electronics

MDPI

Article

Towards On-Board SAR Processing with FPGA Accelerators and a PCIe Interface

Emilio Isaac Baungarten-Leon [1], Gustavo Daniel Martín-del-Campo-Becerra [2], Susana Ortega-Cisneros [1,*], Maron Schlemon [2], Jorge Rivera [1] and Andreas Reigber [2]

[1] Center for Research and Advanced Studies (CINVESTAV) of the National Polytechnic Institute (IPN), Guadalajara 45019, Mexico; emilio.baungarten@cinvestav.mx (E.I.B.-L.); jorge.rivera@cinvestav.mx (J.R.)
[2] Microwaves and Radar Institute (HR), German Aerospace Center (DLR), 82234 Weßling, Germany; gustavo.martindelcampobecerra@dlr.de (G.D.M.-d.-C.-B.); maron.schlemon@dlr.de (M.S.); andreas.reigber@dlr.de (A.R.)
* Correspondence: susana.ortega@cinvestav.mx; Tel.: +52-33-3777-3600

Abstract: This article addresses a novel methodology for the utilization of Field Programmable Gate Array (FPGA) accelerators in on-board Synthetic Aperture Radar (SAR) processing routines. The methodology consists of using High-Level Synthesis (HLS) to create Intellectual property (IP) blocks and using the Reusable Integration Framework for FPGA Accelerators (RIFFA) to develop a Peripheral Component Interconnect express (PCIe) interface between the Central Processing Unit (CPU) and the FPGA, attaining transfer rates up to 15.7 GB/s. HLS and RIFFA reduce development time (between fivefold and tenfold) by using high-level programming languages (e.g., C/C++); moreover, HLS provides optimizations like pipeline, cyclic partition, and unroll. The proposed schematic also has the advantage of being highly flexible and scalable since the IPs can be exchanged to perform different processing routines, and since RIFFA allows employing up to five FPGAs, multiple IPs can be implemented in each FPGA. Since Fast Fourier Transform (FFT) is one of the main functions in SAR processing, we present a FPGA accelerator in charge of the reordering stage of VEC-FFT (an optimized version of FFT) as a proof of concept. Results are retrieved in reversed bit order, and the conventional reordering function may consume more than half of the total clock cycles. Next, to demonstrate flexibility, an IP for matrix transposition is implemented, another computationally expensive process in SAR due to memory access.

Keywords: Field Programmable Gate Array (FPGA); High-Level Synthesis (HLS); Peripheral Component Interconnect express (PCIe); Reusable Integration Framework for FPGA Accelerators (RIFFA); Synthetic Aperture Radar (SAR)

Citation: Baungarten-Leon, E.I.; Martín-del-Campo-Becerra, G.D.; Ortega-Cisneros, S.; Schlemon, M.; Rivera, J.; Reigber, A. Towards On-Board SAR Processing with FPGA Accelerators and a PCIe Interface. *Electronics* **2023**, *12*, 2558. https://doi.org/10.3390/electronics12122558

Academic Editors: Dariusz Kania, Alexander Barkalov, Remigiusz Wiśniewski and Larysa Titarenko

Received: 20 April 2023
Revised: 22 May 2023
Accepted: 23 May 2023
Published: 6 June 2023

1. Introduction

Synthetic Aperture Radar (SAR) systems have been extensively used for Earth remote sensing. They provide high-resolution, light- and weather-independent reconstructions for various applications, including climate and environmental change research and Earth system monitoring [1]. Similar to conventional radar, electromagnetic waves (in the form of a series of short pulses) are transmitted from a spaceborne or airborne platform, backscattered, and finally collected by the receiving antennas. The combination of the echo signals, received over a period of time, allows for the construction of a virtual aperture much longer than the physical antenna length [2].

SAR systems produce large amounts of raw data, which needs to be processed for image reconstruction. Due to limited on-board processing capacities (e.g., power, size, weight, cooling, communication bandwidth, etc.) on SAR platforms, the raw data is commonly sent to the ground station for processing. Nevertheless, since the volume of raw data and computational load of modern SAR systems have been increasing meaningfully, downlinking the large data throughput has become a major bottleneck. For instance, to attain a

0.5 m × 0.5 m resolution, the F-SAR system from the German Aerospace Center (DLR) [3] acquires SAR raw data with approximately 285,000 × 32,768 (azimuth/range) samples per channel. Assuming the complex64 (8-byte) format, each channel results in approximately 70 GB of raw data. With three channels for Maritime Moving Target Indication (MMTI), approximately 210 GB are processed. Seeking to ease this problem, the generation of on-board imagery has been conducted; however, constraints on computational performance, data size, and transfer speeds must be tackled.

Previous related studies have approached the issue in different ways. The work presented in [4] addresses real-time on-board SAR imaging by means of a Field Programmable Gate Array (FPGA) together with a Digital Signal Processor (DSP). The application requirements include an optimal balance between processing delay, throughput, appropriate data format, and circuit scale. The DSP is used to implement auxiliary functions, while the FPGA is employed to implement the main processing flow of the Chirp Scaling (CS) algorithm [5,6], which makes use of the Fast Fourier Transform (FFT)/Inverse FFT (IFFT). Pipeline optimization is applied to the FFT to improve processing efficiency. Long development times, however, are emphasized as one of the main challenges.

The work addressed in [7] presents a heterogeneous array architecture for SAR imaging. An Application Specific Instruction Set Processor (ASIP) is utilized, attaining Giga operations per second. The research seeks to comply with the desired power consumption and lists the main advantages and disadvantages of the technology against other technologies.

i. Central Processing Units (CPUs) are flexible and portable; however, their power efficiency is quite low, a bottleneck for real-time SAR applications.
ii. Graphics Processing Units (GPUs) provide powerful parallel computation and programmability. However, the average power consumption (up to 150 W) limits the application of GPUs for on-board processing.
iii. A multi-DSP architecture allows for the performance of many complex theories and algorithms on hardware. Nevertheless, it entails low power efficiency.
iv. FPGAs possess rich on-chip memory; moreover, computational resources are configurable to meet SAR signal processing requirements, e.g., high throughput rate, desired operations per second, and power consumption. However, the development cycle of FPGAs is relatively long.

The work in [8] addresses real-time SAR imaging systems and focuses on the data format. Specifically, after assessing the advantages offered by fixed-point data processing, the authors propose a solution based on the utilization of a System-on-Programmable-Chip (SoPC), implemented in a Zynq+NetFPGA platform. The use of SoPC is due to high-performance embedded computing. System C is employed to develop the Register Transfer Level (RTL) code.

Concerning spatial-grade devices, [9] presents a comparison between different spatial-grade CPUs, DSPs, and FPGAs. The research demonstrates and quantifies how emerging space-grade processors are continually increasing the capabilities of space missions by supporting high levels of parallelism in terms of computational units. The considered processors include multicore and many-core CPUs (HXRHPPC, BAE Systems RAD750, Cobham GR712RC, Cobham GR740, BAE Systems RAD5545, and Boeing Maestro), DSPs (Ramon Chips RC64 and BAE Systems RADSPEED), and FPGA architectures (Xilinx Virtex-5QV FX130 and Microsemi RTG4). GPUs are excluded since there are no space-grade GPUs. In terms of integer Computational Density (CD) and CD/W, the best results are attained by RC64, Virtex-5QV, and RTG4. In terms of Internal Memory Bandwidth, the best results are achieved by the RC64 and the Virtex-5QV. In terms of External Memory Bandwidth, the best results are attained by the RAD5545 and the Virtex-5QV. Lastly, in terms of Input/Output Bandwidth, best results are achieved by RAD5545, Virtex-5QV, and RTG4.

The research in [10] explores the utilization of Qualcomm's Snapdragon System on a Chip (SoC) technology in space missions. Specifically, it focuses on the successful de-

ployment of the Snapdragon 801 SoC in the Ingenuity Helicopter on Mars and the use of Snapdragon 855 development boards in the International Space Station. The study compares different GPUs such as Nvidia Jetson Nano, Nvidia TX2, Nvidia GTX 560M, and Nvidia GTX 580; it also highlights that GPUs are not commonly used for space computing. The study refers to traditional FPGA implementations with the VIRTEX-5 SX50T FPGA as a benchmark. Interestingly, the research demonstrates that, in certain scenarios, the software implementation on the Snapdragon SoC outperforms the traditional FPGA implementations. This finding emphasizes the computational power and efficiency offered by Snapdragon technology in the context of space-related applications. However, it should be noted that the FPGA VIRTEX-5 SX50T uses 65 nm technology, while the Snapdragon 855 uses 7 nm technology.

Multiple implementations for on-board SAR processing with small and low-power GPU devices have been realized. In [11], the paper demonstrates the successful implementation of SAR processing algorithms on the Jetson TX1 platform. The optimized implementation takes advantage of the GPU's parallel processing capabilities, resulting in improved performance compared to CPU-based approaches. This highlights the potential of the Jetson TX1 platform for accelerating SAR processing tasks in a compact and energy-efficient manner. In [12], the research presents a small UAV-based SAR system that utilizes low-cost radar, position, and attitude sensors while incorporating on-board imaging capability. The system demonstrates the feasibility of cost-effective SAR imaging using a Nvidia Jetson Nano as the host computer of the drone. The choice of a powerful and energy-efficient platform for data processing and control enhances the system's capabilities. Leveraging the Jetson Nano's GPU capabilities and parallel processing power, the system can perform real-time processing, sensor integration, and image reconstruction tasks.

In both cases [11,12], the amount of processed information differs from the previous example of the three-channel MMTI system. Although [11] does not specify the image dimensions, it mentions a data weight of 172 MB; on the other hand, although [12] does not specify the data weight, it mentions the image dimensions (120 m $\times$ 100 m with a resolution of 0.25 m). As a point of comparison, Ref. [13] presents image dimensions similar to the MMTI example; the article tackles on-board SAR processing using SIMD instructions with an Intel® Core™ i7-3610QE (by Intel Corporation in Santa Clara, CA, USA) processor. The image dimensions are 7.5 km by 2.5 km with a resolution of 0.5 m. Note the large difference between the data weight in [11] (172 MB) and in the MMTI example (210 GB), as well as the image size, 120 m $\times$ 100 m in [12] in contrast to 7.5 km $\times$ 2.5 km.

In general, FPGA-based accelerators provide problem-specific processing solutions that are highly parallelized and reliable. Moreover, there is a wide variety of devices to select from, according to the application requirements. Nonetheless, as discussed previously, there are some concerns regarding the implementation, including a long development time and a low Data Transfer Rate (DTR). However, these issues can be solved effectively by making use of tools like High-Level Synthesis (HLS) [14], employed to reduce development time between fivefold and tenfold [15,16], and Reusable Integration Framework for FPGA Accelerators (RIFFA) [17], utilized to develop a Peripheral Component Interconnect express (PCIe) interface. On the one hand, HLS allows describing Hardware (HW) via high-level programming languages (e.g., C/C++, System C, or OpenCL). Additionally, HLS permits applying optimizations like pipeline, cyclic partition, and unroll. On the other hand, RIFFA provides a framework that works directly with the PCIe endpoint, achieving high data transfer speeds (up to 15.7 GB/s for PCIe 3.0 with 16 lanes). RIFFA runs on Windows and Linux with the programming languages C, C++, Python, MATLAB, and Java [17].

Correspondingly, this article addresses a novel methodology for the utilization of FPGA accelerators in on-board SAR processing routines. The methodology consists of using HLS to create Intellectual Property (IP) blocks and using RIFFA to develop a PCIe interface between the CPU and the FPGA. The proposed schematic has the advantage of being highly flexible and scalable since the IPs can be exchanged to perform different

processing routines and since RIFFA allows employing up to five FPGAs while multiple IPs can be implemented in each FPGA.

The suggested methodology, for example, is suitable for the project described in [4], as each stage of the CS algorithm can be implemented as an IP. Making use of HLS would reduce development time significantly, whereas different optimizations (e.g., pipeline, cyclic partition, or unroll) could be applied to meet application requirements. In contrast to [7], an FPGA implementation has greater flexibility than an ASIP, as it is not limited by an instruction set. Similar to [8], the proposed methodology allows performing the same algorithm for different input/output data formats; the main difference is that our methodology provides further optimization capabilities and PCIe infrastructure. Finally, [9] addresses space-grade FPGA systems (i.e., Virtex-5QV and RTG4) that are configurable with the proposed methodology and are able to process large data throughput efficiently.

As proof of concept, we present a FPGA accelerator in charge of the reordering stage of VEC-FFT [18], a software-optimized version of the FFT. Among the most frequently employed SAR focusing techniques, we can find the range-doppler algorithm [19,20], CS [8,9], and Omega-K (ω-k) [21–23], along with their extended versions. Common to all these methods is the use of the FFT, which has a high computational cost as it involves many additions, subtractions, and multiplications. Consequently, optimizing the FFT enhances the performance of such SAR focusing techniques.

VEC-FFT is based on Radix-4 [24] and executed using Single Instruction Multiple Data (SIMD) [25], achieving high efficiency due to parallel data processing. However, after the FFT is computed, results are retrieved in reversed bit order, meaning that data reordering is required as a further step. VEC-FFT shows better performance in contrast to the Fastest Fourier Transform in the West (FFTW) Scalar [18], a popular C++ FFT library. Furthermore, when the reordering stage is not performed, VEC-FFT results are faster than FFTW SIMD [18]. For example, without data reordering, VEC-FFT solves a 16,384-point FFT utilizing only 91,601 clock cycles, whereas FFTW Scalar and FFTW SIMD entail 452,253 and 106,601 clock cycles, respectively. Nevertheless, VEC-FFT requires 142,460 additional clock cycles to compute the reordering post-processing step, for a total of 234,061 clock cycles. Note that the data reordering function consumes more than half of the total clock cycles.

A HW implementation of the VEC-FFT reordering function using the proposed methodology significantly reduces the number of required clock cycles. By instance, from 142,460 to 44,569, for the example above mentioned. Moreover, the IP can be instantiated multiple times, performing more than one reordering function at a time.

Next, to demonstrate flexibility, the VEC-FFT reordering IP is replaced by an IP for matrix transposition, another computationally expensive process due to memory latency. The matrix transpose is implemented for squared matrices with dimensions of 32×32, 64×64, and 128×128. The addressed CPU implementation utilizes 10,424, 38,505, and 210,928 clock cycles, respectively; conversely, the HW implementation, using the proposed methodology, reduces the number of clock cycles to 1041, 4113, and 16,401, correspondingly.

Finally, the main contributions of this work are summarized as follows:

i. A novel methodology for the utilization of FPGA accelerators in on-board SAR processing routines.
    a. Improvement of development time by making use of HLS and RIFFA, which allow working with high-level programming languages (e.g., C/C++).
    b. Use of the optimizations offered by HLS (e.g., pipeline, cyclic partition, and unroll) to meet application requirements.
    c. High flexibility. The schematic has the advantage of being modular, as the IPs can be exchanged to perform different processing routines.
    d. Increased DTR (up to 15.7 GB/s) via a PCIe interface developed with RIFFA.

e. High scalability. RIFFA permits using up to five FPGAs, while multiple IPs can be implemented in each FPGA.

ii. Proof of concept through the implementation of two FPGA accelerators.

a. Reordering stage of VEC-FFT [18].
b. Matrix transpose; aimed at demonstrating flexibility in the utilization of different IPs.

The remainder of the article is organized as follows: Section 2 reviews HLS and RIFFA; Section 3 explains the integration of RIFFA with the HLS Ips; Section 4 addresses the HLS IP for the reordering stage of VEC-FFT and the HLS IP for matrix transposition; Section 5 assesses the performance of the FPGA against the CPU; Section 6 presents the discussion; and finally, Section 7 concludes this work.

2. Development Tools

2.1. High-Level Synthesis (HLS)

Generally, HW description languages (HDLs) like Verilog and Very High-Speed Integrated Circuit HDL (VHDL) involve long development times since they work at a low level of abstraction. Conversely, HLS offers a simpler and faster solution for HW description as it consists of an automated process. It creates a synthesizable Register Transfer Level (RTL) from an algorithm written with a high-level language (i.e., C/C++, System C, or OpenCL), whose design can be placed and routed in an FPGA.

IP blocks created with HLS are commonly less efficient than those created with HDLs since HLS makes use of state machines [14–16]. Utilizing a state machine means having a sequence of steps, which may result in suboptimal parallelization. In addition, HLS is only recommended when dynamic memory and recursive functions are not necessary. The code of a function is to be translated into logic cells (synthesis); therefore, the number of function calls within a recursive function must be known a priori.

The main reason for choosing HLS over HDLs is to improve productivity and reduce development time concerns, as HLS provides correct-by-construction features. Furthermore, for the purposes of this work, HLS implementations are sufficiently fast and efficient. We refer to the software (SW), Vivado Design Suite [26], and Vivado HLS [27], associated with the ZC706 development board [28]. The workflow depicted in Figure 1 shows the methodology followed for the implementation and validation of a SoC via HLS, resulting in the creation of an IP block.

First, the problem definition and its main limitations are set. Next, HW description is performed by means of the SW Vivado HLS as a translation of high-level programming languages. The next three stages consist of validating whether the system is synthesizable and the functions are logically correct. Synthesis means translating C/C++, System C, or OpenCL code into HDL code, such as Verilog or VHDL. If the previous validation passes, the RTL is wrapped as an IP and exported to the SW Vivado Design Suite for synthesis and further analysis. The resultant IP is treated like any other module developed in HDL. Subsequently, HW integration is performed, and additional functional tests are performed with the physical system. If the system fails, then debugging is performed; if the system works correctly, then its functioning is documented in detail. The workflow in Figure 1 has two validation points, depicted via red rhombuses. Part of the validation consists of comparing the results retrieved by the IP with a gold bench variable.

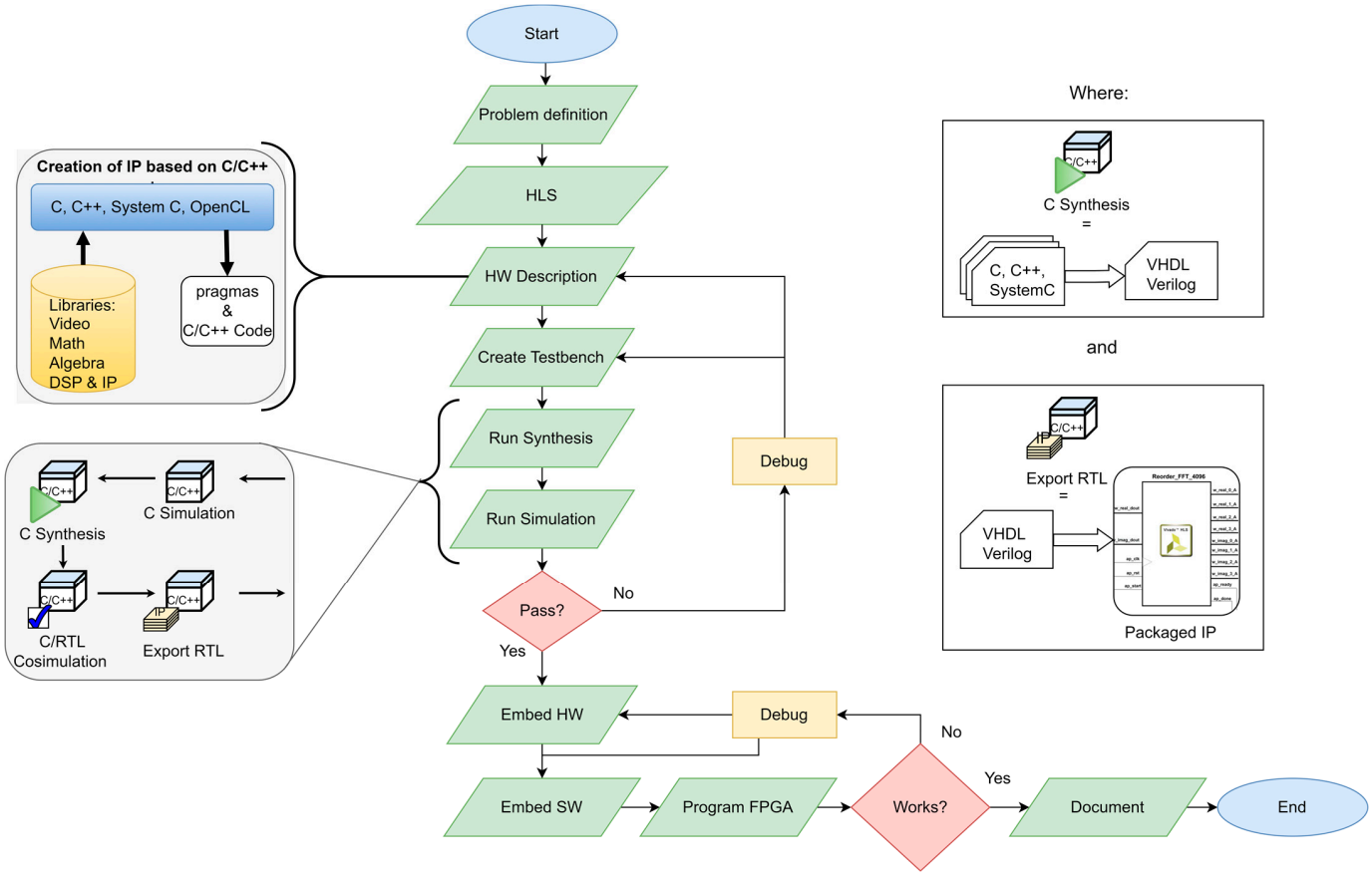

Figure 1. Methodology was followed for the implementation and validation of a SoC via HLS, resulting in the creation of an IP block.

2.2. HLS Optimizations

The HW description process performed by HLS includes different optimizations, known as pragmas. These are aimed at enhancing performance by reducing latency (clock cycles) and resource usage. Since this work pursues reducing the latency of VEC-FFT, we focus on such pragmas, which encompass Pipelining, Unrolling, and Array Partition [29].

- Pipelining divides a loop into multiple stages, which perform different operations. The stages are executed in parallel, each one works simultaneously on different iterations of the loop.
- Unrolling is the process of expanding the iterations of a loop into separate statements to eliminate the overhead associated with the loop control logic. Less instructions per iteration are needed, reducing the number of clock cycles required to complete the loop.
- Array partitioning optimizes the memory usage and performance of software algorithms for implementation on hardware platforms. It involves dividing a large, multi-dimensional array into smaller, manageable partitions that are processed and stored in parallel.

2.3. Reusable Integration Framework for FPGA Accelerators (RIFFA)

RIFFA [17] is an open-source reusable library to connect an FPGA with other devices via PCIe. RIFFA is quite popular, as it can be used in multiple development boards of the main FPGA manufacturers (e.g., Intel and Xilinx). In the following, the main characteristics of RIFFA are addressed [17]. These are divided into two categories: SW and HW.

Software:

- Support for Linux kernels 2.6.27+ and Windows 7 (drivers).

- Implemented with C/C++, Python, MATLAB, or Java.
- Capable of working with multiple FPGAs (up to five) per system.
- Simple functions to Transmit (TX) and Receive (RX) data. For instance, the related C/C++ code for TX/RX is presented next (Scheme 1).

```
int fpga_send(fpga_t * fpga, int chnl, void * data, int len, int destoff, int last, long long timeout);

int fpga_recv(fpga_t * fpga, int chnl, void * data, int len, long long timeout);
```

Scheme 1. C/C++ TX/RX functions provided by RIFFA.

Where:

- `fpga`: Pointer to structure "`fpga_t`".
- `chnl`: Channel number.
- `data`: Pointer to the data array.
- `len`: Data length (32 bits).
- `destoff`: Specifies where to start the data writing.
- `last`: Zero means that the current transfer is not the last in a sequence of transfers. One indicates the last transfer.
- `timeout`: Waiting time value in ms.
- `return`: Number of transmitted/received data.

The function `fpga_send` sends a `len` of data through the FPGA channel `chnl` using the `fpga_t` structure. The values `destoff` and `last` are also sent; `destoff` is used to support the distribution of data across multiple send transactions, whereas `last` indicates whether the data transfers are finished or not. When `last` equals 1, the channel interprets the end of the transaction. Conversely, when `last` equals 0, the channel stands by for additional transactions. Finally, when `timeout` is equal to zero, communication is blocked indefinitely. Multiple threads sending data through the same channel may result in corrupt data or errors. Accordingly, `fpga_send` returns the actual number of words sent. Function `fpga_recv` applies the same principles as `fpga_send`, with the difference that it receives data from the FPGA channel `chnl` in the `data` pointer.

Hardware:

- Works directly with the PCIe endpoint, being capable of saturating the PCIe link.
- Unrequired knowledge of bus addresses, buffer sizes, or PCIe packet formats.
- Independent TX and RX signals.
- Communications model based on Direct Memory Access (DMA) transfers.
- Data is expressed in widths of 32 bits, 64 bits, or 128 bits.
- Well documented.

Figure 2 shows the architectural diagram of RIFFA. Data is transferred through RIFFA's RX/TX modules using addresses provided by the workstation. These engines provide DMA transfer for all RIFFA channels. On the SW side, there is simply one data TX and one data RX function. Different libraries are provided for each supported programming language, i.e., C/C++, Python, MATLAB, and Java.

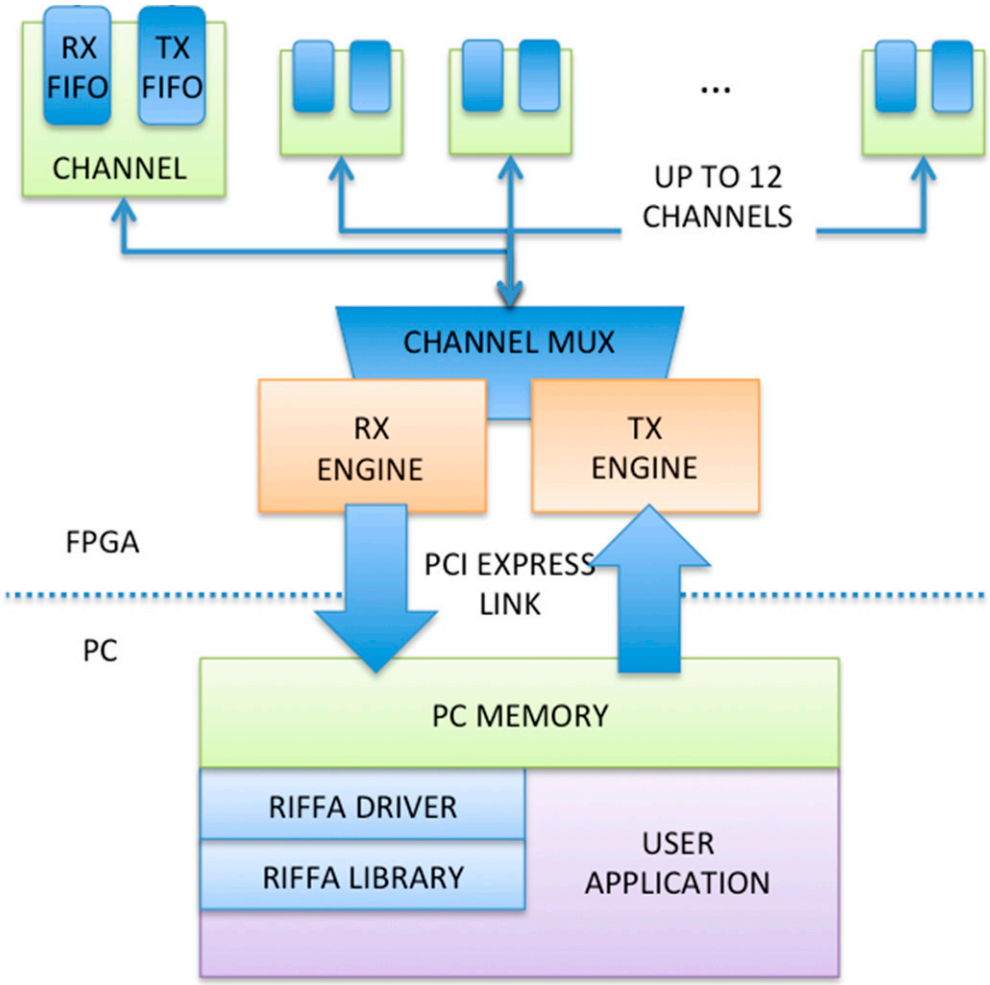

Figure 2. High-level architectural diagram of RIFFA's 2.0 framework [17].

3. Integration of RIFFA

The proposed HW architecture consists of two main elements: RIFFA, responsible for generating the signals utilized by PCIe, and the IPs, whose functionality depends on the application, e.g., the reordering stage of VEC-FFT and matrix transpose. The IPs connect with RIFFA through RIFFA channels, which obey the set of signals presented in Table 1 [17]. These are divided into two subsets: those for RX and those for TX. As shown in Figure 3, the operation of the channels is based on a finite-state machine composed of eight states. The state machine manages the receiving, processing, and sending of data.

Table 1. Control signals to receive (RX) and send (TX) data through RIFFA [17].

RX and TX Control Signals		
RX Engine		
Name	**I/O**	**Description**
CHNL_RX_CLK	O	Provide the clock signal to read data from the incoming FIFO.
CHNL_RX	I	Goes high to signal incoming data. Will remain high until all incoming data is written to the FIFO.
CHNL_RX_ACK	O	Must be pulsed high for at least 1 cycle to acknowledge the incoming data transaction.

Table 1. *Cont.*

RX and TX Control Signals		
RX Engine		
Name	**I/O**	**Description**
CHNL_RX_LAST	I	High indicates this is the last received transaction in a sequence.
CHNL_RX_LEN[31:0]	I	Length of the received transaction in 4-byte words.
CHNL_RX_OFF[30:0]	I	Offset in 4-byte words indicating where to start storing received data (if applicable in design).
CHNL_RX_DATA[DWIDTH-1:0]	I	Receive data.
CHNL_RX_DATA_VALID	I	High if the data on CHNL_RX_DATA is valid.
CHNL_RX_DATA_REN	O	When high and CHNL_RX_DATA_VALID is high, consumes the data currently available on CHNL_RX_DATA.
TX Engine		
CHNL_TX_CLK	O	Provide the clock signal to write data to the outgoing FIFO.
CHNL_TX	O	Set it high to signal a transaction. Keep it high until all outgoing data is written to the FIFO.
CHNL_TX_ACK	I	Will be pulsed high for at least one cycle to acknowledge the transaction.
CHNL_TX_LAST	O	High indicates this is the last send transaction in a sequence.
CHNL_TX_LEN[31:0]	O	Length of the sent transaction in 4-byte words.
CHNL_TX_OFF[30:0]	O	Offset in 4-byte words indicating where to start storing sent data in the PC thread's receive buffer.
CHNL_TX_DATA[DWIDTH-1:0]	O	Send data.
CHNL_TX_DATA_VALID	O	Set high when the data on CHNL_TX_DATA is valid. Update when CHNL_TX_DATA is consumed.
CHNL_TX_DATA_REN	I	When high and CHNL_TX_DATA_VALID is high, consumes the data currently available on CHNL_TX_DATA.

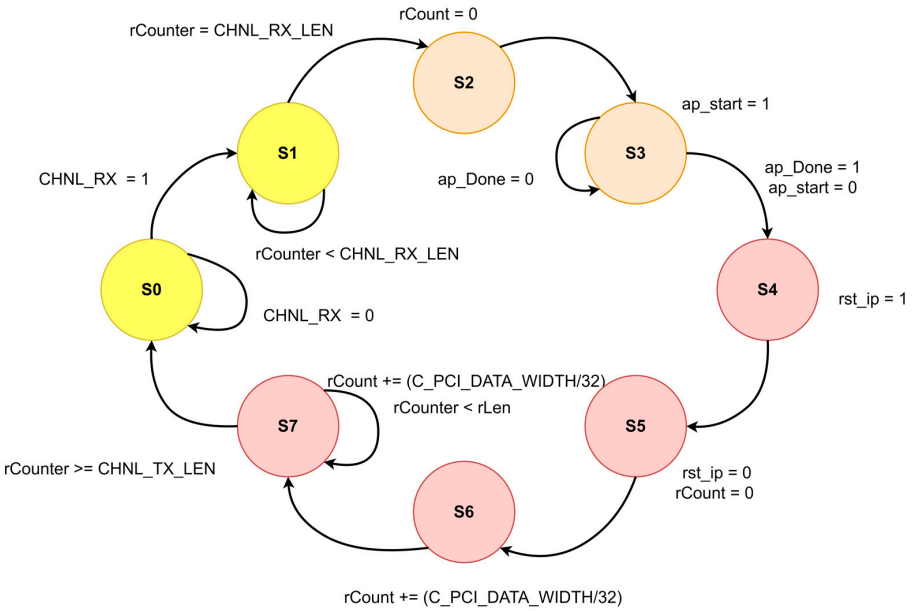

Figure 3. State machine for the operation of RIFFA channels: ■, PCIe port receives data; ■, IP processes the data; ■, PCIe port sends the processed data.

The receiving of data starts at state 0, waiting for the signal "CHNL_RX". Subsequently, state 1 collects the data and counts the number of received words, taking into consideration the signal "CHNL_RX_DATA_VALID". If the count is equal to "CHNL_RX_LEN", it continues to the next stage (processing). The data processing begins with a counter reset "rCount = 0", which occurs in state 2. Following that, state 3 asserts "ap_start" to indicate the start of the IP and waits for the assertion "ap_done", which indicates that the IP finished its work (e.g., data reordering or matrix transpose). Data sending starts at state 4 with the reset of the IP via assertion "rst_ip". State 5 sets "rst_ip" to zero and initializes "rCount". Next, state 6 loads "rCount" with the number of data integers, making use of signal "C_PCI_DATA_WIDTH". Recall that an integer is a data type with a width of 32 bits. Finally, at state 7, RIFFA is instructed to send the data from the FPGA to the CPU through the signal "CHNL_TX_LEN". After this, the finite state machine goes to state 0 and waits for the next data to be sent.

To simplify the implementation process, both the RIFFA channels and IPs are instantiated within RIFFA. The IP integrates with RIFFA through the RX/TX control signals specified in [17]. The IP is then connected to the RIFFA channel using the "ap_start" and "ap_done" signals. Once the integration is performed, the data transfer process begins with the fpga_send function, which sends data from the PC memory to the FPGA. The channel waits until all data is received and the "ap_start" signal is asserted, indicating that the IP can start processing the data. Once the IP finishes processing, it asserts the "ap_done" signal, and the channel waits for the fpga_recv function to retrieve the processed data from the FPGA.

Figure 4 provides a visual representation of the integration of HLS IPs with RIFFA by means of channels. Observe how the various components work together to enable efficient data transfer and processing.

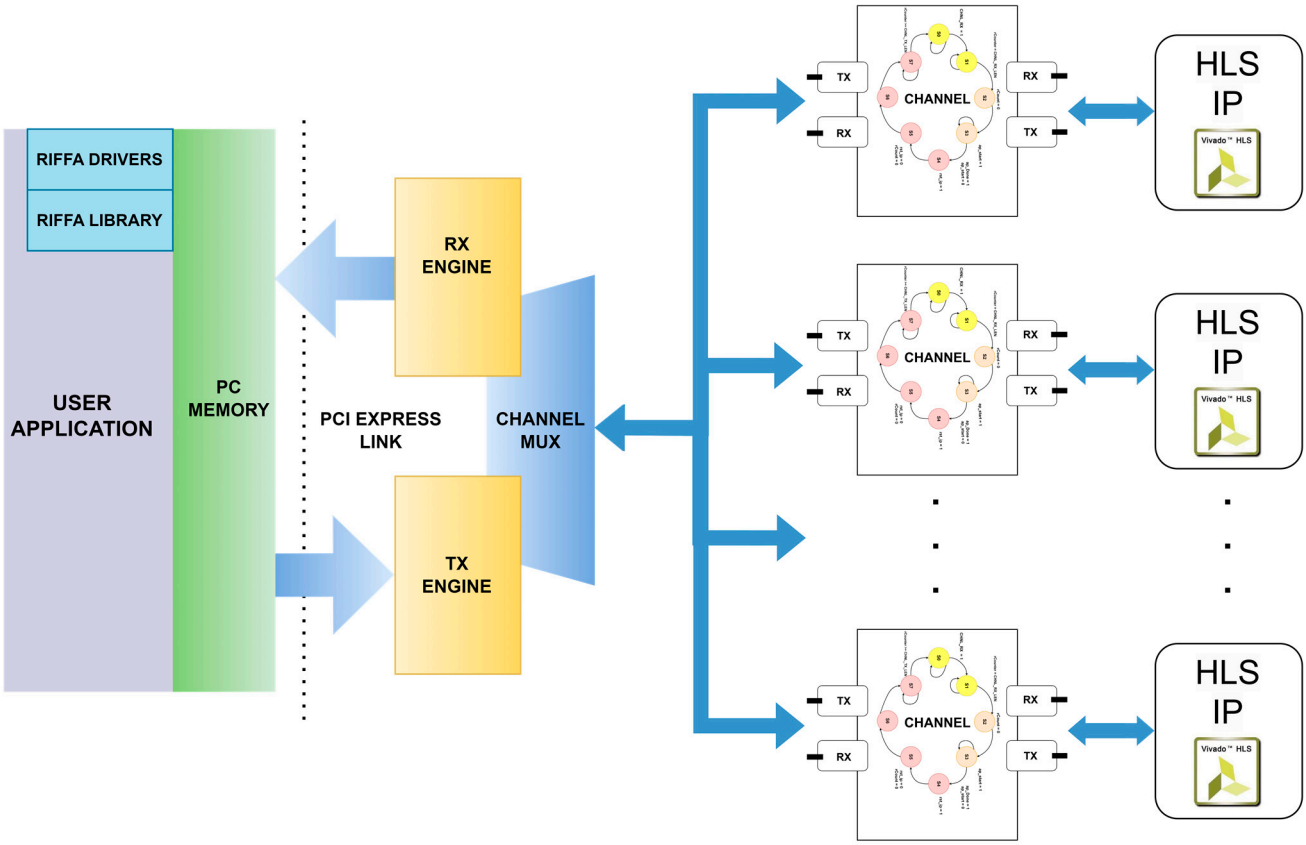

Figure 4. Schematic of the system. Up to 12 channels can be used per PCIe port.

4. Proof of Concept

Towards the usage of FPGA accelerators for on-board SAR processing, this section addresses the proof of concept of the proposed methodology via the implementation of two HLS IPs: the reordering stage of VEC-FFT and matrix transpose. The HW implementations are described in detail, providing insights for further implementations of SAR algorithms on FPGAs. First, VEC-FFT is briefly reviewed, explaining the necessity of a HW accelerator for the reordering stage. Subsequently, the implementation of such IP is presented, already integrated with RIFFA. Next, aimed at demonstrating flexibility in the utilization of different IPs, the IP for matrix transposition is addressed.

4.1. Vectorized Fast Fourier Transform (VEC-FFT)

VEC-FFT is an optimized FFT function based on the radix-4 algorithm [24]. The optimizations mainly concern an efficient implementation of Intel's SIMD instructions, leading to an ideal vectorization level and system usage rate. The main optimizations performed are listed next [18]:

- Separation of the input data into its real and imaginary parts.
- Splitting of the Radix-4 algorithm into its different stages with the goal of optimizing each stage separately.
- Restructuring of the Radix-4 butterfly [24] to reduce computational effort.
- Pre-calculation and pre-vectorization of the twiddle factors.
- Pre-identification of elements that must be swapped in the reordering final step.

The computations performed by FFT Radix-4 [18] are exemplified by:

$$\begin{bmatrix} X^F(k) \\ X^F\left(k+\frac{N}{4}\right) \\ X^F\left(k+\frac{N}{2}\right) \\ X^F\left(k+\frac{3N}{4}\right) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} A^F(k) \\ W_N^k B^F(k) \\ W_N^{2k} C^F(k) \\ W_N^{3k} D^F(k) \end{bmatrix} \tag{1}$$

$$k = 0,\ 1,\ \ldots,\ N-1,\ \forall N = 4^n,\ n \in \mathbb{N}.$$

with

$$A^F(k) = \sum_{n=0}^{\frac{N}{4}-1} x(4n) W_N^{4nk},$$

$$W_N^k B^F(k) = W_N^k \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_N^{4nk},$$

$$W_N^{2k} C^F(k) = W_N^{2k} \textstyle\sum_{n=0}^{\frac{N}{4}-1} x(4n+2) W_N^{4nk},$$

$$W_N^{3k} D^F(k) = W_N^{3k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) W_N^{4nk}$$

where $W_N^k = e^{-j\frac{2\pi k}{N}}$.

The restructuration of the kernel, as well as its separation into a real and imaginary part, are represented by Equations (2) and (3):

$$\begin{bmatrix} X^F_{re}(k) \\ X^F_{re}\left(k+\frac{N}{4}\right) \\ X^F_{re}\left(k+\frac{N}{2}\right) \\ X^F_{re}\left(k+\frac{3N}{4}\right) \end{bmatrix} = \begin{bmatrix} Sum^F_{re}(AC) + Sum^F_{re}(BD) \\ Sub^F_{re}(AC) + Sub^F_{re}(BD) \\ Sum^F_{re}(AC) - Sum^F_{re}(BD) \\ Sub^F_{re}(AC) + Sub^F_{re}(BD) \end{bmatrix}, \tag{2}$$

where

$$Sum^F_{re}(AC) = A^F_{re}(k) + W^{2k}_N C^F_{re}(k),$$

$$Sum^F_{re}(BD) = W^k_N B^F_{re}(k) + W^{3k}_N D^F_{re}(k),$$

$$Sub^F_{re}(AC) = A^F_{re}(k) - W^{2k}_N C^F_{re}(k),$$

$$Sub^F_{re}(BD) = W^k_N B^F_{im}(k) - W^{3k}_N D^F_{im}(k);$$

and

$$\begin{bmatrix} X^F_{im}(k) \\ X^F_{im}\left(k+\frac{N}{4}\right) \\ X^F_{im}\left(k+\frac{N}{2}\right) \\ X^F_{im}\left(k+\frac{3N}{4}\right) \end{bmatrix} = \begin{bmatrix} Sum^F_{im}(AC) + Sum^F_{im}(BD) \\ Sub^F_{im}(AC) - Sub^F_{im}(BD) \\ Sum^F_{im}(AC) - Sum^F_{im}(BD) \\ Sub^F_{im}(AC) + Sub^F_{im}(BD) \end{bmatrix}, \tag{3}$$

where

$$Sum^F_{im}(AC) = A^F_{im}(k) + W^{2k}_N C^F_{im}(k),$$

$$Sum^F_{re}(BD) = W^k_N B^F_{im}(k) + W^{3k}_N D^F_{im}(k),$$

$$Sub^F_{re}(AC) = A^F_{im}(k) - W^{2k}_N C^F_{im}(k),$$

$$Sub^F_{re}(BD) = W^k_N B^F_{re}(k) - W^{3k}_N D^F_{re}(k).$$

Equations (1)–(3) are depicted in Figure 5. Figure 5a shows a Radix-4 butterfly (Equation (1)) of 16 points, with entries given in normal order and outputs retrieved in reversed bit order. Figure 5b represents the restructuration of the kernel (Equations (2) and (3)), aimed at reducing computational effort by reutilizing previously calculated values. Figure 1c exemplifies Equation (2), in which the formerly computed values $Sum^F_{re}(AC)$, $Sum^F_{re}(BD)$, $Sub^F_{re}(AC)$, and $Sub^F_{re}(BD)$, are reused.

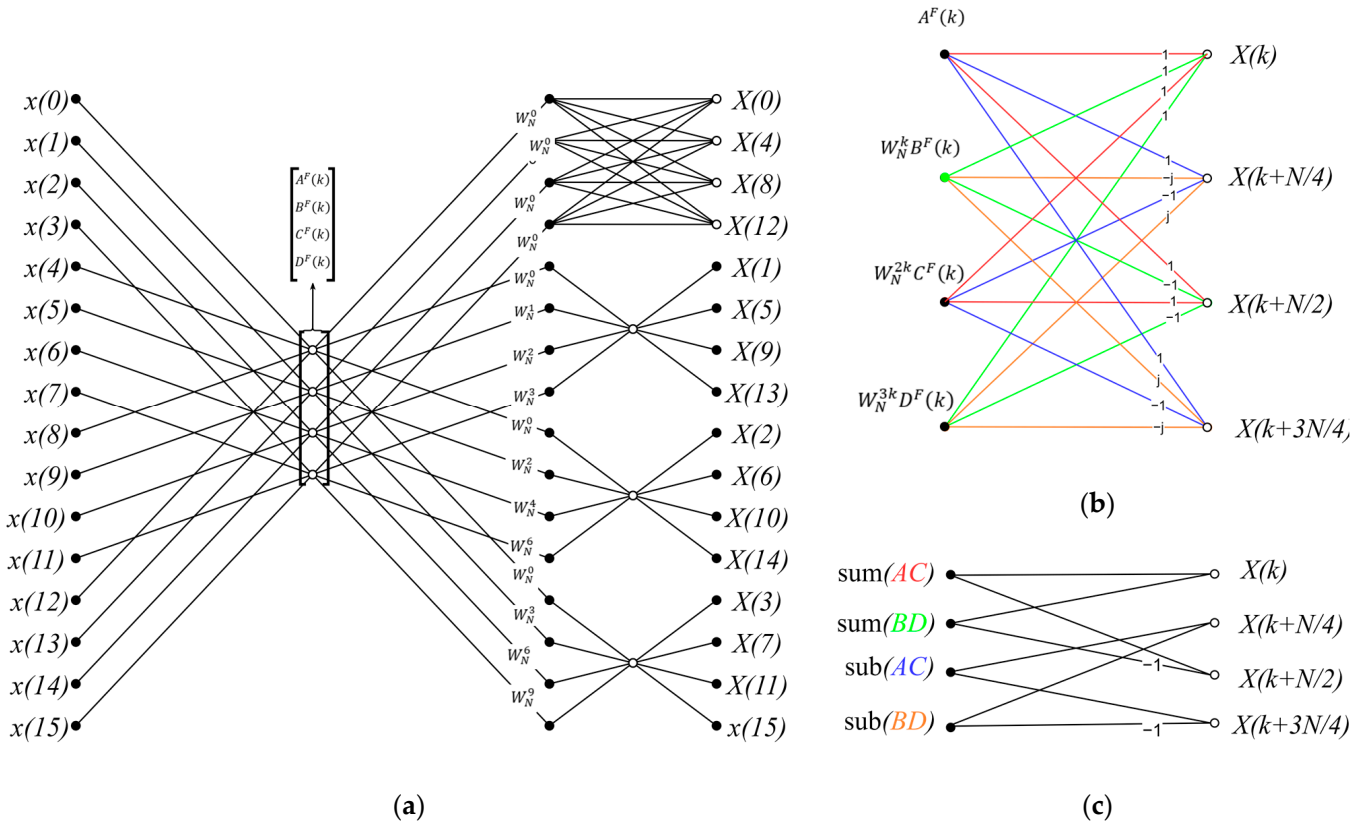

Figure 5. (**a**) Radix-4 butterfly of 16 points, with entries given in normal order and outputs retrieved in reversed bit order. (**b**) Restructuration of the kernel (Equations (2) and (3)). (**c**) Graphic representation of Equation (2), handling the real data in the FFT.

Tables 2 and 3 show the computation performance of VEC-FFT in CPU clock cycles, with and without the reordering stage [18]. The tests consider FFT sizes ranging from 64 to 16,384 points. When comparing the performance of a 16,384-point FFT, it turns out that the reordering stage has a significant impact on the overall performance, consuming 60% of the total processing time. For instance, for 16,384 points, VEC-FFT requires 234,061 clock cycles; conversely, it consumes only 91,601 clock cycles when data reordering is excluded. One of the main reasons for this difference in clock cycles is the amount of memory changed when computing the reordering stage: 8064 for both real and imaginary parts.

Table 2. VEC-FFT performance in clock cycles [18].

Size	Min	Median	Mean	Standard Deviation
64	253	262	265	10
256	1207	1238	1248	87
1024	5428	5508	5625	308
4096	24,974	25,442	25,722	891
16,384	227,400	229,619	234,061	11,282

Table 3. VEC-FFT performance in clock cycles without reordering [18].

Size	Min	Median	Mean	Standard Deviation
64	187	199	199	9
256	766	802	804	31
1024	3649	3712	3746	162
4096	17,079	17,343	17,464	437
16,384	88,576	90,200	91,601	6222

Additionally, as mentioned in [18], the increase in repetitive calls of FFT functions (e.g., VEC-FFT) reduces performance due to the HW administration performed by the Kernel in the Operating System (OS). Recall that the Kernel is the main layer between the OS and the HW, responsible for memory and process management, file systems, device control, and networking. The repetitive calls of FFT functions (e.g., VEC-FFT) are common in SAR processing. Such is the case of the first and last stages of Omega-K in [23], which require performing 2D FFT on the imagery, involving the computation of FFT series.

4.2. HLS IP for the Reordering Stage of VEC-FFT

HLS offers the possibility of using the optimizations mentioned in Section 2 by means of a single line of code, as seen in Scheme 2. However, since the C/C++ synthesis is performed by HLS, it is difficult to know a priori which pragmas perform more efficiently for certain tasks. Consequently, it is common to compare the different optimizations after synthesis. Five different optimizations are applied: cyclic partition, pipeline with one Block Random Access Memory (BRAM) port, pipeline with two BRAM ports, unroll, and unroll and pipeline (1 BRAM port). The obtained results are presented in Table 4, contrasting those retrieved without any optimization.

```
#pragma HLS PIPELINE
#pragma HLS UNROLL factor=4
#pragma HLS array partition variable=AB cyclic factor=4
```

Scheme 2. Usage of pragmas in HLS. Each sentence represents a different optimization: pipeline, unroll, and cyclic partition, respectively.

Table 4. Optimizations were applied to VEC-FFT's reordering function (1024 points). Three main parameters are assessed: maximum frequency at which the circuit works, latency (amount of clock cycles required by the IP to finish the task), and resources used, i.e., number of slices, Look-Up Tables (LUTs), Flip Flops (FFs), and DSPs.

Parameter		HLS Implementations					
		Cyclic Partition	Pipeline with 2 BRAM Ports	Pipeline with 1 BRAM Port	Unroll and Pipeline (1 BRAM Port)	Unroll	Without Optimization
Maximum Frequency		118.119 MHz	121.227 MHz	121.227 MHz	125.723 MHz	130.548 MHz	130.548 MHz
Latency (Clock cycles)		2713	2738	4500	9411	23,714	23,906
Resources used	Slices	778	582	465	1007	751	751
	LUTs	2180	1607	1275	2341	1942	1942
	FFs	2096	1692	1197	2632	2208	1861
	DSPs	12	8	4	30	20	20

The VEC-FFT reordering IP is optimized with a cyclic partition due to the attained lower latency, as observed in Table 4. Eight Block Random Access Memories (BRAMS) are needed: 4 BRAMs for real data and 4 BRAMs for imaginary data. The connections between the IP, the memories, and the input/output signals are shown in Figure 6. The integration of RIFFA with the IP is achieved through the schematic depicted in Figure 4, as explained previously. Data is sent from the CPU to the memory blocks via PCIe, splitting the 128-bit frame into four frames of 32 bits, one per memory (imaginary or real data). The data reordering IP starts working until the acknowledgement of data reception "ap_start" is received. Subsequently, IP processes the data and saves the results in the memories. After the processing is finished, "ap_done" is asserted, and the data is sent to the CPU via PCIe.

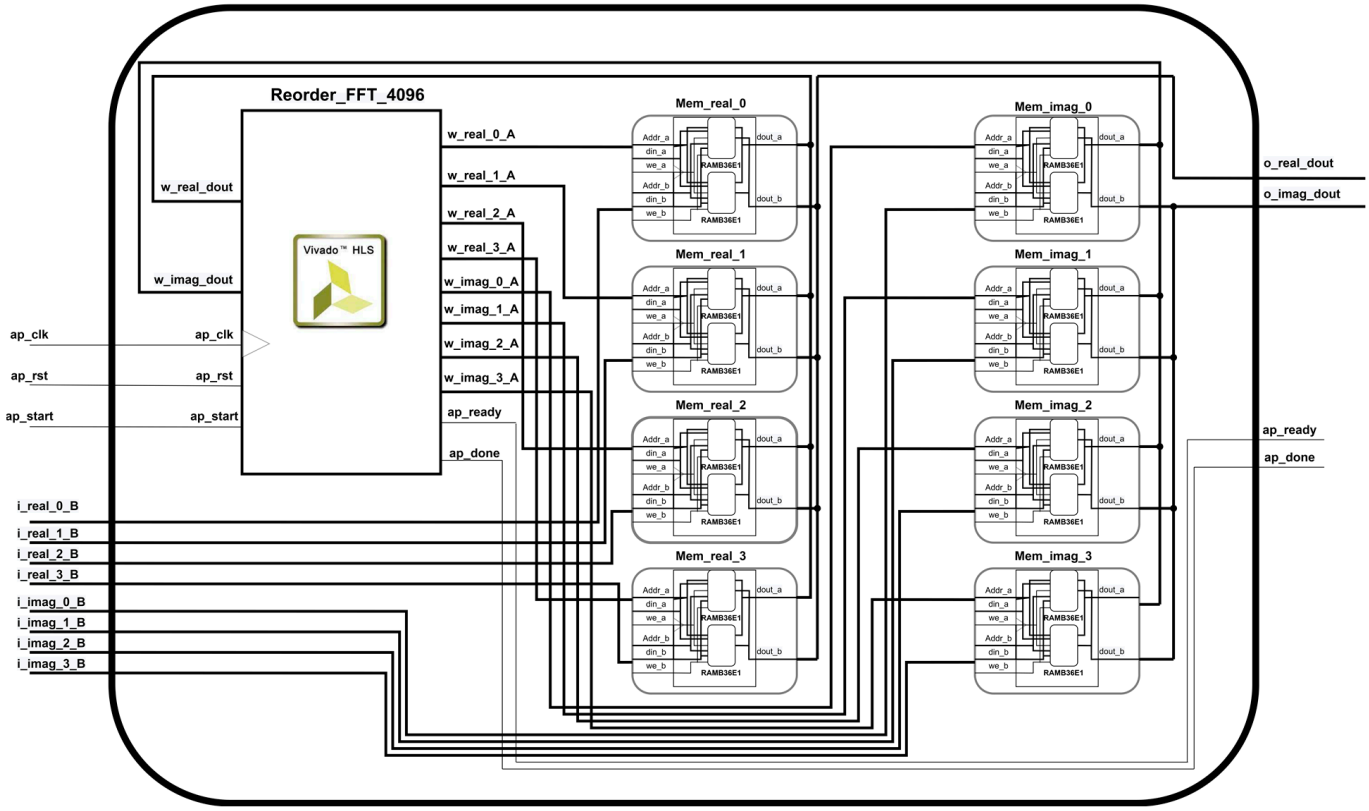

Figure 6. Schematic of the data reordering module (4096 points). The module employs 8 BRAMs to store the data.

The results retrieved by the IP are compared against a gold bench variable as part of the validations performed within the HLS workflow in Figure 1. A tolerance value of 1×10^{-5} is set, meaning a difference between both attained results of ± 0.00001, which leads to a mean square error (MSE) in the order of 1×10^{-10}. With respect to FFTW and FFTW SIMD [18], VEC-FFT presents an MSE on attained results in the order of 1×10^{-9}.

4.3. HLS IP for Matrix Transpose

The matrix transpose transforms a given matrix by interchanging its rows and columns so that the rows become columns and the columns become rows. Matrix transpose is regularly used in SAR imaging to compute the 2D FFT [19–23]. First, the 1D FFT is performed along each row of a matrix; then, the matrix is transposed; afterwards, the 1D FFT is applied along each row; finally, the matrix is transposed one more time to put the resultant array in the correct order. The sequence in which the 1D FFTs are performed is important, as it affects the orientation of the resulting frequency spectrum.

The HLS IP for matrix transposition is generated from the C++ code presented in Scheme 3. Five different optimizations are applied, as described in Table 5.

```
#include "Transpose.h"
#define N 16384
void Transpose(volatile float Input_Data[N],volatile float
Output_Data[N],float x, float y){
     int i,j;
     int X,Y;
     X = (int)x;
     Y = (int)y;
     for(i=0; i < X; i++){
          for(j=0; j < Y; j++){
               Output_Data[i+j*X] = Input_Data[j+i*Y];}
     }}
```

Scheme 3. C++ code for matrix transposition.

Table 5. Optimizations applied for matrix transposition (128 × 128 points). Three main parameters are assessed: the maximum frequency at which the circuit works, latency, and resources used, i.e., number of slices, LUTs, FFs, and DSPs.

Parameter		HLS Implementations					
		Cyclic Partition	**Pipeline with 2 BRAM Ports**	**Pipeline with 1 BRAM Port**	**Unroll and Pipeline (1 BRAM Port)**	**Unroll**	**Without Optimization**
Maximum Frequency		131.65 MHz	131.69 MHz	131.69 MHz	131.69 MHz	131.69 MHz	131.65 MHz
Latency (Clock cycles)		16,401	16,803	17,035	16,675	20,771	33,025
Resources used	Slices	482	482	465	704	1493	399
	LUTs	1301	1031	1301	1901	3534	1078
	FFs	547	547	547	785	1215	351
	DSPs	12	8	12	12	40	3

Due to lower latency, cyclic partition is chosen for the final implementation of the HLS IP, which makes use of four BRAMS. The code utilized for such HLS optimization is presented in Scheme 4. The matrix transpose module has a similar structure as the one presented in Figure 6. As seen in Figure 4, the IPs are connected to a specific RIFFA channel to send and receive data through the PCIe port.

```
void Transpose(volatile float Input_Data[N],volatile float
Output_Data[N],float x, float y){
#pragma HLS array_partition variable=Input_Data cyclic factor=4
#pragma HLS array_partition variable=Output_Data cyclic factor=4
     for(i=0; i < X; i++){
          for(j=0; j < Y; j++){
               Output_Data[i+j*X] = Input_Data[j+i*Y];}
          }}
```

Scheme 4. HLS cyclic partition optimization for matrix transposition.

5. Performance Assessments

5.1. VEC-FFT Data Reordering

Table 6 presents the clock cycles required by both implementations, CPU and FPGA. The results attained in [18] are included to show the considerable impact that CPU technology has on performance. Appendix A summarizes the system resources employed in this research, whereas Appendix B depicts the system resources utilized in [18]. For completeness, Table 7 compares the clock cycles employed by VEC-FFT (with and without HW accelerator) against FFTW and FFTW SIMD [18] for a 16,384-point FFT.

Table 6. Data reordering measured in clock cycles: CPU vs. FPGA.

VEC-FFT Data Reordering			
CPU vs. FPGA in Clock Cycles			
CPU	**CPU as in [18]**	**FPGA**	**Size**
4452	1879	2713	1024
19,592	8379	11,161	4096
178,068	142,460	44,569	16,384

Table 7. FFT of 16,384 points measured in clock cycles: VEC-FFT vs. FFTW.

FFT of 16,384 Points			
VEC-FFT vs. FFTW in Clock Cycles			
VEC-FFT+HW IP	**VEC-FFT [18]**	**FFTW [18]**	**FFTW SIMD [18]**
136,170	234,061	452,253	106,601

Table 8 addresses the RX/TX DTR, measured in bytes per second (B/s). SW DTR measures the time to fetch, process, and store data between memory and CPU, whereas HW DTR measures the time to send and receive data between memory and FPGA. The FPGA makes use of PCIe 2.0 with four lanes, attaining a maximum theoretical DTR of 2 GB/s. Eighty thousand repetitions are measured, divided into sections of ten thousand, to avoid possible OS noise. Since the CPU sends and receives data while working with it, the time of the entire function is measured. The measurements highlighted in green represent the best-achieved performance.

Table 8. RX/TX data throughput: SW vs. HW of VEC-FFT Data Reordering with 16,384 Points, the green background shows the best performance.

VEC-FFT Data Reordering of 16,384 Points			
	SW	**HW**	
		TX	**RX**
8 cycles of 10 K repetitions	61.83 µs	80.60 µs	80.09 µs
	62.63 µs	80.62 µs	80.14 µs
	61.72 µs	80.37 µs	80.10 µs
	61.96 µs	80.53 µs	81.06 µs
	62.02 µs	80.63 µs	80.15 µs
	61.43 µs	80.65 µs	80.25 µs
	62.11 µs	80.53 µs	80.13 µs
	63.93 µs	80.54 µs	80.12 µs
Throughput	4.3379 GB/s	1.6308 GB/s	1.6364 GB/s

At this point, only one IP is evaluated at a time. However, it is possible to run multiple IPs at the same time. Due to the limitations of the FPGA and motherboard, a maximum of three IPs is set. Figure 7 shows the behavior of the system with three IPs.

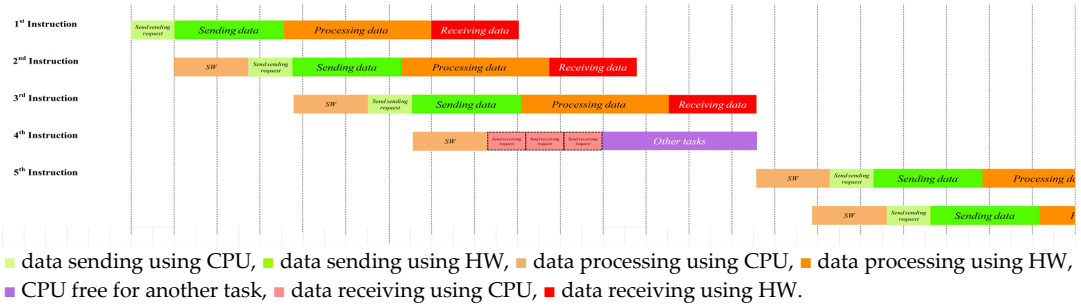

Figure 7. Pipeline optimization for VEC-FFT data reordering (16,384 points) using three IPs. For easy assessment, different colors are employed.

The FPGA works at 100 MHz; therefore, it requires more time to execute the algorithm than the CPU, which works at 3.6 GHz. The time difference is used to perform three additional VEC-FFT reorderings in the CPU; this means that the CPU and the FPGA work in parallel. The results in clock cycles of such evaluations are presented in Table 9. These were measured in SW, with a frequency of 3.6 GHz.

Table 9. RX/TX data throughput in clock cycles: SW vs. HW with three VEC-FFT Data Reordering IPs, the green background shows the best performance.

Six VEC-FFT Data Reordering of 16,384 Points		
	SW	**HWwith 3 IPs (3 VEC-FFT Reorderingin CPU)**
8 cycles of 100 K repetitions	1,169,660	1,159,188
	1,163,704	1,143,108
	1,149,216	1,153,364
	1,165,484	1,156,580
	1,159,100	1,148,352
	1,149,184	1,154,900
	1,169,496	1,160,336
	1,159,256	1,150,128
Throughput	4.105 GB/s	4.127 GB/s

5.2. Matrix Transpose

As mentioned in the introduction, the proposed HW architecture offers the flexibility of using different IPs. Therefore, for demonstration purposes, a matrix transpose IP is developed, another computationally expensive process due to memory latency. The matrix transpose is implemented for squared matrices with dimensions of 32 × 32, 64 × 64, and 128 × 128, respectively. Table 10 presents the required clock cycles for the different implementations of the matrix transpose, while Table 11 shows the DTR for a 128 × 128 matrix transpose. Finally, Table 12 presents the performance in clock cycles when working with three IPs and performing the transpose of three 128 × 128 matrices. Measurements in Table 12 are performed at 3.6 GHz.

Table 10. Matrix transpose measured in clock cycles: CPU vs. FPGA.

Matrix Transpose		
CPU vs. FPGA in Clock Cycles		
CPU	**FPGA**	**Size (N × M)**
10,424	1041	**32× 32**
38,505	4113	**64× 64**
210,928	16,401	**128× 128**

Table 11. RX/TX data throughput: SW vs. HW of 128 × 128 Matrix Transpose, the green background shows the best performance.

128 × 128 Matrix Transpose			
	SW	**HW**	
		TX	**RX**
8 cycles of 10 K repetitions	90.67 µs	40.17 µs	40.07 µs
	90.50 µs	40.20 µs	40.06 µs
	90.53 µs	40.20 µs	40.08 µs
	90.52 µs	40.19 µs	40.06 µs
	90.71 µs	40.25 µs	40.10 µs
	92.11 µs	40.38 µs	40.03 µs
	90.67 µs	40.21 µs	40.02 µs
	90.66 µs	40.27 µs	40.07 µs
Throughput	1.4483 GB/s	1.6314 GB/s	1.6375 GB/s

Table 12. RX/TX data throughput in clock cycles: SW vs. HW with three 128 × 128 Matrix Transposes IPs, the green background shows the best performance.

Three 128 × 128 Matrix Transposes		
	SW	**HWwith 3 IPs**
8 cycles of 10 K repetitions	646,959	279,195
	633,711	281,649
	629,814	281,325
	619,368	281,310
	631,227	279,813
	660,726	279,978
	629,667	281,460
	627,411	280,152
Throughput	2.285 GB/s	2.5 GB/s

5.3. Power Consumption

Reports on power consumption can be generated with the Vivado design suite [26]. As an example, for the development board xc7z045ffg900-2, we present in Figure 8 the power consumption of a single IP for the reordering stage of VEC-FFT (1024 points). The environmental parameters include an Output Load of 0 pF, an ambient temperature of 40 °C, and an Airflow of 500 linear feet per minute. The report yielded a total consumption of 2.174 W, of which 1.716 W are related to RIFFA, 0.136 W to the IP accelerator, and 0.322 W to other factors.

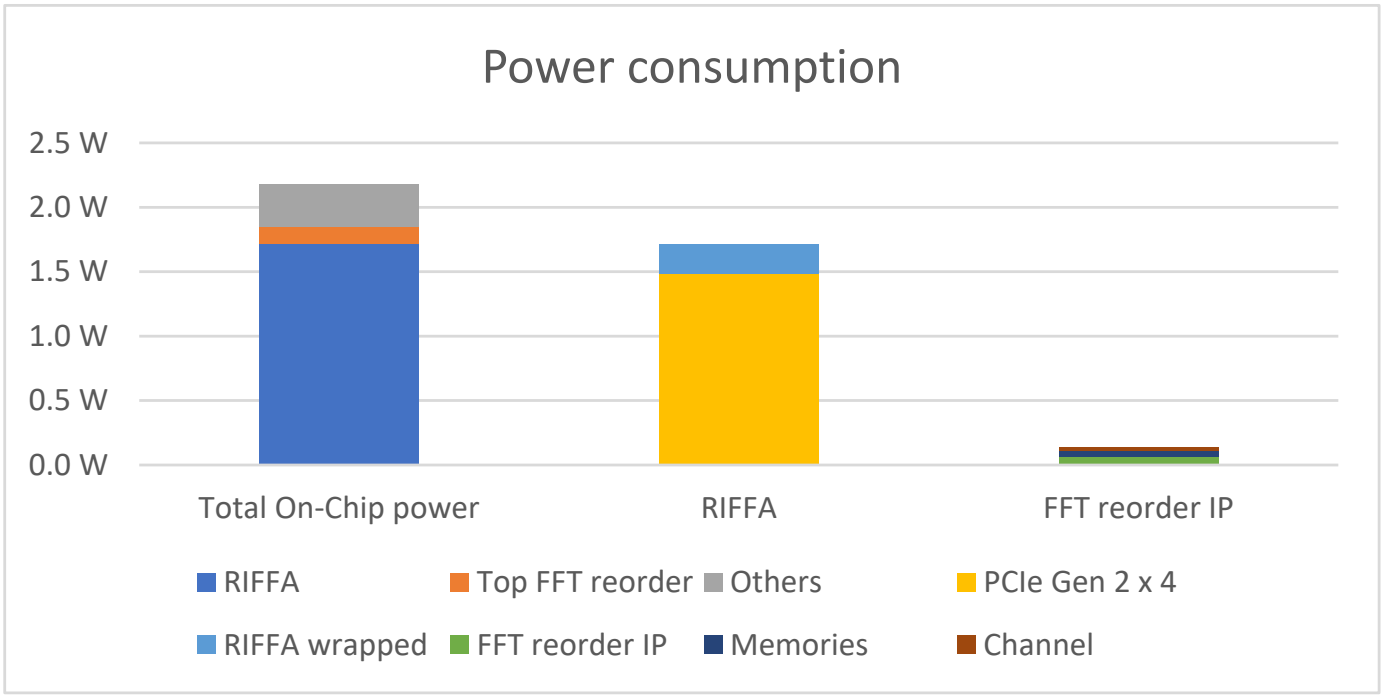

Figure 8. Power consumption report: single IP for the reordering stage of VEC-FFT.

Figure 9 depicts the total power consumption associated with multiple IPs. The graph exhibits a linear relationship, indicating that the power consumption increases by approximately 0.132 W for each FFT reordering. Power consumption encompasses several components, including the VEC-FFT IP, memories, and channels. Additionally, the RIFFA framework contributes to the overall power consumption, with a linear increase of approximately 0.123 W per channel (because RIFFA's architecture is modified to use more channels). This information highlights the direct impact of the number of FFT reorderings and channels on power consumption in the system. As the number of FFT reorderings or channels increases, the power consumption rises accordingly. It is important to consider such a trend in power consumption when designing and implementing on-board SAR systems, as it directly affects the system's overall power requirements and efficiency.

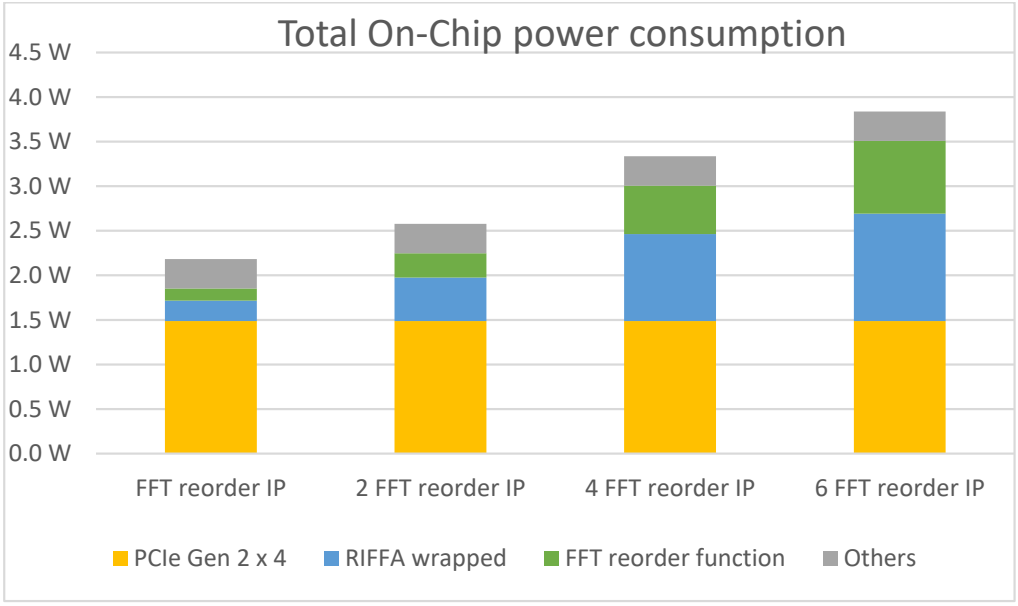

Figure 9. Total on-chip power consumption: multiple IPs for the reordering stage of VEC-FFT.

6. Discussion

Tables 6 and 10 present the processing clock cycles required for each implementation of VEC-FFT data reordering and matrix transpose, respectively. As can be observed, HW implementations perform better than SW implementations when a large amount of memory access is required. In Table 10, for example, the FPGA presents a significant improvement in clock cycles in contrast to the CPU since the matrix transpose consists of a series of value changes in memory. The FPGA performs multiple memory accesses when storing and loading multiple data sets at the same time.

Concerning Table 7, at first glance, it might seem that FFTW SIMD has better performance than VEC-FFT+HW IP; however, recall that the FPGA is capable of executing multiple IPs at the same time. This means that, using the same clock cycles, VEC-FFT+HW IP can perform as many FFT reorderings as there are IPs.

Tables 8 and 11 present the attained DTR for VEC-FFT data reordering (16,384 points) and 128 $\times$ 128 matrix transpose, correspondingly, using the CPU clock as reference. On the other hand, Tables 9 and 12 consider the same cases, respectively, but with multiple (three) IPs. As can be seen, the data throughput increases when multiple IPs are used and higher parallel processing is performed. For the particular case of VEC-FFT data reordering, the achieved data throughput is above the DTR defined (limited) by the PCIe port since both the CPU and FPGA perform the VEC-FFT data reordering, as depicted in Figure 7.

Figure 8 shows the power consumption of a single IP for the reordering stage of VEC-FFT (1024 points), using the FPGA xc7z045ffg900-2. As can be observed, the PCIe ports consume most of the power. An increase in power consumption is expected with more IPs, but it is still lower in comparison to the PCIe ports.

Power consumption depends on the architecture and the implemented system. The addressed implementation consumes 2.174 W with one VEC-FFT IP and two channels, 2.577 W with two VEC-FFT IP and four channels, 3.337 W with four VEC-FFT IP and eight channels, and 3.838 W with six VEC-FFT IP and twelve channels. It is important to note that the implemented system can be further optimized to achieve more efficient power consumption. For example, in the case of VEC-FFT data reordering, modifying the system to use a single channel can result in a power saving of approximately 0.123 W. Additionally, when power consumption is a primary constraint, optimizations that prioritize lower power consumption can be chosen instead of lower latency.

On-board SAR systems have specific requirements that depend on the application and platform. These requirements include compact size and weight to accommodate the limited

space and weight constraints of airborne and spaceborne platforms like drones or satellites. Power efficiency is crucial to ensuring longer mission endurance and minimal energy consumption, considering the limited power sources available. Real-time processing capability is essential for rapid data acquisition, processing, and image reconstruction to provide timely and actionable information. High-resolution imaging is a requirement to capture detailed images of the Earth's surface, necessitating the use of sensors and algorithms that can achieve the desired resolution. On-board systems should also consider data storage and transmission capabilities, including sufficient storage capacity, data compression techniques, and efficient data transfer methods. Furthermore, on-board SAR systems must be designed to withstand harsh environmental conditions encountered during aerial or spaceborne missions, ensuring resilience to temperature variations, vibration, electromagnetic interference, and external elements.

Overall, on-board SAR systems must be compact, power-efficient, capable of real-time processing, and able to capture high-resolution images. They need to integrate sensors effectively, facilitate data storage and transmission, and exhibit resilience in harsh environments. Adapting the system design to meet the specific requirements of each application and platform is vital for successful deployment and data acquisition.

The combination of FPGA and CPU implementations using a PCIe interface offers several advantages for SAR systems. It leverages the parallel processing power of FPGAs alongside the flexibility and customizability of CPUs, resulting in the accelerated execution of computationally intensive SAR algorithms. The PCIe interface facilitates efficient data transfer between the FPGA and CPU, enabling seamless communication and real-time processing. This implementation optimizes resource utilization, reduces power consumption, and provides scalability options for handling varying processing requirements.

This study utilizes PCIe 2.0 with four lanes, reaching a maximum DTR of 2 GB/s, limited by the PCIe technology. In order to achieve the same or better performance than the CPU, a different PCIe technology might be employed. Correspondingly, Figure 10 shows the DTR for different PCIe technologies and the number of lanes. For example, PCIe 1.0 with four lanes attains a maximum DTR of 1 GB/s, whereas more recent versions of PCIe with four lanes attain a maximum DTR of 2 GB/s, 3.9 GB/s, 7.8 GB/s, and 15.7 GB/s for PCIe 2.0, 3.0, 4.0, and 5.0, respectively. Note that increasing the number of lanes increases the DTR achieved.

PCI Express link performance

Version	Introduced	Line code	Transfer rate	Throughput				
				x1	x2	x4	x8	x16
1.0	2003	8b/10b	2.5 GT/s	0.250 GB/S	0.500 GB/S	1.000 GB/S	2.000 GB/S	4.000 GB/S
2.0	2007	8b/10b	5.0 GT/s	0.500 GB/S	1.000 GB/S	2.000 GB/S	4.000 GB/S	8.000 GB/S
3.0	2010	128b/130b	8.0 GT/s	0.985 GB/S	1.969 GB/S	3.938 GB/S	7.877 GB/S	15.754 GB/S
4.0	2017	128b/130b	16.0 GT/s	1.969 GB/S	3.938 GB/S	7.877 GB/S	15.754 GB/S	31.508 GB/S
5.0	2019	128b/130b	32.0 GT/s	3.938 GB/S	7.877 GB/S	15.754 GB/S	31.508 GB/S	63.015 GB/S

Lower performance
Same performance
Higher performance
ZC706 Evaluation Board

PCIe DTR for 128 × 128 matrix.

Figure 10. Comparative table of PCIe DTR for different versions and numbers of lanes. Each color depicts different performance in comparison to CPU processing: red = lower performance; yellow = equal performance; and green = higher performance.

A suitable PCIe technology and the addition of more FPGAs increase parallel processing capacity, overcoming CPU measurements easily. For instance, PCIe 2.0 with 16 lanes attains a theoretical DTR of 8 GB/s, or about 7.4 GB/s in practice. For VEC-FFT data reordering of 16,384 points, such technology reduces the TX/RX time from 61.43 µs (see Table 8) to 35.43 µs, i.e., approximately 26 µs less than the CPU. This technology is available on multiple boards, e.g., the Xilinx Virtex-7 FPGA VC709, the Solar Express 125, and the XUP-PL4. Moreover, RIFFA [17] supports using a maximum of five FPGAs per system with PCIe 3.0 or previous versions; also, since RIFFA works with a direct memory access engine, it could be upgraded to PCIe 4.0.

In the context of wider expectations and requirements in on-board SAR processing, the proposed methodology aligns with the need for real-time and efficient data processing. The high DTR achieved, up to 15.7 GB/s, through the PCIe interface developed using RIFFA addresses the challenge of data downlink by minimizing transfer bottlenecks and enabling faster communication between the CPU and FPGA. This capability is vital for on-board SAR systems, where timely and efficient data transfer is crucial for real-time decision-making and analysis. As mentioned, the data size with three channels for MMTI with 0.5 m $\times$ 0.5 m resolution is about 210 GB; correspondingly, with a DTR of 15.7 GB/s, the transfer bottlenecks are significantly reduced.

Electromagnetic interference avoidance is a relevant factor in SAR space missions. Accordingly, the proposed methodology allows the employing of space-grade FPGAs like the Xilinx Virtex-5QV FX130 and Microsemi RTG4, which have an average power consumption of 9.97 W and 3.91 W, respectively [9].

The choice of optimizations and the balance between power consumption and processing capacity will ultimately depend on the specific requirements and constraints of the project. As highlighted previously, there is a direct relationship between power consumption and processing capacity, necessitating careful consideration and trade-offs to achieve the desired system performance while optimizing power efficiency.

Related to a GPU implementation, leaving aside the limitations of GPUs for space missions, GPU accelerators offer significant advantages for SAR processing. Their parallel processing capabilities allow for simultaneous computation on multiple data elements, making them well-suited for SAR algorithms. GPUs provide high computational power, enabling faster processing and analysis of SAR data compared to CPU-based approaches. With careful optimization and algorithm design, GPUs can deliver substantial speedups. Additionally, GPU libraries, such as cuFFT and cuBLAS, provide optimized functions for efficient SAR data processing. However, GPUs present limitations. Memory constraints and bandwidth limitations can pose challenges when dealing with large datasets. Some SAR algorithms may not be easily parallelizable or may have irregular data access patterns, affecting GPU performance. GPU programming complexity and power consumption are additional considerations. Despite these limitations, GPUs remain a valuable tool for SAR processing, and with careful optimization and algorithm design, they can provide substantial speedups compared to CPU-based approaches.

Although multiple implementations with low-power GPUs (Jetson TX1, Jetson TX2, or Jetson Nano) of on-board SAR processing have been made, such as [11,12], the amount of data that these low-power GPUs can process is limited. On the other hand, the use of GPUs with higher processing capacity, such as the one mentioned in [7] (the Tesla K10 GPU Accelerator), has a power consumption of more than 150 W, which is a factor to consider in on-board processing.

The advantages of the addressed methodology include the utilization of HLS to create IP blocks and RIFFA to develop a PCIe interface. Employing HLS significantly reduces development time, typically between fivefold and tenfold [15,16]. This accelerates the design process and enables faster iterations and optimizations, which is one of the primary constraints in multiple on-board SAR implementations regarding FPGA implementations, as mentioned in [4,7–9].

Additionally, HLS offers several optimizations, including pipeline, cyclic partition, and unroll techniques, which can further enhance the performance of the FPGA-based SAR processing routines. These optimizations contribute to improved efficiency and resource utilization, allowing for better utilization of the FPGA's capabilities. E.g., the FFT could be implemented with the radix-8 algorithm, and instead of using eight memories as shown in Figure 6, sixteen memories would be used, and eight data sets would be processed at a time instead of four.

The RIFFA framework facilitates the use of up to five FPGAs, with the potential for multiple IPs to be implemented in each FPGA. This scalability ensures that the methodology

can handle more complex SAR processing tasks or accommodate larger data volumes, enhancing its applicability to different scenarios.

In comparison to existing approaches, the proposed methodology stands out by leveraging the combined advantages of HLS, IP blocks, RIFFA, and FPGA accelerators. While individual components and techniques have been employed in previous research, their integration and application specifically for on-board SAR processing is relatively novel. By capitalizing on these technologies, the methodology tackles the limitations and challenges faced by conventional approaches, such as time-consuming development cycles, limited scalability, and suboptimal processing efficiency.

Nevertheless, the use of FPGA accelerators presents some disadvantages that need to be considered. One significant disadvantage is the resource consumption of FPGA accelerators. FPGA designs require careful management of resources such as logic elements, memory, and interconnects. Implementing SAR processing algorithms on FPGAs can be resource-intensive, potentially requiring larger and more expensive FPGA devices to accommodate the computational requirements. This can increase the overall cost of the on-board SAR system. Additionally, FPGA accelerators may operate at lower frequencies compared to other processing technologies like CPUs or GPUs. This lower operating frequency can limit the processing speed and real-time capabilities of on-board SAR systems, affecting their ability to handle time-sensitive tasks efficiently.

The implementation code of this article can be found in a public GitHub repository: https://github.com/Baungarten-CINVESTAV/Towards-On-Board-SAR-Processing-with-FPGA-Accelerators-and-a-PCIe-Interface (accessed on 11 April 2023). The repository consists of three main folders:

i. HLS: contains the files related to the optimizations presented in Tables 4 and 5 and the VEC-FFT reordering implementation for different data sizes.
   a. FFT_Optimization_test: scripts and reports of the addressed HLS optimizations for VEC-FFT reordering.
   b. Transpose_Optimization_test: scripts and reports of the different HLS optimizations for matrix transpose.
   c. FFT_Reorder_x_num: scripts and reports of the VEC-FFT reordering function for different data sizes, using cyclic partition.
ii. Riffa_Scripts: provides the C code used to measure the number of clock cycles and to transfer the data from the CPU to the FPGA and vice versa.
iii. Verilog: contains the Verilog files and performance reports of the schematic presented in Figure 4.

A readme file in the GitHub repository explains the points above in detail.

7. Conclusions

Aimed at incorporating FPGA accelerators into on-board SAR processing algorithms, this article introduces a novel methodology to combine HW and SW via PCIe. Two main tools are employed, the HLS synthesizer (i.e., Vivado HLS) and RIFFA, reducing development time significantly (between fivefold and tenfold) and attaining high transfer speeds, up to 15.7 GB/s for PCIe 3.0 with 16 lanes. HLS is a fast and efficient way of creating IPs by transforming high-level programming languages (e.g., C/C++, System C, or OpenCL) into HDL (i.e., Verilog or VHDL). Furthermore, it also permits easily implementing optimizations like pipeline, unroll, and array partition. RIFFA, on the other hand, provides a framework that works directly with the PCIe endpoint; therefore, it is no longer necessary to implement PCIe communication.

The development of onboard SAR systems presents various challenges and considerations. These systems must meet specific requirements such as compact size, power efficiency, real-time processing capability, high-resolution imaging, and resilience to harsh environmental conditions. Integrating FPGA accelerators with a CPU using a PCIe interface offers several advantages for on-board SAR processing. The PCIe interface ensures efficient

data transfer between the FPGA and CPU (up to 15.7 GB/s using the current methodology), minimizing bottlenecks and enabling high-speed communication. The FPGA's high parallelism and customizable nature enhance the acceleration of computationally intensive SAR algorithms, keeping the high parallelism through multiple IPs and optimizations. The scalability of the implementation allows for handling large data sizes, ensuring efficient processing, and up to five FPGAs can be used per system. However, it is important to note that FPGA accelerators also have disadvantages, including logic resource consumption and cost implications due to the need for larger FPGA devices. Additionally, the lower operating frequency of FPGAs compared to CPUs or GPUs may impact real-time processing capabilities. Despite these limitations, the FPGA+CPU implementation with a PCIe interface remains a promising solution for on-board SAR processing, enabling efficient and high-performance data processing.

In order to exemplify the advantages of the suggested methodology, we refer to the VEC-FFT algorithm [18]. VEC-FFT retrieves results in reversed bit order, and the reordering stage consumes more than half of the total clock cycles. Therefore, an FPGA accelerator for the data reordering function is developed that communicates via PCIe with a CPU, where the rest of the VEC-FFT algorithm is implemented.

Modular and scalar implementations are possible, meaning that the IPs are interchangeable and/or have multiple instantiations. For demonstration purposes, the original (reordering) IP is replaced by an IP that performs the computationally expensive matrix transpose. Experimental results show the capabilities of the introduced methodology, together with the main advantages of parallel processing. The proposed solution allows for the use of different PCIe technologies according to specific needs. Being scalable, up to five FPGAs can be employed, each with multiple IPs.

The comparisons between CPU and FPGA for VEC-FFT re-encoding reveal similar performance levels. However, it is important to consider the limitations of the ZC706 board used in these comparisons. The ZC706 board operates at a working frequency of 100 MHz and utilizes PCIe 2.0 with 4 lanes. Despite these limitations, the FPGA still demonstrates competitive performance. To further enhance performance, one could explore the use of an FPGA with a higher working frequency or leverage more advanced PCIe technology.

Moreover, in the case of matrix transpose, the FPGA outperforms the CPU, highlighting the inherent advantages of FPGA-based processing. Even with the limitations imposed by the ZC706 board, the FPGA demonstrates favorable performance in this scenario. This underscores the potential of FPGAs for accelerating computational tasks such as matrix operations.

The current research and methodology provide a practical guide for developing FPGA accelerators quickly and efficiently using HLS. The creation of modular IPs offers the advantage of reusability, allowing the same IP to be utilized for different SAR algorithms. Furthermore, the integration of the RIFFA framework offers a robust infrastructure for efficient communication between the FPGA and the CPU, significantly minimizing bottlenecks and enabling parallel processing capabilities. The direct DMA provided by RIFFA facilitates seamless data transfer between the FPGA and the CPU, further enhancing the overall performance of the system.

This research serves as a valuable resource for developers seeking to harness the power of FPGAs for accelerating SAR processing and similar applications. By following the presented methodology, developers can achieve faster development cycles, improved performance, and enhanced efficiency in FPGA-based accelerator designs.

Author Contributions: Conceptualization, E.I.B.-L., G.D.M.-d.-C.-B. and S.O.-C.; methodology, E.I.B.-L. and S.O.-C.; software, E.I.B.-L., S.O.-C. and M.S.; validation, E.I.B.-L., G.D.M.-d.-C.-B., S.O.-C., M.S., J.R. and A.R.; formal analysis, E.I.B.-L., G.D.M.-d.-C.-B., S.O.-C., M.S., J.R. and A.R.; investigation, E.I.B.-L. and G.D.M.-d.-C.-B.; resources, E.I.B.-L., S.O.-C., J.R. and A.R.; data curation, E.I.B.-L., G.D.M.-d.-C.-B. and M.S.; writing—original draft preparation, E.I.B.-L.; writing—review and editing, E.I.B.-L., G.D.M.-d.-C.-B., S.O.-C., M.S., J.R. and A.R.; supervision, G.D.M.-d.-C.-B. and S.O.-C.; project

administration, G.D.M.-d.-C.-B., and S.O.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available in this article.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. System resources used for this research.

Processor	Name	Intel Core i3-4160
	Code Name:	Haswell
	Cores:	2
	L1 Data cache:	64 KB
	L1 Inst cache:	64 KB
	L2 cache:	512 KB
	L3 cache:	3072 KB
	Base frequency:	3.6 GHz
Memory	Type	DDR 3
	Capacity	16 (2 × 8) GB
	Frequency	667 MHz
	CL	9 Clocks
	tRC	49.5 ns
	rRFC	260 ns
	tRAS	36 ns
Operating System	Linux	
	Distribution	Ubuntu 18.04 LTS
	Version	18.04 LTS (64 bits)

Appendix B

Table A2. System resources in [18].

Processor	Name	Intel Core i7 8700
	Code Name:	Coffee Lake
	Cores:	6
	L1 Data cache:	32 KB
	L1 Inst cache:	32 KB
	L2 cache:	256 KB
	L3 cache:	12,288 KB
	Base frequency:	3.2 GHz
Memory	Type	DDR 4
	Capacity	64 (2 × 32) GB
	Frequency	2333 MHz
	CL	15 Clocks
	tRC	15 Clocks
	rRFC	15 Clocks
	tRAS	15 Clocks
Operating System	Linux	
	Distribution	Ubuntu 18.04 LTS
	Version	18.04 LTS (64 bits)

References

1. Moreira, A.; Prats-Iraola, P.; Younis, M.; Krieger, G.; Hajnsek, I.; Papathanassiou, K.P. A tutorial on synthetic aperture radar. *IEEE Geosci. Remote Sens. Mag.* **2013**, *1*, 6–43. [CrossRef]
2. Curlander, J.C.; McDonough, R.N. *Synthetic Aperture Radar*; Wiley: New York, NY, USA, 1991; Volume 11.
3. Reigber, A.; Scheiber, R.; Jager, M.; Prats-Iraola, P.; Hajnsek, I.; Jagdhuber, T.; Papathanassiou, K.P.; Nannini, M.; Aguilera, E.; Baumgartner, S.; et al. Very-high-resolution airborne synthetic aperture radar imaging: Signal processing and applications. *Proc. IEEE* **2021**, *101*, 759–783. [CrossRef]
4. Yu, W.; Xie, Y.; Lu, D.; Li, B.; Chen, H.; Chen, L. Algorithm implementation of on-board SAR imaging on FPGA+ DSP platform. In Proceedings of the 2019 IEEE International Conference on Signal, Information and Data Processing, Chongqing, China, 11–13 December 2019; pp. 1–5.
5. Zhang, Y.; Qu, T. Focusing highly squinted missile-borne SAR data using azimuth frequency nonlinear chirp scaling algorithm. *J. Real-Time Image Process.* **2021**, *18*, 1301–1308. [CrossRef]
6. Chen, X.; Yi, T.; He, F.; He, Z.; Dong, Z. An Improved Generalized Chirp Scaling Algorithm Based on Lagrange Inversion Theorem for High-Resolution Low Frequency Synthetic Aperture Radar Imaging. *Remote Sens.* **2019**, *11*, 1874. [CrossRef]
7. Wang, S.; Zhang, S.; Huang, X.; An, J.; Chang, L. A highly efficient heterogeneous processor for SAR imaging. *Sensors* **2019**, *19*, 3409. [CrossRef] [PubMed]
8. Li, B.; Li, C.; Xie, Y.; Chen, L.; Shi, H.; Deng, Y. A SoPC based fixed point system for spaceborne SAR real-time imaging processing. In Proceedings of the 2018 IEEE High Performance extreme Computing Conference, Waltham, MA, USA, 25–27 September 2018; pp. 1–6.
9. Lovelly, T.M.; George, A.D. Comparative analysis of present and future space-grade processors with device metrics. *J. Aerosp. Inf. Syst.* **2017**, *14*, 184–197. [CrossRef]
10. Towfic, Z.; Ogbe, D.; Sauvageau, J.; Sheldon, D.; Jongeling, A.; Chien, S.; Mirza, F.; Dunkel, E.; Swope, J.; Ogut, M. Benchmarking and testing of Qualcomm snapdragon system-on-chip for JPL space applications and missions. In Proceedings of the 2022 IEEE Aerospace Conference (AERO), Big Sky, MT, USA, 5–12 March 2022; IEEE: Piscataway, NJ, USA; pp. 1–12.
11. Pavlov, V.A.; Belov, A.A.; Tuzova, A.A. Implementation of synthetic aperture radar processing algorithms on the Jetson TX1 platform. In Proceedings of the 2019 IEEE International Conference on Electrical Engineering and Photonics (EExPolytech), St. Petersburg, Russia, 17–18 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 90–93.
12. Svedin, J.; Bernland, A.; Gustafsson, A.; Claar, E.; Luong, J. Small UAV-based SAR system using low-cost radar, position, and attitude sensors with onboard imaging capability. *Int. J. Microw. Wirel. Technol.* **2021**, *13*, 602–613. [CrossRef]
13. Schlemon, M.; Scheiber, R.; Baumgartner, S.; Joshi, S.K.; Jaeger, M.; Pasch, S. On-board Processing Architecture of DLR's DBFSAR/V-SAR System. In Proceedings of the EUSAR 2022, 14th European Conference on Synthetic Aperture Radar, Leipzig, Germany, 25–27 July 2022; pp. 1–5.
14. *High-Level Synthesis*, 1st ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–15.
15. Pelcat, M.; Bourrasset, C.; Maggiani, L.; Berry, F. Design productivity of a high level synthesis compiler versus HDL. In Proceedings of the 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, Agios Konstantinos, Greece, 17–21 July 2016; pp. 140–147.
16. Zamiri, E.; Sanchez, A.; Yushkova, M.; Martínez-García, M.S.; de Castro, A. Comparison of different design alternatives for hardware-in-the-loop of power converters. *Electronics* **2021**, *10*, 926. [CrossRef]
17. Jacobsen, M.; Richmond, D.; Hogains, M.; Kastner, R. RIFFA 2.1: A reusable integration framework for FPGA accelerators. *ACM Trans. Reconfigurable Technol. Syst.* **2015**, *8*, 1–23. [CrossRef]
18. Schlemon, M.; Naghmouchi, J. Fft optimizations and performance assessment targeted towards satellite and airborne radar processing. In Proceedings of the 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing, Porto, Portugal, 9–11 September 2020; pp. 313–320.
19. Wu, J.; Xu, Y.; Zhong, X.; Sun, Z.; Yang, J. A three-dimensional localization method for multistatic sar based on numerical range-doppler algorithm and entropy minimization. *Remote Sens.* **2017**, *9*, 470. [CrossRef]
20. Işıker, H.; Özdemir, C. Adaptation of stepped frequency continuous waveform to range-Doppler algorithm for SAR signal processing. *Digit. Signal Process.* **2020**, *106*, 102826. [CrossRef]
21. Wu, S.; Wang, H.; Li, C.; Liu, X.; Fang, G. A modified Omega-K algorithm for near-field single-frequency MIMO-arc-array-based azimuth imaging. *IEEE Trans. Antennas Propag.* **2021**, *69*, 4909–4922. [CrossRef]
22. Liu, W.; Sun, G.C.; Xia, X.G.; You, D.; Xing, M.; Bao, Z. Highly squinted MEO SAR focusing based on extended Omega-K algorithm and modified joint time and Doppler resampling. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 9188–9200. [CrossRef]
23. Wang, C.; Su, W.; Gu, H.; Yang, J. Focusing bistatic forward-looking synthetic aperture radar based on an improved hyperbolic range model and a modified Omega-K algorithm. *Sensors* **2019**, *19*, 3792. [CrossRef]
24. Rao, K.R.; Kim, D.N.; Hwang, J.J. *Fast Fourier Transform: Algorithms and Applications*; Springer: Dordrecht, The Netherlands, 2010; Volume 32.
25. Intel Corporation. Intrinsics Guide. Available online: https://software.intel.com/sites/landingpage/IntrinsicsGuide/#=undefined&techs=FMA&expand=2297,3924,4202,2607,2755,2553&text=256 (accessed on 14 April 2023).
26. Feist, T. Vivado design suite. *White Pap.* **2012**, *5*, 30.

27. Xilinx. *Vivado Design Suite User Guide: High-Level Synthesis (UG902)*; Xilinx, Inc.: San Jose, CA, USA, 2020.
28. ZC706 Evaluation Board User Guide v1.8. 2019. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf (accessed on 14 April 2023).
29. SDAccel Development Environment Help for 2019.1. 2019. Available online: https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/hls-pragmas-okr1504034364623.html#fde1504034360078 (accessed on 14 April 2023).