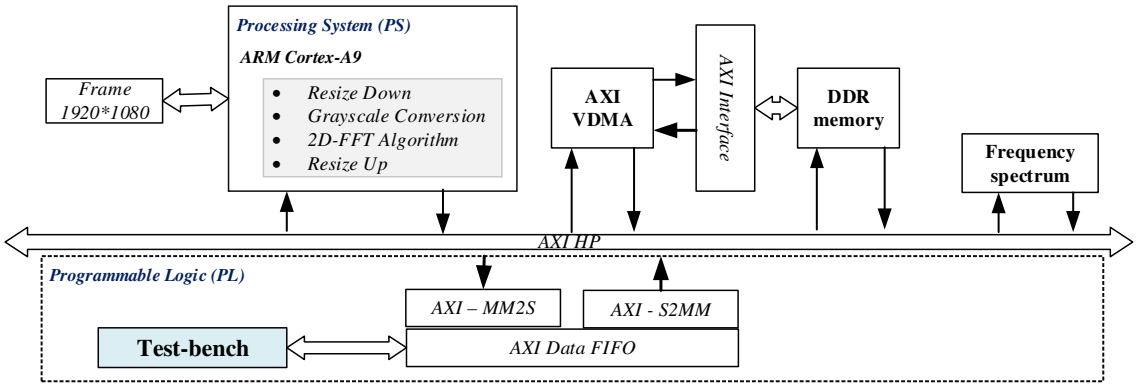


Fig.3: Software implementation based on ARM Cortex 9 processor.

4.1. Pre-processing

The goal of the pre-processing step is to reduce the complexity and the amount of processed data. The input frame is first affected by a resizing up followed by gray-scale conversion. Generally, reducing the size of the frame does not substantially affect the important details. The spectrum Nearest-Neighbor interpolation algorithm is used to resize down/up a frame. After reducing the frame, the gray-scale conversion is applied[25]. The above-mentioned optimizations are combined to obtain the best system performance and to decrease the computation time. The complete system steps are presented in Fig. 4.

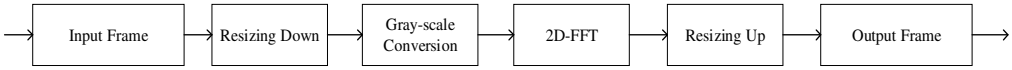


Fig. 4 A complete system steps.

4.2. Evaluation

The optimization used (resizing up / down, grayscale conversation) reduce the complexity and the computation time of the 2D-FFT algorithm due to the decreasing of the pixels to be treated. An image of resolution 1920x1080 is used as input. Fig. 5 presents the resize scale and the processing time in function with the PSNR of the software implementation of the 2D-FFT algorithm based on ARM Cortex-A9 processor. Fig. 5(a) presents the PSNR variation with the resize scale down for a high-resolution image, and Fig. 5(b) presents the processing time in function with resizing scale down for the whole 2D-FFT algorithm.

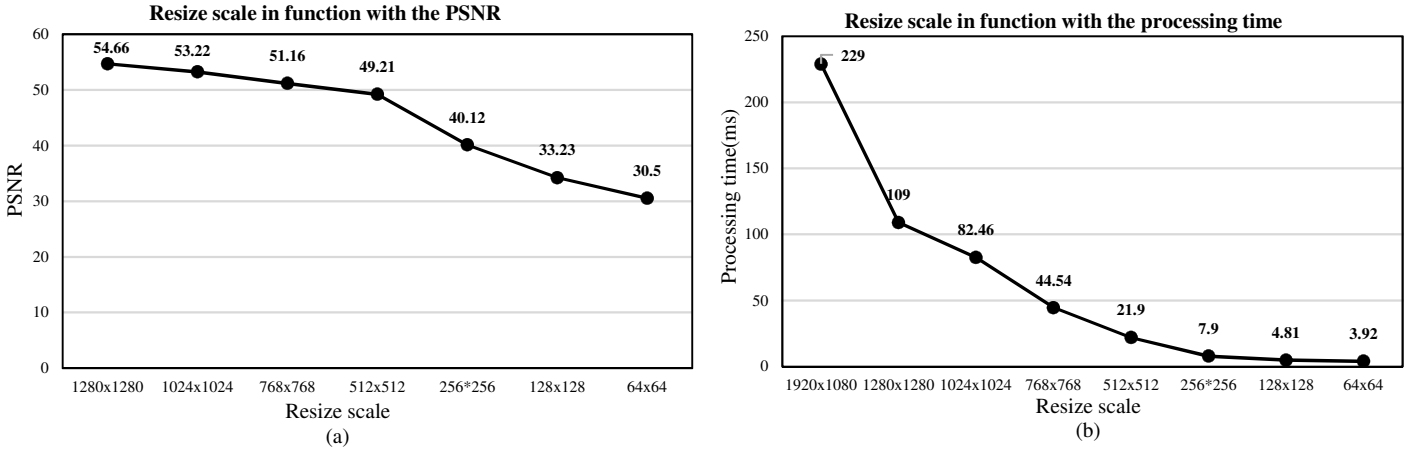


Fig. 5: Resize downscale in function with PSNR and processing time based on ARM Cortex-A9 processor of the 2D-FFT algorithm for different size of image.

The output spectrum quality is analyzed for different scaling parameters by using the PSNR parameter. At the same time, the processing time is assessed for the same scaling parameters. The typical value of the PSNR used in signal processing to measure the quality is bigger than 35 dB. It is found that the processing time obtained with the resized image at 256x256 pixels is more than 29 times lower than that obtained with the size 1920x1080 pixels with a PSNR value equal to 40.12 dB. By these simulations, we emphasize the need to resize the images in order to dramatically decrease the processing time without a significant decrease in the quality. As a result, we have resized the input frame at 256x256 pixels to reduce the amount of processed data.

5. Hw/Sw Co-Design Implementation of the 2D-FFT Algorithms

This section presents 2D-FFT algorithm implementation tested on hybrid processing units: ARM Cortex-A9 processor and FPGA. The advantage of the Zynq SoC from Xilinx is the ability to generate any algorithm written with High-Level Language: C, C++, and SystemC. Some other tools can generate RTL description using python language. The HLS tool is used to reduce design time and greatly facilitate the deployment of complex algorithms in the FPGA. However, the functions written with C/C++ can be optimized and specified based on many directives in order to enhance the performance and reduce the processing time using the HLS tools. The Zynq SoC can automatically manage the exchange of data between the software part and the hardware part. Traditional and optimized RC algorithms will be accelerated on FPGA. Fig. 6 presents an overview of the Hw/Sw co-design of the traditional RC algorithm based on a hybrid platform combining an ARM processor with an FPGA.

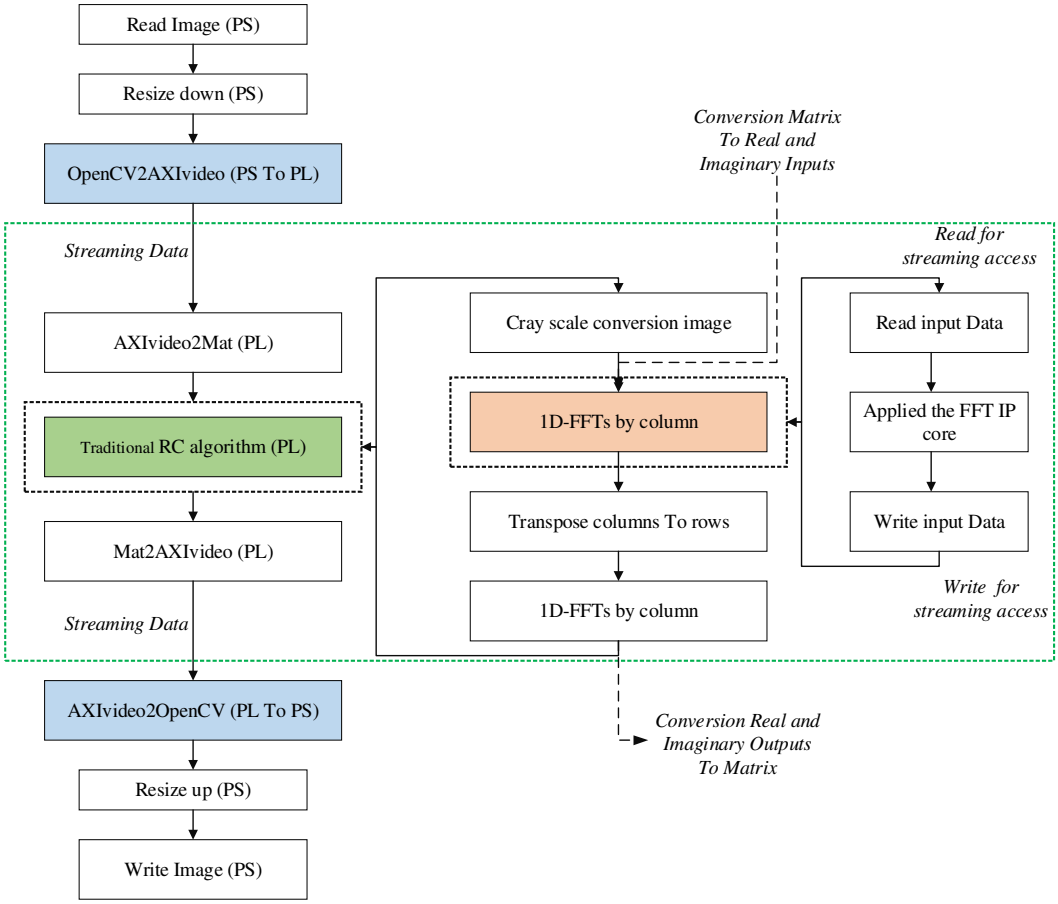


Fig. 6 Overview of the Hw/Sw co-design of the traditional RC algorithm using Zynq SoC.

An FPGA-based implementation is used processed the traditional RC algorithm into hardware acceleration to achieve real-time implementation. As shown in Fig. 6, the

algorithms that are processed as hardware part are gray-scale conversion, 1D-FFT algorithm on columns, transpose operation, and the second 1D-FFT algorithm on columns, while the read/write frame and resizing down/up are processed as software. This choice is made because reading and writing processes do not require any arithmetic operations. Generally, the ARM processor is well optimized for this kind of operations. The *hls::AXIvideo2Mat* IP-Core is used to converts data stored in *hls::Mat* format to an AXI4 video stream format. While *hls::Mat2AXIvideo* IP-Core is used to perform the inverse of the *hls::AXIvideo2Mat* IP-Core.

More specifically, traditional or optimized RC algorithms are built with two blocks 1D-FFT core. Local memory is used to read and to write the input and the output data for each block. Each block has three steps: read the input data via the local memory, apply the proposed FFT IP core, write the output data and transfer it to the local memory, respectively. The proposed FFT IP core is formed by 32 radix-2 butterflies: 32 FIFOs with 32 bits constituting the real part, 32 FIFOs with 32 bits constituting the imaginary part, FIFO logic for routing outputs to the input, and complex twiddle factor with 24 bits. The real and imaginary output parts are fixed-point data with 32bits. Fig. 7(a) shows a diagram of the proposed 1D-FFT IP core.

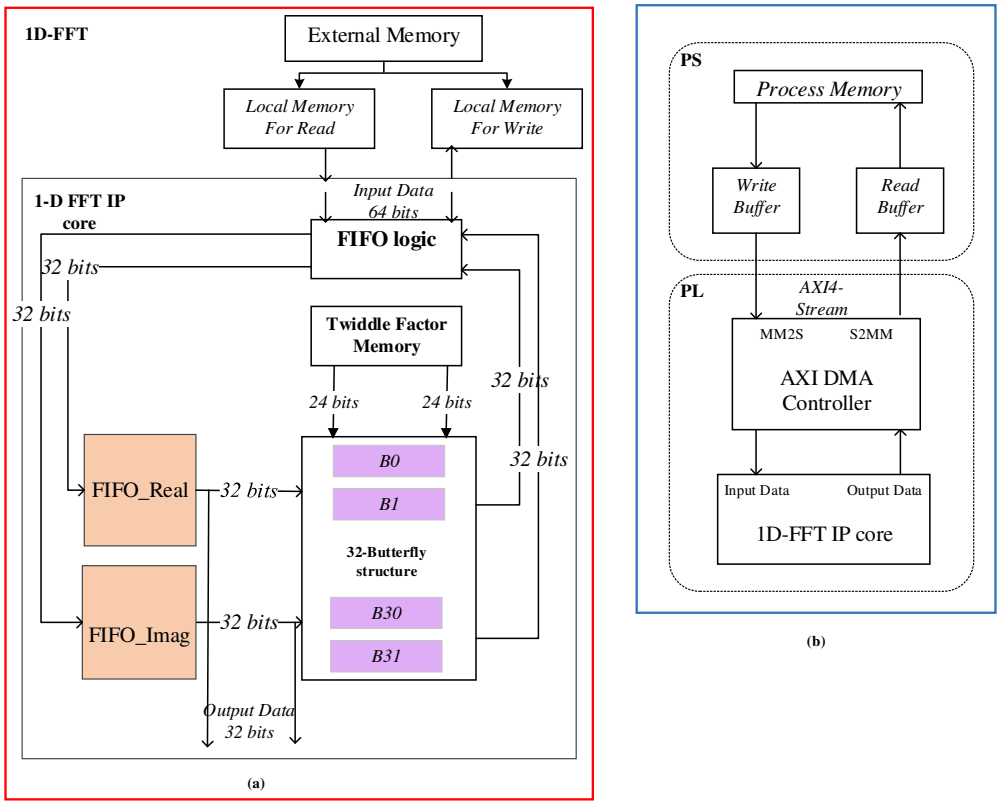


Fig. 7. (a) Diagram of the proposed 1D-FFT IP core, (b) hierarchical sequence of memories between the PS and the PL for the 1D-FFT accelerator.

However, when saving the input frame in the local memory, the amount of BRAM resources is increased. An AXI bus interface can be used for communication between the PS and the PL. Vivado HLS provides three types of AXI communication: AXI4-Lite bus, AXI4 interface, and AXI4-Stream. The first type is used to transfers the mapped data in low-rate memory. The second type is suited for high-performance memory-mapped operations, which is for sending and receiving bigger data. The third type is AXI4-Stream will be used for this project, it is suitable for streaming a large amount of data at high speed, such as video streaming. Row buffers save the input data in the local memory and transmitted it to AXI VDMA. These buffers introduced by the class *HLS:: stream*. The VDMA allows transferring data between the ARM and the FPGA or conversely. The interface of each buffer is controlled with the *#pragma directives*.

Fig. 7 (b) shows the hierarchical sequence of memories between the PS and the PL for the 1D-FFT IP core. The scatter-gather mode of AXI DMA is used to configure each interface and to transfer data to DDR memory. AXI DMA controls direct data transfers between AXI4-Stream target devices and system memory. From the DDR memory, the data is read and write via an AXI4 stream master. Memory-mapped to AXI4 stream (*MM2S*) master and AXI stream to memory-mapped (*S2MM*) slave are respectively two channels provided by AXI DMA for reading and writing data. The *MM2S* bus reads the data from DDR memory and transmits them to PL, while *S2MM* bus writes the data from PL and transmits them to DDR memory. As a result, AXI DMA core controls the data transfers, the address generation, and the transaction planning in memory between the DDR memory and PL.

5.1. Optimization

The 2D-FFT algorithm needs a high amount of resources and should be optimized. Using the Vivado HLS device, we can handle FPGA implementation constraints by several optimization techniques, known as synthesis pragmas or directives. The performance of the hardware implementation varies depending on the configurations of directives. Here, we can analyze the impact of some of them:

***DATAFLOW*:** this directive is introduced to enhance the design throughput by allowing the data to be transformed in sequential order from one loop flow to the next loop. If a next loop uses the data generated in the previous loop, it is not necessary to wait for the end of the previous loop. They will immediately move to the next loop when the data is generated. This directive parallels the communication between different functions. In this paper, FIFO memory is the default port type of FFT IP core. Two sequential loops

are required to transfer the data from BRAM to FIFO (Read) and to load the data from FIFO to BRAM (Write).

***PIPELINE*:** while the DATAFLOW directive orients the parallel communication between the different functions, the PIPELINE directive orients the parallel communication between the different operations of the same function. The DATAFLOW and the PIPELINE directives allow executing all operations simultaneously. In this paper, for each FFT IP core, the PIPELINE directive is used to map data from BRAM to FIFO, but it is necessary to use with the DATAFLOW directive, otherwise, it would not benefit. In addition, the pipeline directive in this project can be implemented using the *#pragma HLS PIPELINE directive [II = <N>]* where N is the number of clock cycles per pipeline. Fig.8 presents how dataflow and pipeline optimizations works.

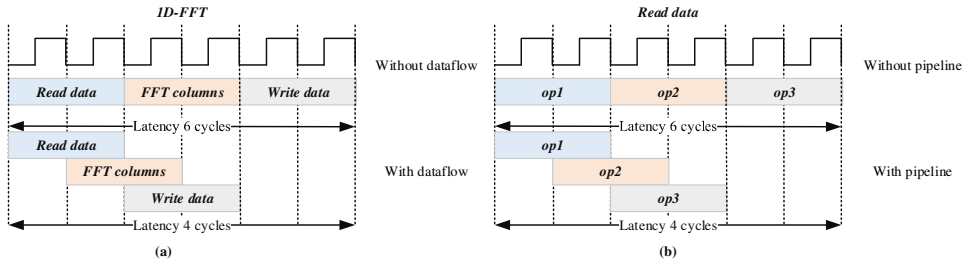


Fig. 8. (a) Hierarchy of the dataflow optimization and (b) Hierarchy of the pipeline optimization.

***UNROLL*:** this directive allows alliterations to run in parallel. This directive has the ability to perform each iteration in one clock cycle when the loop is complete unrolled, and in a separate clock cycle when the loop is rolled. More directive need to be used in order to target real-time applications and to cover more complex designs. Table 1 presents the key features of six widely-used synthesis directives provided by Vivado HLS[30].

Table 1 Configuration of synthesis directives.

Directives	**Target**	**Configuration**
Dataflow	Top loop/function	Enabled/Disabled
Pipeline	Each loop level	Enabled/Disabled
Loop unroll	Each loop level	Unrolling factors
Function inline	Each function	Yes/No
Loop flatten	Inter loop levels	Yes/No
Function pipeline	Each function	Enabled/Disabled

5.2. Evaluation

Table 2 presents the resources utilization of the hardware implementation for the Traditional and the optimized RC algorithms based on Zynq 7z020 FPGA for frame at a resolution of 256x256 pixels. The maximum frequency that can be achieved for this design is 166.67 MHz. Vivado HLS was used to get these resources.

Table 2 Resource utilization of the hardware implementation for the Traditional RC and the optimized RC algorithms based on Zynq 7z020 FPGA with frequency 166.67 MHz.

Functions/ Resources		**BRAM**	**DSP**	**FF**	**LUT**
Traditional RC Algorithm	*Total*	290	79	33516	27232
	Available	280	220	106400	53200
	Utilization (%)	103	35	31	51
Optimized RC Algorithm	*Total*	30	73	32805	26514
	Available	280	220	106400	53200
	Utilization (%)	10	33	30	49

In general, lower resources utilization means lower design cost. To determine the surface usage of an FPGA design, the use of resources is a critical metric. Therefore, a better design should use less resources. According to the comparison, the optimized RC algorithm uses less resource than the traditional RC algorithm. The most obvious difference between both algorithms is that the optimized RC algorithm is introduced without transpose operation, which requires a high number of BRAM and DSP. Consequently, the utilization ratio is greater than 100%. After the first 1D-FFT algorithm, the data stored in the local memory are transposed and stored for another time. This caused an increase in the number of memories used by the transposition operation (256 BRAMs are used). DSPs show the complexity of each algorithm in terms of the number of arithmetic operations, logic operations, and shift operations. The optimized RC algorithm needs one reading/writing transaction between the local memories for each block 1D-FFT, while the traditional algorithm needs three reading/writing transaction between the local memories: two transactions for the two blocks 1D-FFT and one transaction for transposition operation. Thus, the traditional algorithm requires more resources, which increases complexity. Besides, Table 3 introduces the two performance parameters Latency and Timing. They used to measure the effectiveness of any vision application. These two parameters are used to verify the hardware acceleration performance based on the FPGA. Timing represents the processing time, while the latency represents the number of iterations necessary to

process a frame. Therefore, it is possible to compute the processing time of each step for hardware implementation. As well as, the processing time is computed as follows:

$$t_{frame} = \text{Max Latency} * \text{Max Ti min g} \quad (3)$$

Table 3 Performance of the hardware implementation for Traditional RC and optimized RC algorithms based on Zynq 7z020 FPGA.

Functions/ Resources	**Latency**		**Max Timing**
	min	**max**	**(ns)**
Traditional RC Algorithm	3215	65556	5.55
Optimized RC Algorithm	3550	8422	5

The results show a required clock period of 5.55ns and 5ns for the traditional and the optimized RC algorithms, respectively. In addition, the traditional and optimized algorithms require respectively 65556 and 8422 number of cycles to process a frame at a resolution of 256x256 pixels. Thus, the traditional RC algorithm needs high number of cycles and more resources caused by the transpose operation. According to the comparison, the optimized RC algorithm requires less number of cycles and less resources than the traditional RC algorithm. Therefore, it is better to choose an algorithm that needs less number of cycles and less resources to respect real-time execution and memory consumption constraints. As a result, the optimized RC algorithm should be made applicable to FPGA.

Four architectures are used to implement the FFT IP: Radix-4 Burst I/O, Radix-2 Burst I/O, Radix-2 Lite Burst I/O, and Pipelined Streaming I/O. Table 4 shows a comparison between these architectures in order to choose the best architecture in terms of resources used and processing time for the hardware implementation of the optimized RC algorithm to process frame at a resolution of 256x256 pixels.

Table 4 Comparison between the four architectures used for the hardware implementation of the optimized RC algorithm in terms of resource used and the number of cycles.

Functions/ Resources		**BRAM**	**DSP**	**FF**	**LUT**	**Latency**	
						min	**max**
Radix-4 Burst I/O	*Total*	56	49	32803	26514	3788	8660
	Available	280	220	106400	53200		
	Utilization (%)	20	22	30	49		
Radix-2 Burst I/O	*Total*	40	17	32803	26514	7686	12558
	Available	280	220	106400	53200		
	Utilization (%)	14	7	30	49		
Radix-2 Lite Burst	*Total*	40	9	32803	26514	12680	17552
	Available	280	220	106400	53200		

I/O	*Utilization (%)*	14	4	30	49		
Pipelined Streaming I/O	*Total*	30	73	32805	26514		
	Available	280	220	106400	53200	3550	8422
	Utilization (%)	10	33	30	49		

Radix-4 Burst I/O architecture loads and processes the data separately. This architecture uses an iterative approach to process data with a large number of butterflies. It takes latency (8660 number of cycles) but uses more resources (20% of BRAM). The Radix-2 Burst I/O architecture uses the same principle as Radix-4, but with a smaller butterfly, which increases the latency (12558 latency). The Radix-2 Lite Burst I / O architecture is a variant of the Radix-2 Burst I / O, which uses a time-multiplexed approach. This architecture used on butterflies that are composed of a smaller core. It means that it uses less resources (4% of DSP), but with higher latency (17552 number of cycles). Finally, Pipelined Streaming I/O architecture provides continuous data processing. This architecture connects many Radix-2 butterflies to produce a higher throughput and uses less resources. The resources used with the Pipelined Streaming I/O architecture are less than the other three architectures including Radix-4 Burst I/O, Radix-2 Burst I/O, and Radix-2 Lite Burst I/O architectures. Pipelined Streaming I/O architecture requires less resources and less latency. Therefore, it has been used for the hardware implementation of the optimized RC algorithm. Based on the pipelined streaming architecture, the DATAFLOW and PIPELINE directives, the inputs data processed in sequential order from the FIFOs, also, the outputs data of each radix-2 butterfly are written in sequential order. Table 5 presents the effects of the directives based on the Pipelined Streaming I/O architecture on the use of resources and the performance.

The real contribution is to find the best trade-off between FPGA cost and gained performances in term of latency. The the number of resources consumed is a crucial metric to determine the area usage of an FPGA design. The directives effects on BRAM and DSP usage are shown in Fig. 9.1, while dircetives effects on FF and LUT usage are shown in Fig. 9.2. The increase in resource utilization is probably due to several simultaneous function calls, caused by the multiple directives optimization. However, Fig.9.3. show the effectiveness of our included multiple directives optimization solution in term of decreasing latency over than 356x. Table 5 presents the number of resources used of each step for the hardware implementation of the optimized RC algorithm based on FPGA to generate one frame at a resolution of 256x256 pixels. Also, it introduces the Timing and Max Latency.

Fig.10 introduces the processing time of each step for the software and the Hw/Sw Co-design implementations. The Hw/Sw co-design implementation of the optimized RC algorithm is accelerated by integrating the grayscale conversion function and the 2D-FFT algorithm into the FPGA.

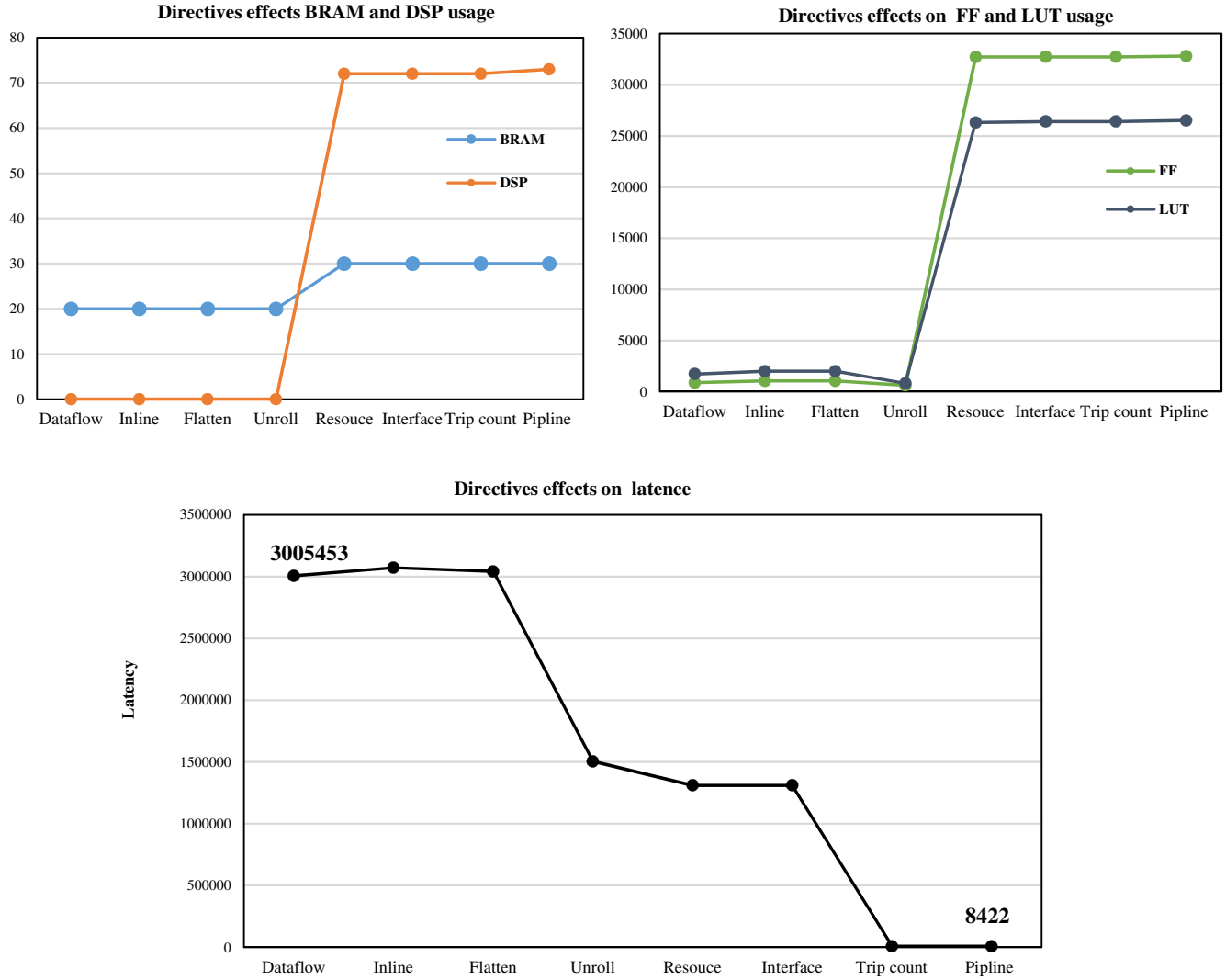


Fig. 9: Analysis of the directives effects: Fig.(9.1) Directives effects on BRAM and DSP usage, Fig.(9.2) dircetives effects on FF and LUT usage, Fig.(9.3) Directives effects on latence.

Table 5 Resource utilization and timing of each step for the hardware implementation of the optimized RC algorithm based on FPGA.

Functions/ Resources		BRAM	DSP	FF	LUT	Timing (ns)	Max Latency
AXIvideo2Mat		0	0	259	245	3.66	67331
Gray scale conversion		0	3	671	676	5. 55	69633
Matrix To Real part		0	0	35	46	5	66305
First 1D-FFT	*Read data (In columns)*	0	0	62	58	5	1121
	Proposed FFT IP core (In columns)	5	36	16104	12805	5	3196

	Write data (In rows)	0	1	121	144	4.8	4101
	Read data (In rows)	0	0	62	58	5	1121
Second 1D-FFT	*Proposed FFT IP core (In columns)*	5	36	16104	12805	5	3196
	Write data (In columns)	0	0	117	47	4.8	1121
Real and Imaginary parts To Matrix		0	0	35	46	5	66305
Mat2AXIvideo		0	0	131	116	2.57	66561

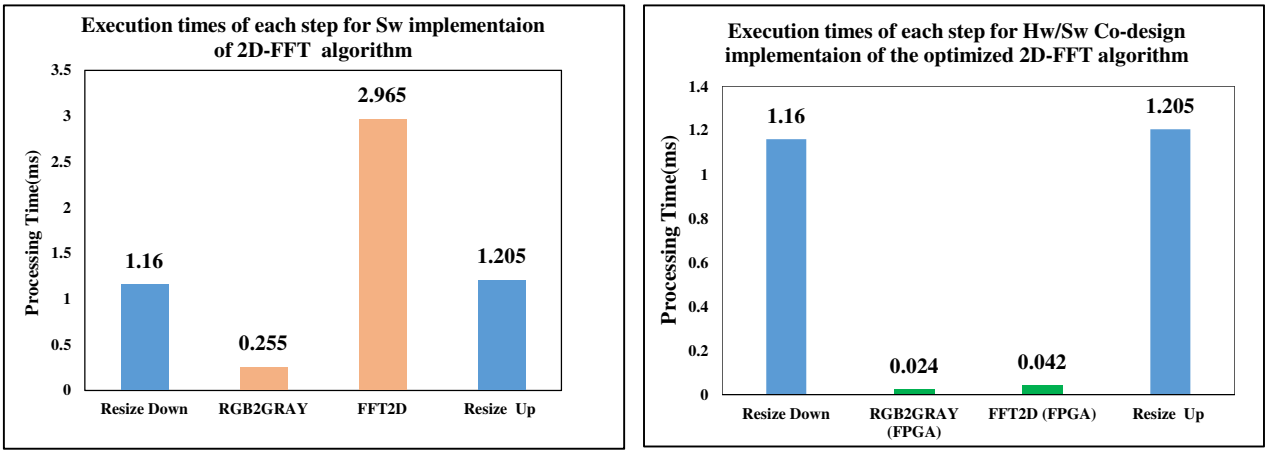


Fig.10 The processing time of each step for the 2D-FFT algorithm based on ARM Cortex-A9 processor, and a hybrid platform ARM combining an ARM processor with an FPGA.

The hardware acceleration of the gray-scale conversion function and 2D-FFT algorithm achieve a speedup of 10.62x and 72.59x, respectively, compared to the software implementation based on the ARM Cortex 9 processor. The hardware implementation based on FPGA means higher performance than software implementation. This higher performance presents the interest of a high-level synthesis using Zynq SoC. In addition, the software implementation of the 2D-FFT function can take advantage of the SIMD engine.

The results of the test bench obtained with HLS will be compared to the MATLAB implementation of each step of the optimized RC algorithm to verify that the Hw/Sw co-design implementation works correctly. A similar implementation of the 2D-FFT algorithm has been introduced in Matlab. Due to the simplicity and benefits of implementing multidimensional arrays, the implementation with Matlab was much shorter and easier in terms of development time. Note that the optimized RC algorithm has been computed is a pipeline which processes the input frame as a continuous stream of pixel values. When calculating PSNR between the results of the MATLAB implementation with the proposed architecture using Vivado HLS must be the same. Therefore, it is proven that the hardware implementation of the optimized RC algorithm works correctly. To compute the PSNR, we use an image of 256x256 pixels. The optimized RC algorithm has been introduced. The $(X_{fl}(k))$ is the 2D-FFT exact with a

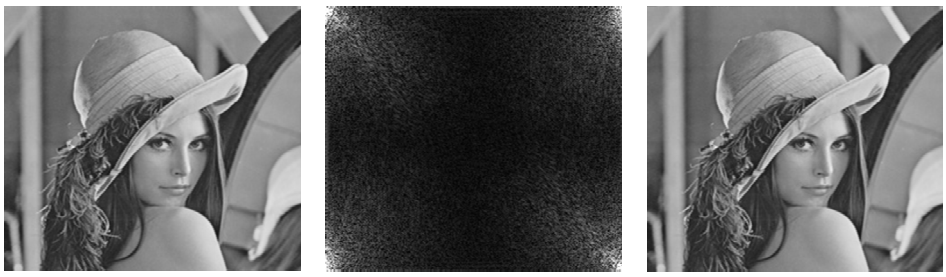
floating-point arithmetic obtained by Matlab 64-bit precision. The $(X_{fx}(k))$ is obtained with the proposed architecture. Table 6 shows the comparison between PSNR obtained by the hardware implementation and the MATLAB implementation. The PSNR is defined by:

$$PSNR(db) = 10\log 10 \frac{\left(\sum_k |X_{fk}(k)|^2\right)}{\left(\sum_k |X_{fl}(k)|^2 - \sum_k |X_{fx}(k)|^2\right)} \quad (4)$$

Table 6 Comparison results between Hw/Sw co-design implementation with Matlab for Lena image at 256*256 resolution.

Functions	Optimized RC algorithm (PSNR)
Abs(2D-FFT)	36,1576
2D-IFFT	∞

The original and reconstructed images obtained using the Hw/Sw co-design implementation for the optimized RC algorithm based on a hybrid platform combining an ARM Cortex 9 processor and an FPGA using Zynq SoC are shown in Fig. 11. It can be seen that the PSNR value between the Hw/Sw co-designs for the optimized RC algorithm using Zynq SoC and the 2D-FFT algorithm using Matlab is greater than 36 dB and ∞ when compared the inverse 2D-FFT.



(a) **(b)** **(c)**

Fig. 11 (a) shows the Lena image, (b) shows the spectrum image obtained by the optimized RC algorithm using Zynq SoC, and (c) shows the reconstructed Lena image.

6. Discussion

In this work, three implementations (one software and two Hw/Sw co-designs) are used to assess the performance of the 2D-FFT algorithm: software implementation based on ARM Cortex-A9 processor and two Hw/Sw Co-design implementations based on a hybrid platform combining an ARM processor and FPGA circuit. Many optimizations are used to improve the evaluation for each implementation. The software

implementation based on ARM Cortex-A9 processor provides advantages in terms of flexibility of the programmability based on the well-known OpenCV libraries. However, the HW/SW co-design flow of the traditional or optimized RC algorithms needs an intermediate evaluation of sub-program, which would be very fastidious in terms of programmability. Then, we should mention that numerous directives have been used to get parallel processing. Consequently, the Hw/Sw co-design implementation provides a better compromise between the performance and the programmability. Moreover, an RTL simulation for the Hw/Sw Co-design solution is evaluated to give detailed information about the number of resources used, Max Latency, and the timing. In addition, many optimizations were introduced. The resizing down/up and the grayscale conversion functions from the OpenCV are used to reduce the number of pixels treated. Consequently, the complexity and the processing time of the 2D-FFT algorithm are reduced.

Since the traditional RC algorithm requires more memory and more dedicated hardware blocks due to the transpose operation. In this paper, an innovative architecture namely, "optimized RC algorithm" using the Zynq SoC is proposed. It implemented without transpose operation. As a result, the optimized RC algorithm uses less memory and less dedicated hardware blocks and requires less Latency, which is suitable for the implementation in FPGA. In addition, many directives such as DATAFLOW, PIPELINE, and LOOP UNROLL are used to improve the performance. Hence, these directives are used to make the optimized RC algorithm more suitable for real-time implementation. The combination of pipelined streaming architecture and the DATAFLOW and PIPELINE directives provides real-time constraints. Table 7 summarizes the results obtained from the different implementations.

Table 7 Processing time for different implementations.

Functions/ Resources	**Optimizations**	**Processing Time (ms)**	**Frame Size**
Softawre implementation	*Original implementation without optimizations(ARM)*	229	1980x1020
	Original implementation with optimizations (ARM)	7.9	1980x1020
	Gray scale conversion + 2D-FFT algorithm (ARM)	3.22	256x256
First HW/SW Co-design implementation	*HW/SW Co-design implementation with optimizations (ARM+FPGA)*	2.7	1980x1020
	Gray scale conversion + traditional RC algorithm (FPGA)	0.384	256x256
Second HW/SW Co-design implementation	*HW/SW Co-design implementation with optimizations (ARM+FPGA)*	2.38	1980x1020
	Gray scale conversion + optimized RC algorithm (FPGA)	0.066	256*256

The experiment results demonstrate that the proposed Hw/Sw co-design implementation based on the optimized RC algorithm is 1.13x, 3.31x and 96.21 faster than the Hw/Sw co-design implementation based on the traditional RC algorithm, pure software implementations with and without optimizations, respectively. The hardware implementation of the grayscale conversion function and the Optimized RC algorithm performed on FPGA is 48.78x faster than the corresponding functions performed on ARM Cortex-A9 processor for a frame of 256x256 pixels. While the hardware implementation of the grayscale conversion function and the Traditional RC algorithm provide an acceleration of 8.38x. In addition, the hardware implementation of the Optimized and the Traditional RC algorithms produces an acceleration of 70.59x and 8.23x, respectively, compared to the corresponding functions performed on ARM processor. Therefore, the optimized architecture for the 2D-FFT algorithm significantly improves the resources uses (BRAM, DSP, FF, and LUT) and the processing time (Timing and Latency). As depicted in table 8, the proposed design of the 2D-FFT algorithm takes less number of clock cycles for execution and requires less processing time compared to existing FPGA-based implementations for an image at a resolution of 256x256 pixels.

Table 8 2D FFT performance comparison with existing FPGA-based implementations for an image at a resolution of 256x256 pixels.

References	FPGA used	Architecture	Frequency (MHz)	Clock cycles	Processing time
[26] (2010)	Xilinx Virtex-5	Pipelined Streaming	100	605,000	6.05 (ms)
[29] (2011)	Spartan -3	Radix-4	91	---	10.1(s)
[8] (2013)	Spartan-3	Radix-4	103	---	3.4 (s)
[27](2018)	XC7K410T	---	80	---	1.24 (ms)
Proposed (2019)	Zynq 7z020	Pipelined Streaming	166.67	8422	0.042 (ms)

7. Conclusion

This work presents three implementations (one software and two Hw/Sw co-designs) of the 2D-FFT algorithm introduced by Cooley and Tukey. All of them have been implemented on ARM Cortex-A9 processor and FPGA using the Zynq SoC. The design was realized using the Vivado HLS. To begin, a software implementation of the 2D-FFT algorithm is introduced on ARM Cortex-A9 processor. After that, a Hw/Sw co-design solution of the traditional 2D-FFT algorithm is introduced on a hybrid platform

combining an ARM Cortex-A9 processor with an FPGA. The traditional 2D-FFT algorithm uses more resources and more processing time due to the transpose operation. Therefore, to respect real-time execution and memory consumption constraints, an optimized architecture for the 2D-FFT algorithm using the Zynq SoC is proposed on FPGA. It provides an acceleration of 70.59x, while the traditional 2D-FFT algorithm provides an acceleration of 8.23x compared to the corresponding functions performed on ARM processor. Several directives are applied to improve the performance. About the processing time of the whole system, the proposed Hw/Sw co-design implementation based on the optimized RC algorithm is 1.13x, 3.31x and 96.21 faster than the Hw/Sw co-design implementation based on the traditional RC algorithm, pure software implementations with and without optimizations, respectively.

References

[1] Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, *19*(90), 297-301.

[2] Yan, Y., Wang, H., Li, C., Yang, C., & Zhong, B. (2013). An effective unconstrained correlation filter and its kernelization for face recognition. *Neurocomputing*, *119*, 201-211.

[3] Wang, Q., Alfalou, A., & Brosseau, C. (2017). New perspectives in face correlation research: a tutorial. *Advances in Optics and Photonics*, *9*(1), 1-78.

[4] Ouerhani, Y., Jridi, M., & Alfalou, A. (2010, July). Fast face recognition approach using a graphical processing unit "GPU". In *2010 IEEE International Conference on Imaging Systems and Techniques* (pp. 80-84). IEEE.

[5] Cheng, K. M., Lin, C. Y., Chen, Y. C., Su, T. F., Lai, S. H., & Lee, J. K. (2013, October). Design of vehicle detection methods with opencl programming on multi-core systems. In *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*(pp. 88-95). IEEE.

[6] Govindaraju, N. K., Lloyd, B., Dotsenko, Y., Smith, B., & Manferdelli, J. (2008, November). High performance discrete Fourier transforms on graphics processors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (p. 2). IEEE Press.

[7] Smach, F., Miteran, J., Atri, M., Dubois, J., Abid, M., & Gauthier, J. P. (2007). An FPGA-based accelerator for Fourier Descriptors computing for color object recognition using SVM. *Journal of Real-Time Image Processing*, *2*(4), 249-258.

[8] Ouerhani, Y., Jridi, M., & Alfalou, A. (2012). Area-Delay Efficient FFT Architecture Using Parallel Processing and New Memory Sharing Technique. *Journal of Circuits, Systems, and Computers*, *21*(06), 1240018.

[9] Jothi, R. A., & Palanisamy, V. (2018). Performance Enhancement of Minutiae Extraction Using Frequency and Spatial Domain Filters. *International Journal of Pure and Applied Mathematics*, *118*(7), 647-654.

[10] Jridi, M., Napoléon, T., & Alfalou, A. (2018). One lens optical correlation: application to face recognition. *Applied optics*, *57*(9), 2087-2095.

[11] Ouerhani, Y., Alfalou, A., & Brosseau, C. (2017, August). Road mark recognition using HOG-SVM and correlation. In *Optics and Photonics for Information Processing XI* (Vol. 10395, p. 103950Q). International Society for Optics and Photonics.

[12] Napoléon, T., & Alfalou, A. (2014, May). Local binary patterns preprocessing for face identification/verification using the VanderLugt correlator. In *Optical Pattern Recognition XXV* (Vol. 9094, p. 909408). International Society for Optics and Photonics.

[13] Yanqing, D., Guoqing, Y., & Yanjie, Z. (2017). Remote Sensing Image Content Retrieval Based on Frequency Spectral Energy. *Procedia Computer Science*, *107*, 448-453.

[14] Lamas-Seco, J., Castro, P., Dapena, A., & Vazquez-Araujo, F. (2015). Vehicle classification using the discrete fourier transform with traffic inductive sensors. *Sensors*, *15*(10), 27201-27214.

[15] Yang, W., Wang, S., Hu, J., Zheng, G., & Valli, C. (2018). A fingerprint and finger-vein based cancelable multi-biometric system. *Pattern Recognition*, *78*, 242-251.

[16] Dehai, Z., Da, D., Jin, L., & Qing, L. (2013, November). A PCA-based face recognition method by applying fast fourier transform in pre-processing. In *3rd International Conference on Multimedia Technology (ICMT-13)*. Atlantis Press.

[17] Zhang, D., Ding, D., Li, J., & Liu, Q. (2014). A novel way to improve facial expression recognition by applying fast fourier transform. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. 1).

[18] Zhang, C., & Prasanna, V. (2017, February). Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 35-44). ACM.

[19] Chen, S., & Li, X. (2014, June). Input-adaptive parallel sparse fast fourier transform for stream processing. In *Proceedings of the 28th ACM international conference on Supercomputing* (pp. 93-102). ACM.

[20] Hyun, E., Kim, S. D., Ju, Y. H., Lee, J. H., You, E. N., Park, J. H., ... & Kim, S. G. (2011, October). FPGA based signal processing module design and implementation for FMCW vehicle radar systems. In *Proceedings of 2011 IEEE CIE International Conference on Radar* (Vol. 1, pp. 273-275). IEEE.

[21] Walid, K., & Sajed, M. (2006). Design and implementation of a RADIX-4 FFT using FPGA technology. *IFAC Proceedings Volumes*, *39*(21), 248-252.

[22] Raju, K. S., Sengar, V., Gangal, M., Tanwar, P., & Prasad, P. B. Hardware Implementation of Discrete Fourier Transform and its Inverse Using Floating Point Numbers.

[23] Lenart, T., Gustafsson, M., & Öwall, V. (2008). A hardware acceleration platform for digital holographic imaging. *Journal of Signal Processing Systems*, *52*(3), 297-311.

[24] Crockett, L. H., Elliot, R. A., Enderwitz, M. A., & Stewart, R. W. (2014). *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media.

[25] Kortli, Y., Marzougui, M., Bouallegue, B., Bose, J. S. C., Rodrigues, P., & Atri, M. (2017, March). A novel illumination-invariant lane detection system. In *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)* (pp. 166-171). IEEE.

[26] Yu, C. L., Chakrabarti, C., Park, S., & Narayanan, V. (2010, March). Bandwidth-intensive FPGA architecture for multi-dimensional DFT. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 1486-1489). IEEE.

[27] Li, L., & Wyrwicz, A. M. (2018). Parallel 2D FFT implementation on FPGA suitable for real-time MR image processing. *Review of Scientific Instruments*, *89*(9), 093706.

[28] Rios-Navarro, A., Tapiador-Morales, R., Jimenez-Fernandez, A., Amaya, C., Dominguez-Morales, M., Delbruck, T., & Linares-Barranco, A. (2018, July). Performance evaluation over HW/SW co-design SoC memory transfers for a CNN accelerator. In *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)* (pp. 1-4). IEEE.

[29] Ouerhani, Y., Jridi, M., & Alfalou, A. (2011, August). Implementation techniques of high-order FFT into low-cost FPGA. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)* (pp. 1-4). IEEE.

[30] Zhao, J., Feng, L., Sinha, S., Zhang, W., Liang, Y., & He, B. (2019). Performance Modeling and Directives Optimization for High Level Synthesis on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.