Onboard Processing of Synthetic Aperture Radar Backprojection Algorithm in FPGA

David Mota, Helena Cruz, Pedro R. Miranda, Rui Policarpo Duarte, José T. de Sousa, Horácio C. Neto, and Mário P. Véstias

***Abstract*—Synthetic aperture radar is a microwave technique to extracting image information of the target. Electromagnetic waves that are reflected from the target are acquired by the aircraft or satellite receivers and sent to a ground station to be processed by applying computational demanding algorithms. Radar data streams are acquired by an aircraft or satellite and sent to a ground station to be processed in order to extract images from the data since these processing algorithms are computationally demanding. However, novel applications require real-time processing for real-time analysis and decisions and so onboard processing is necessary. Running computationally demanding algorithms on onboard embedded systems with limited energy and computational capacity is a challenge. This article proposes a configurable hardware core for the execution of the backprojection algorithm with high performance and energy efficiency. The original backprojection algorithm is restructured to expose computational parallelism and then optimized by replacing floating-point with fixed-point arithmetic. The backprojection core was integrated into a system-on-chip architecture and implemented in a field-programmable gate array. The proposed solution runs the optimized backprojection algorithm over images of sizes 512×512 and 1024×1024 in 0.14 s (0.41 J) and 1.11 s (3.24 J), respectively. The architecture is $2.6\times$ faster and consumes $13\times$ less energy than an embedded Jetson TX2 GPU. The solution is scalable and, therefore, a tradeoff exists between performance and utilization of resources.**

***Index Terms*—Field-programmable gate arrays (FPGA), onboard processing, real-time, synthetic aperture radar (SAR).**

Manuscript received February 25, 2022; revised April 2, 2022; accepted April 19, 2022. Date of publication April 25, 2022; date of current version May 16, 2022. This work was supported in part by the National Funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and PTDC/EEI-HAC/31819/2017 (SARROCA), and by project IPL/2021/smartSPACE_ISEL through Instituto Politécnico de Lisboa. The work with Helena Cruz would like to acknowledge Fundação para a Ciência e a Tecnologia for the support through under Grant SFRH/BD/144133/2019. *(Corresponding author: Mário P. Véstias.)*

David Mota is with Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, 1959-007 Lisbon, Portugal (e-mail: david.mota@isel.pt).

Helena Cruz, Pedro R. Miranda, José T. de Sousa, and Horácio C. Neto are with INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, 1000-029 Lisboa, Portugal (e-mail: helena.cruz@tecnico.ulisboa.pt; pmiranda@iobundle.com; jose.desousa@inesc-id.pt; hcn@inesc-id.pt).

Rui Policarpo Duarte is with INESC-ID and Celestia Portugal, 1000-029 Lisboa, Portugal (e-mail: rui.duarte@tecnico.ulisboa.pt).

Mário P. Véstias is with INESC-ID and Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, 1959-007 Lisboa, Portugal (e-mail: mario.vestias@isel.pt).

Digital Object Identifier 10.1109/JSTARS.2022.3169828

I. Introduction

SYNTHETIC aperture radar (SAR) is a remote sensing technique widely used to monitor the surface of the Earth with application in many different areas, including ships and oil spills tracking, terrain erosion, drought and landslides, deforestation, and fires [1]. One of the main features of SAR that makes it a very attractive technique for remote sensing is its capacity of operating under adverse whether conditions with clouds, smoke, or rain and without a light source.

SAR sensors can be installed on satellites, aircrafts, or unmanned aerial vehicle, depending on the application. Data collected from these sensors must be processed to obtain images from the Earth surface. The backprojection (BP) [2], [3] time-domain algorithm is one the most used algorithms for SAR image formation due to its robustness and quality of results.

SAR algorithms, and BP in particular, are quite computation and memory intensive and, therefore, processing is usually done at a ground station. However, running the generation of images onboard opens a new vast set of applications over recovered images, such as, for example, image classification and detection. This would permit to undertake real-time analysis and decisions onboard.

Using high-performance computing platforms, such as multicore processors and graphics processing units (GPUs), to run SAR image formation algorithms is not feasible onboard since they require high power, while onboard computing systems are usually low power.

An alternative is to consider dedicated hardware architectures implemented in field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC). These allow the design of an optimized hardware architecture, which improves performance and reduces power. Compared to ASIC, FPGAs are less performing and require more power. However, they allow redesigning the algorithm after deployment and are less costly for medium volume production.

Therefore, FPGAs are attractive devices for onboard processing systems since they offer high performance, compact size, reduced weight, and low power, while at the same time allow onboard hardware redesign to cope with the needs of different missions and algorithmic modifications.

The hardware architecture designed with FPGAs can be fully optimized for each particular algorithm, such as the BP algorithm. The data representation and the arithmetic operators can all be explored and designed to obtain the best architectures with particular tradeoffs between hardware resources, performance, and energy. This is particularly true for hardware accelerators, where operations are preferably implemented using

fixed-point arithmetic due to the overhead introduced by the implementation of floating-point units. Therefore, this work considers a full optimization of the representation of the variables and the arithmetic operators.

Another major aspect of custom-designed hardware architectures is the right balance between computation and communication. The BP projection algorithm requires memory transfers of large volume of data. As the parallel computation increases, the pressure over the available memory bandwidth increases up to a point where increasing the parallelism leads to no further performance improvements. To reduce the memory bandwidth pressure and allow further parallelization, data must be cached on-chip and reutilized as much as possible. The memory architecture is, therefore, a major aspect when designing these algorithms. The work proposed in this article redesigns the BP algorithm so that a tradeoff is established between on-chip memory (OCM) and parallelism, that is, the OCM size determines the degree of parallelism of the design. The integrated design of software and hardware, where the software algorithm is reorganized oriented by the hardware design, further improves the computing architecture.

The main contribution of this work is, therefore, a novel hardware architecture for the execution of the BP algorithm for SAR image generation. The proposed architecture is configurable to achieve different tradeoffs between performance and hardware cost. The project of the architecture integrates several contributions, namely BP algorithm rescheduling for balanced computation and communication and data type analysis and design to improve the efficiency of the architecture.

The proposed core was integrated in a system-on-chip (SoC) architecture with a soft processor, implemented in a medium-density FPGA and tested with synthetic and real images obtained from the ARFL dataset.[1] The system was compared to the execution of the algorithm in a desktop i7 processor, an embedded GPU with a Jetson TX2 board, a NVIDIA GeForce RTX 2060 GPU, and previous FPGA design for BP processing.

The rest of this article is organized as follows. Section II describes the related work on algorithms and architectures for the fast execution of the BP algorithm. Section III presents the BP algorithm in detail. Section IV describes the design of the BP in hardware, including algorithm rescheduling, data type optimization, and hardware design. Section V details the performance and resources of the proposed system. Finally, Section VI concludes this article.

II. Related Work

Two types of algorithms are used commonly for SAR image formation: frequency-domain and time-domain algorithms [4]. Frequency-domain algorithms have a computational complexity of $\mathcal{O}(N^2 \log N)$, for N pulses of $N \times N$ samples of an image, lower than the $\mathcal{O}(N^3)$ complexity of time-domain algorithms. The disadvantage of the frequency-domain algorithms is that they are not applicable to all cases and require multiple assumptions, including geometric approximations and particular flying trajectories. Other associated problems include a nonperfect motion compensation that can lead to some image focusing problems and interpolation in the frequency domain that can generate some unexpected elements in the formed image.

In spite of having a bigger complexity, time-domain algorithms are not affected by the limitations associated with the frequency-domain algorithms. The BP [2], [3] time-domain algorithm does not require a separate motion compensation and is not subject to the constraints mentioned before. Besides, it works with the spotlight, stripmap, and scan imaging modes. The BP algorithm implementation for circular SAR designed by LeRoy Gorham and Linda Moore [2] is a ready to use MATLAB implementation that accepts different dataset formats such as the Gotcha Volumetric SAR Dataset [5], Backhoe Data Dome [6], and GMTI Challenge Problem [7]. The PERFECT Suite [8] provides an implementation of the BP algorithm, among others, with compute unified device architecture and OpenMP versions.

The BP algorithm is quite computation and memory intensive. Therefore, it has been modified to speedup the SAR image formation process. The fast BP algorithm [9], the fast factorized BP algorithm [10], and the accelerated BP algorithm [11] are examples of the improved BP algorithm. In [12], the BP algorithm was designed with fixed-point arithmetic and executed in a CPU, with performance improvements up to 25%, depending on the image size, with negligible image quality reduction. The time to process the SAR data was reduced from 84 to 75 s. These optimizations reduce the computation time of SAR data but they are still unable to deal with real-time requirements.

To further reduce the computation times of the BP algorithm, multiprocessing computing architectures are used. The BP algorithm is highly parallelizable, so multiple operations can be executed in parallel, which reduces the execution time of the algorithm. GPUs [13]–[18] and multicore processors [19] are the two most common multicore processing architectures considered for BP algorithm speedup.

In [19], several approximate strength reduction optimizations, such as quadratic polynomial approximations, Taylor series as square root calculation method, and trigonometric function strength reduction, are used to reduce the computational load of the algorithm. Large images are processed in one second using a multicore platform with 16 processors. In [15], the authors note that in practice not all pulses are going to contribute to every pixel of a SAR image. This is taken into consideration to reduce the number of computations.

High-performance computing platforms require high power, which is not available in onboard computing systems which are low power. Embedded GPUs can be considered since they have lower power requirements. In [16], a mobile graphics with a peak performance close to 1 TFLOPS is used to run the BP algorithm in about 3 s. The work in [18] reduces the processing time of the fast factorized BP in a Jetson TX2 GPU for images of size $2k \times 2k$ from 35 to 5 s, depending on the stage factor of the algorithm. Embedded GPUs tradeoff power by computing capacity and are still based on general purpose architectures with some energy inefficiencies.

FPGAs and ASIC devices allow the design of custom hardware accelerators with better performance and energy. ASICs are faster and require lower energy than FPGAs but are expensive and do not allow hardware modifications after fabrication, such as the FPGAs.

In [20], the BP algorithm is implemented in FPGA with a mesh of 64 processor cores to provide real-time processing.

[1] [Online]. Available: https://www.sdms.afrl.af.mil/index.php?collection=gotcha

Cholewa *et al.* [21] developed a BP module to generate one line of the final image at a time, looping over all pulses for each line. The results obtained show that the implementation scales almost linearly with the parallelization factor. David [22] provides an automatic framework based on OpenCL to map the BP algorithm in FPGA. The solution explores the available parallelism of the algorithm and shows good results when compared to designs generated from hardware description languages. Another implementation of the BP algorithm in FPGA [23] considers multiple independent core units that receive raw data and generate a pixel contribution to be accumulated to the current pixel value. In this implementation, an Arria-V SoC from Altera is used, which also integrates an ARM Cortex-A9 dual-core processor. The design is able to process an image in 120 ms. The work in [18] also runs the fast factorized BP algorithm in two different FPGAs reaching execution times for images of size $2k \times 2k$ range from 0.8 to 42 s, depending on the target FPGA and the algorithm configuration. Data in the FPGA are represented with 16-bits. In [24], a hardware/software solution is proposed for SAR image processing. The proposal partitions the solution in hardware and software, mapping some of the most computation intensive parts of the algorithm to hardware.

The utilization of 16-bit data in the FPGA design from [18] is a major design optimization. One of the main concerns when implementing algorithms using accelerators is the tradeoff between performance and precision. GPUs, for example, tend to use single or half-precision instead of double-precision floating-point either because the devices do not support it or because of the overhead introduced. However, assuming a constant fixed-point size for all variables is inefficient. A constant size may not be enough for some variables, which reduces precision, or may be too much for some variables, which requires unnecessary hardware. Instead of a fixed-point representation for the whole variables of the algorithm, this work considers a full optimization of the representation of the variables.

Memory organization is another major aspect of any computing platform. Previous FPGA designs rely on a high external memory bandwidth to feed multiple parallel cores. To reduce the memory bandwidth requirements as the parallelism increases, data must be properly cached on-chip and reutilized. The architecture proposed in this article redesigns the BP algorithm to allow a scalable architecture without the necessity to increase the memory bandwidth.

III. Backprojection Algorithm

The BP algorithm is a SAR image formation algorithm that converts radar echo data in a SAR image. The algorithm determines the contribution of each reflected pulse for each pixel on the output SAR image. The BP algorithm takes as input the location of the image platform for each pulse, the location of the output pixels, and the SAR dataset. In the description of the BP algorithm, the following nomenclature is considered.

R	Distance from platform location to each pixel.
$R0$	Distance to the first range bin. This value is constant.
x_k, y_k, z_k	Radar platform location in Cartesian coordinates.
x, y, z	Pixel location in Cartesian coordinates.
r_c	Range to center of the swath from radar platform.

Algorithm 1: BP algorithm pseudocode, based on [8].

for all Y pixels y **do**
  for all X pixels x **do**
    acc $\leftarrow 0$
    for all pulses p **do**
      $R \leftarrow \sqrt{(x-x_k)^2+(y-y_k)^2+(z-z_k)^2} - r_c$
      $b \leftarrow \lfloor (R-R0)/\Delta R \rfloor$
      $w \leftarrow (R-R0)/\Delta R - b$
      $g_{x,y} \leftarrow (1-w)\cdot g(p,b) + w \cdot g(p,b+1)$
      $mf \leftarrow \cos(2\cdot k_u \cdot R) + i\sin(2\cdot k_u \cdot R)$
      prod $\leftarrow g_{x,y} \times mf$
      acc $\leftarrow$ acc + prod
    end for
    $f(x,y) \leftarrow$ acc
  end for
end for

$f(x,y)$	Value of each pixel (x,y).
θ_k	Aperture point.
ω	Minimal angular velocity of wave.
$g_{x,y}(r_k,\theta_k)$	Wave reflection received at r_k at θ_k [calculated using the linear interpolation in (3)].
Data(p,n)	Wave sample of pulse p and range bin n.

The BP algorithm performs a sequence of steps for each input point (pixel, pulse) as follows.

1) It computes the distance from the radar platform to the pixel under analysis
$$R = \sqrt{(x-x_k)^2+(y-y_k)^2+(z-z_k)^2} - r_c. \quad (1)$$
2) It converts the distance to an associated range position from the dataset of received echoes
$$b = \left\lfloor \frac{R-R0}{\Delta R} \right\rfloor. \quad (2)$$
Assuming that range bins are equally spaced, the factor $\frac{1}{R(n+1)-R(n)}$ can be replaced by multiplication with a constant $1/\Delta R$.
3) It obtains the samples at the computed range via linear interpolation as [23]
$$g_{x,y} = (1-w)\times g(p,b) + w \times g(p,b+1) \quad (3)$$
where $w = \frac{R-R0}{\Delta R} - b$.
4) It computes the complex exponential for the matched filter from R [see (4)]
$$e^{i\omega 2|\overrightarrow{r_k}|} = \cos(2\cdot\omega\cdot R) + i\sin(2\cdot\omega\cdot R). \quad (4)$$
5) It scales the sampled value by the matched filter to determine the pixel contribution
$$\text{prod}(x,y,k) = g_{x,y}\cdot e^{i\cdot\omega\cdot 2\cdot|R|}. \quad (5)$$
6) It accumulates the contribution into the pixel. The final value of each pixel is given
$$f(x,y) = \sum_k g_{x,y}\cdot e^{i\cdot\omega\cdot 2\cdot R}. \quad (6)$$

The BP algorithm with the steps enumerated above is described in Algorithm 1, where $k_u = \frac{2\pi f_c}{c}$ represents the wave number, f_c is the carrier frequency of the waveform, and c is the speed of light.

The computational complexity of the algorithm for an image of size $iy \times ix$ is proportional to $iy \times ix \times p$ that increases quadratically with the size of the image and linearly with the number of pulses.

To improve the interpolation quality of the BP algorithm, it is common to upsample data in the range dimension prior to BP. This work considers $8\times$ upsampling via FFT/IFFT. The input data are first translated to the frequency domain using an FFT, the result is zero-padded and then converted back to the time domain using an IFFT. The BP algorithm operates on the upsampled data.

IV. Design and Implementation of the Hardware Architecture

In this section, the hardware architecture to run the BP algorithm on FPGA is described. The methodology to design the system has the following steps.

1) Algorithm rescheduling to improve memory access: The algorithm was reorganized to improve data accesses from external memory and to increase OCM reuse. Loop tiling exploits spatial and temporal locality allowing data to be accessed in tiles permitting to execute the same operations over a block of data stored in local memory.
2) Data type optimization: The data type representations were converted from single- and double-precision floating point to fixed point. This allows to reduce the complexity of arithmetic operators implemented in hardware and the memory required to store data. The fixed-point representation (number of integer and fractional bits) was found for each variable in order to achieve an algorithm accuracy close to the signal-to-noise ratio (SNR) specified by the designer. The fixed-point format determines the hardware area of the final solution as well as the performance.
3) Design the upsampling and the BP algorithm in hardware: A dedicated configurable accelerator is designed to run both the upsampling and the BP algorithm.

A. Algorithm Rescheduling

The output pixels of the BP algorithm can be calculated independently of each other, so the algorithm is highly parallelizable and the outcome is not affected by the order in which pixels are calculated.

The two main methods of improving an algorithm execution in hardware are parallelization of computations and datapath pipeline. A datapath pipeline achieves a throughput of one output pixel per clock cycle, but this requires reading a complex data point from external memory in a single cycle. This is difficult to achieve in the BP algorithm since in each clock cycle two complex data points, $g(p, b)$ and $g(p, b + 1)$, would have to be retrieved from memory for each pulse p. Since the reading order of pulses may not be sequential (depends on b), several clock cycles are needed to read each data point from external memory. A turnaround solution consists of storing all bins of a pulse in on-chip random access memory (RAM) memory. This on-chip allows random cycle access to data which permits pipeline execution.

Algorithm 2: New scheduling for the BP algorithm where the contribution of a single pulse is calculated for all output pixels in a line.

for all pixels in Y y **do**
  for allpixels in X x **do**
    $\text{acc}(x) \leftarrow 0$
  end for
  for all pulses p **do**
    for all pixels in X x **do**
      $R \leftarrow \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} - r_c$
      $b \leftarrow \lfloor (R - R0)/\Delta R \rfloor$
      $w \leftarrow (R - R0)/\Delta R - b$
      $g_{x,y} \leftarrow (1 - w) \cdot g(p, b) + w \cdot g(p, b + 1)$
      $mf \leftarrow \cos(2 \cdot k_u \cdot R) + i \sin(2 \cdot k_u \cdot R)$
      $\text{prod} \leftarrow g_{x,y} \times mf$
      $\text{acc}(x) \leftarrow \text{acc}(x) + \text{prod}$
    end for
  end for
  for all pixels in x **do**
    $f(x, y) \leftarrow \text{acc}(x)$
  end for
end for

The execution throughput of the algorithm can further be improved with multiple pipelined datapaths. Using OCM to cache data permits multiple accesses to data points using multiport memories. Memory with multiple ports can be implemented on FPGA using OCM and data replicated in multiple memories to allow multiple parallel accesses.

The large data volume would require a large OCM to store all data on-chip that may not be enough even in large FPGA devices. For example, considering an image with 512 single floating-point complex pulses upsampled eight times needs $512 \times 512 \times 8 \times 4 \times 2 = 16$ MBytes of OCM. Therefore, only a subset of data can be loaded at a time. Since there are no data dependencies between the calculation of different output pixels and there are no constraints over the order with each output pixels can be produced, input data $g(p, b)$ can be reused by calculating the contribution of each pulse for a set of pixels (see Algorithm 2).

This scheduling of the algorithm calculates the contribution of each pulse in all pixels in a line before proceeding to the next pulse. Therefore, the data associated with a single pulse have been reused a number of times equal to the number of pixels in a line. This reduces the pressure over the link to external memory to retrieve data pulses since the data are cached in OCM and reused multiple times. It also allows parallel pixel calculation since data can be easily replicated to offer multiple memory ports.

Another aspect of this algorithmic organization is that data loading can be done in parallel with data processing, that is, data for the next pulse can be loaded while the data of the present pulse are being processed. The execution time of each iteration is, therefore, determined by the highest latency between data transfer and/or upsampling and data processing. The most efficient solution is the one with a balanced latency between communication/upsampling and processing.

In case the pipelined data processing is slower, parallel datapaths can be used. Otherwise, data reuse can be augmented to

Algorithm 3: New scheduling with tiling for the BP algorithm where the contribution of a single pulse is calculated for all output pixels in a set of lines.

for all ik in Y/TL y **do**
  for all pixels in X*Lines xl **do**
    $acc(xl) \leftarrow 0$
  end for
  for all pulses p **do**
    for all pixels in TL l **do**
      for all pixels in X x **do**
        $R \leftarrow \sqrt{(x-x_k)^2+(y-y_k)^2+(z-z_k)^2} - r_c$
        $b \leftarrow \lfloor (R-R0)/\Delta R \rfloor$
        $w \leftarrow (R-R0)/\Delta R - b$
        $g_{x,y} \leftarrow (1-w)\cdot g(p,b) + w \cdot g(p,b+1)$
        $mf \leftarrow \cos(2\cdot k_u \cdot R) + i\sin(2\cdot k_u \cdot R)$
        prod $\leftarrow g_{x,y} \times mf$
        acc$(x,l) \leftarrow$ acc(x,l) + prod
      end for
    end for
  end for
  for all pixels in Lines l **do**
    for all pixels in x **do**
      $f(x,l) \leftarrow$ acc(x,l)
    end for
  end for
end for

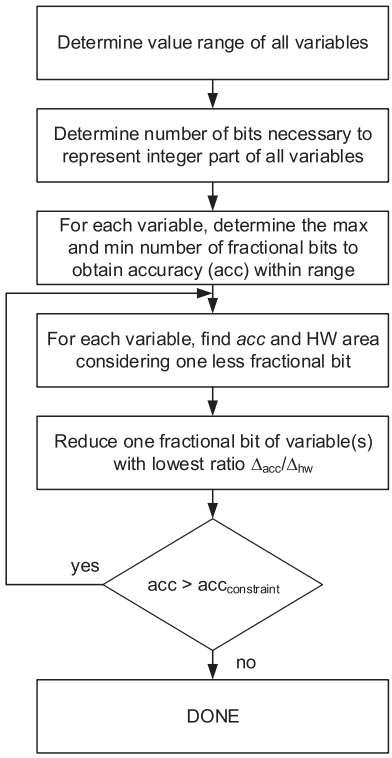

Fig. 1. Methodology for conversion of floating-point representation to fixed-point exploring the tradeoff between accuracy and hardware utilization.

increase the time available for data transfer and upsampling. Increasing data reused can be easily achieved with the BP algorithm by increasing the number of pixels processed for each pulse (see Algorithm 3).

In the new algorithm rescheduling, each pulse is used for all pixels in a subset of lines TL, that is, a tiling of TL lines. So, data pulse reuse is increased TL×, which reduces the pressure over the external memory access and upsampling calculation by a proportional amount. The drawback of the solution is that more OCM is necessary to store partial accumulations for all pixels under processing.

B. Data Type Optimization

The original implementation of the BP algorithm uses double-precision floating-point arithmetic. This offers a large dynamic range of values with high precision. However, it is more computationally intensive and occupies more memory than other floating-point representations, such as float or half-float, and fixed-point or integer representations. When implemented in hardware, the latency and the occupied resources are larger than the other simpler implementations.

Therefore, the hardware implementation of an algorithm is usually preceded by an analysis of the dynamic range of all data so that hardware-friendly data representations can be used. In this work, fixed-point representations were used to replace double-precision floating-point with negligible accuracy loss. To achieve this, the methodology represented in Fig. 1 was followed.

The methodology considers two assessment metrics: hardware area and SNR. Other metrics can be considered by extending the assessment equation. Three metrics can be used for hardware area: lookup tables (LUTs), DSPs, and BRAMs. Some modules of the datapath use only LUTs so the number of occupied LUTs alone is considered as the hardware metric. Of course, a component can be implemented with a variable number of DSPs, LUTs, and BRAMs. For example, a multiplier can be implemented with both LUTs and DSPs, with LUTs only or with DSPs only. However, to reduce the complexity of the design exploration problem, a medium DSP usage implementation was assumed, where the utilization of both LUTs and DSPs is balanced. Further exploration of this aspect can be considered in future developments.

Performance is not considered explicitly as a metric because the circuit is fully pipelined. Changing the bitwidth of a fixed-point number may influence the critical path and, consequently, the maximum operating frequency. However, reducing the bitwidth has an impact on both area and performance. Since the area is being optimized it will also improve the performance. So, the optimization is guided by the area and the operating frequency is determined for a final solution. The results of all architectures assume a fixed frequency for all designs, even knowing that smaller designs can run at a higher frequency. The frequency is mostly influenced by the routing when the FPGA has a high percentage of utilization. These factors are difficult to include in the design exploration tool, so they are omitted. Therefore, performance was not considered as an assessment metric.

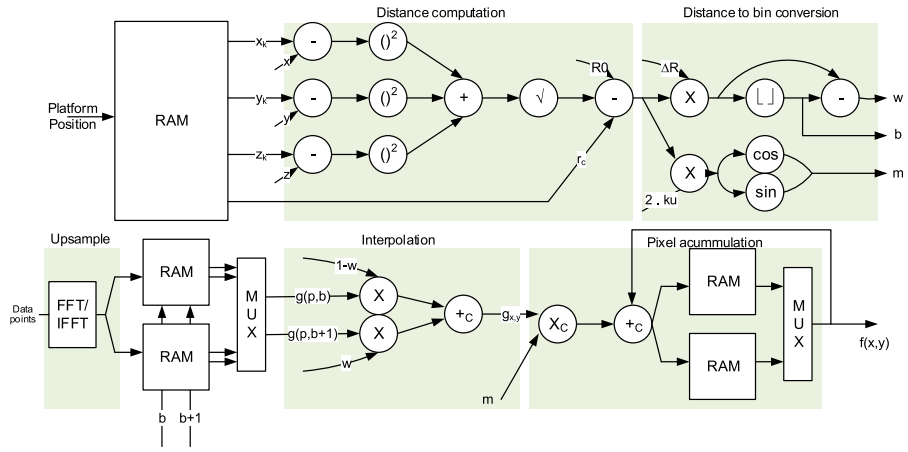

Fig. 2. Hardware datapath of the hardware module that implements the BP algorithm.

The hardware area of the circuit used in the methodology is obtained from models of the hardware implementation of arithmetic operators. A model was considered for each type of arithmetic unit: addition/subtraction, accumulation, multiplication, square-root, and trigonometric functions, sin/cos, within the variation range of the operands. For example, the number of LUTs in addition equals the operand sizes, the number of LUTs. For the remaining operations, a table with the number of LUTs and DSPs was determined for all operand sizes within the range of the variables involved in each operation.

Initially, the value range of all variables is determined for a set of images. Considering the integer part of the highest values of each variable, the number of necessary bits to represent the integer part is found. Then, considering each variable at a time, we find the maximum and minimum number of bits to represent the fractional part. The maximum number corresponds to the necessary number of bits to keep the accuracy loss as small as possible. The minimum number of bits corresponds to those bits necessary to obtain an accuracy higher than a user specified value. This value determines the achievable reduction in hardware resources.

Then, starting with the highest number of fractional bits for all variables, the number of fractional bits is reduced. The variable with the lowest ratio $\frac{\Delta_{\text{acc}}/\text{Total}_{\text{acc}}}{\Delta_{\text{hw}}/\text{Total}_{\text{hw}}}$ is chosen to reduce one fractional bit. Here, Δ_{acc} and Δ_{hw} represent the variations in accuracy and hardware area estimation, respectively. $\text{Total}_{\text{acc}}$ and Total_{hw} represent the accuracy and total hardware area estimation, respectively, of the algorithm.

The process is iteratively repeated until achieving the lowest acceptable accuracy defined by the user. The range of values for the number of fractional bits determines the range of the accuracy and the utilization of hardware resources.

C. Hardware Design of the BP Algorithm

This section presents the proposed hardware design of Algorithm 3 described in Section IV-A. The proposed hardware module, SAR_IP, is configurable in the number of fractional bits, the tiling factor, and the number of parallel datapaths. The full circuit with a single datapath is represented in Fig. 2.

The circuit is fully pipelined to increase the throughput and several on-chip buffers for data reuse and communication/computation overlap. Fig. 2 identifies the main modules but they are all connected in a stream-like fully pipelined architecture. Synchronization only happens at the input and output of data with valid/ready signals. All operators, including the upsample, are pipelined.

The module includes one on-chip buffer to store the platform positions. The contents of this memory are constant throughout the execution of the algorithm. A second on-chip buffer is used to store the data points. At each time, all data points of a single pulse are stored in the buffer and reused. While a set of data points of a particular pulse is being used, the data points of the next pulse are loaded to the on-chip buffer. Therefore, this on-chip buffer is split into memories that work in a ping-pong fashion.

The pixel accumulation is also implemented with OCM. Likewise, while the output pixels associated with a set of data points are being generated, the previous output pixels are being transferred to external memory. So, this buffer also works in a ping-pong fashion.

The size of the operators is statically configured before synthesizing the circuit. The tiling factor that determines the degree of data reuse is also configurable. The larger the tiling block, the larger size of the on-chip memories to store the accumulations. The hardware datapath that implements the BP algorithm can be replicated to allow parallel calculation of output pixels. Algorithmically this corresponds to unrolling the X loop of Algorithm 3. The nested loops of the designed architecture accept continuous dataflow of data with an iteration interval of one. Fig. 3 illustrates the implementation of SAR_IP core with two parallel datapaths.

Increasing the number of parallel datapaths requires a proportional increase of the hardware resources and memory. The upsample and the position memories are shared by the parallel datapaths. Since each datapath needs dual port access to the data memory, this memory is replicated with the same contents for each datapath to increase the number of memory ports.

The execution time of the BP algorithm, can be theoretically estimated for a particular image size ix, iy, number of pulses p, tiling TL, upsample factor U, and number of parallel datapaths DP. Since the architecture is fully pipelined and neglecting the

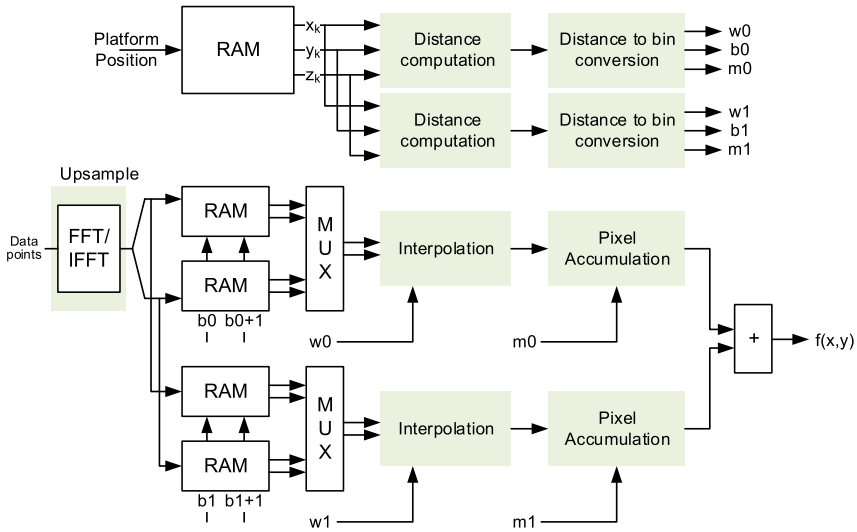

Fig. 3. Hardware design of the BP algorithm with two parallel datapaths.

initial pipeline filling and final pipeline emptying, the number of clock cycles $\text{BP}_{\text{cycles}}$ to execute the whole BP algorithm, including upsampling, is estimated as

$$\text{BP}_{\text{cycles}} = \frac{iy}{\text{TL}} \times \max\left(\frac{ix \times \text{TL} \times p}{\text{DP}}, \text{Upsample}(p, U)\right) \quad (7)$$

where $\text{Upsample}(p, U)$ is the number of cycles to execute the upsample of p pulses by the upsample factor U. The expression considers the number of cycles to run the BP algorithm and the cycles to run the upsample. The worst case determines the total latency of the circuit.

The number of cycles is, therefore, determined by the maximum number of cycles between upsampling and BP. The designer must determine the best hardware configuration for a particular image size, memory bandwidth, and operating frequency to obtain the most efficient solution, that is, a solution with the minimal idle times among all processing modules.

Another important aspect of the architecture is the required OCM. The total OCM in bytes of the proposed architecture is given by

$$\begin{aligned}\text{OCM} = {} & ix \times \text{size of (PlatPos)} \\ & + 2 \times U \times p \times \text{size of (dataIn)} \times \left\lceil \frac{\text{DP}}{2} \right\rceil \\ & + ix \times \text{TL} \times \text{DP} \times \text{size of (dataOut)}. \end{aligned} \quad (8)$$

The maximum tiling factor and the number of parallel datapaths are constrained by the resources of the target FPGA device.

V. Experimental Results

The SAR_IP core has been described in VHDL, integrated in an SoC architecture and tested on FPGA. The target device is an Artix-7 XC7A200 T FPGA in the Nexys Video trainer board. The hardware design and implementation have been done with Vivado Design Suite 2020.3 and the power of the circuits has been estimated with Vivado power estimator tool.

The Nexys video trainer board has 512 MB of DDR3 memory with a maximum measured 1.2 GB/s of memory bandwidth to the FPGA. The FPGA has 215 K logic cells with 269 200 registers, 134 600 LUTs, 365 BRAMs, and 740 DSPs.

All architectures were tested with synthetic images of sizes 512×512 with 512 pulses and 1024×1024 with 1024 pulses, and a real image from the AFRL dataset with a size of 512×512 and 512 pulses. The 1024×1024 output images are the same as those with half the size but with half the sampling. Any other sizes can be considered with a proper configuration of the architecture.

A. Analysis of the Error With Fixed-Point Design

The proposed methodology for floating-point to fixed-point conversion was utilized to convert the algorithm to a custom fixed-point design. Different SNR constraints were considered, namely, 40 db, 60 db, 80 db, 100 db, 120 db, and the highest SNR. In all cases, the fixed-point representation for all variables was determined (see results for images of size 512×512 in Table I).

From the table, it is possible to observe the number of bits considered for the integer part and the range of fractional bits for each variable of Algorithm 1. The square root is one of the most critical with a low range of variability of the fractional part. The SNR with a double-precision floating-point is 138.87 dB. The highest SNR achieved with fixed precision was 137.65 dB, close to the floating-point implementation. The upsample operations were also implemented with fixed-point with the variable range identified in the table as $g(\text{data})$.

The fixed-point representations are similar for other image sizes, with a small increase in the number of bits of the integer part, namely in the R, b, and acc variables with from 1 to 2 bits more.

The last column shows the results for double-precision floating-point arithmetic. The difference in accuracy is negligible.

B. Analysis of the Circuit Area

Algorithm 3 was designed and implemented in the FPGA as an accelerator in the form of an IP core (SAR_IP). The full

TABLE I
OUTPUT OF THE FLOAT TO FIXED-POINT TOOL FOR DIFFERENT SNRS

Variable	Int. bits	Frac. bits of each configuration						Double FP
g (data)	1	15	18	21	29	31	31	64
$(x - xk)$	13	11	13	15	16	21	21	64
$(y - yk)$	13	11	13	15	16	21	21	64
$(z - zk)$	13	11	13	15	16	21	21	64
$(x - xk)^2$	26	16	20	21	21	21	21	64
$(y - yk)^2$	26	16	20	21	21	21	21	64
$(z - zk)^2$	26	16	20	21	21	21	21	64
SQRT	26	18	18	18	18	21	21	64
R	14	18	24	30	30	34	34	64
b	12	6	16	16	22	22	22	64
w	0	6	6	16	22	22	22	64
$g(x,y)$	1	12	15	15	25	28	29	64
sin/cos	2	9	12	15	18	21	24	64
mf	4	10	15	15	26	26	26	64
prod	2	12	16	16	28	30	30	64
acc	10	10	21	21	24	28	28	64
f	10	16	16	16	16	16	16	64
Metrics								
SNR (dB)		40.11	60.14	80.17	100.6	121.29	137.65	138.87
MSE		117.71	1.04	0.01	1×10^{-4}	7.7×10^{-6}	2.2×10^{-7}	2.1×10^{-7}
PSNR (dB)		138.86	159.4	179.18	199.00	220.69	222.87	222.89

TABLE II
CHARACTERIZATION OF THE PROPOSED SAR_IP FOR IMAGES OF SIZE 512×512 CONSIDERING THE ALGORITHM WITH DOUBLE FLOATING-POINT ARITHMETIC

Double FP, 512×512**, SNR = 138.87 dB**			
	LUTs	**DSPs**	**BRAMs**
Upsample	9737	68	44
BP	15 713	63	56
Total	25 450	131	100
Double FP, 1024×1024**, SNR = 138.99 dB**			
Upsample	11 232	88	78
BP	15 878	63	58
Total	26 945	151	134

The architecture has a single datapath and a tiling of 8.

TABLE III
CHARACTERIZATION OF THE PROPOSED SAR_IP FOR IMAGES OF SIZE 512×512 WITH SIX DIFFERENT SNRS (ARQ. 1: 40.11 DB, ARQ. 2: 60.14 DB, ARQ. 3: 80.17, ARQ. 4: 100.6, ARQ. 5: 121.29, AND ARQ. 6: 137.65)

Arq. 1 - SNR = 40.11 dB			
	LUTs	**DSPs**	**BRAMs**
Upsample	6830	36	13
BP	5703	33	22
Total	12533	69	35
Arq. 2 - SNR = 60.14 dB			
Upsample	7385	54	18
BP	6165	33	30
Total	13550	87	48
Arq. 3 - SNR = 80.17 dB			
Upsample	7800	54	18
BP	6289	33	32
Total	14089	87	50
Arq. 4 - SNR = 100.6 dB			
Upsample	8472	54	18
BP	6677	34	33
Total	15149	88	51
Arq. 5 - SNR = 121.29 dB			
Upsample	9042	54	18
BP	6809	50	37
Total	15851	104	55
Arq. 6 - SNR = 137.65 dB			
Upsample	9514	54	18
BP	8114	54	38
Total	17628	108	58

The architectures have a single datapath and a tiling of 8.

precision algorithm was also implemented to obtain a reference to which the fixed-point implementations can be compared with (see Table II).

The accelerator is configurable and, therefore, can be designed with custom data-point representations for all variables. Different architectures (Arq.) were generated and implemented considering the fixed-point representations reported in Table I for different SNR (see the occupation of FPGA resources in Table III for an image of size 512×512).

The best architecture has an SNR close to the SNR achieved with double floating-point. Also, the architecture with lowest SNR is about 30% lower than the architecture with best SNR.

Considering images of size 1024×1024, the upsample module was configured for 1024 points (see results in Table IV).

The architecture for the larger images mainly differs in the upsample module since the datapath only differs in the final accumulators. The architecture with the highest SNR needs more resources, as expected. Choosing the right architecture will depend on the acceptable SNR and the resources of the target device.

Since the proposed architecture is configurable in terms of parallelism and tiling factor, we have analyzed the occupied resources of the SAR-IP for different configurations of tiling and the number of datapaths (see Tables V and VI).

The core was configured with up to eight parallel datapaths. The most balanced solution depends on the correct computation balance between the upsample and the BP hardware module. When the number of datapath units increases, the execution of the BP algorithm reduces proportionally. Since the upsample always needs the same time to compute, the tiling has to increase proportionally to the number of datapaths to balance the execution of both upsample and BP. Assuming that both modules work at the same frequency, a balanced solution is achieved when both parallelism and tiling increase proportionally.

TABLE IV
CHARACTERIZATION OF THE PROPOSED SAR_IP FOR IMAGES OF SIZE 1024×1024 WITH SIX DIFFERENT SNRS (ARQ. 1: 40.32 DB, ARQ. 2: 60.56 DB, ARQ. 3: 80.05, ARQ. 4: 100.39, ARQ. 5: 120.63, AND ARQ. 6: 136.78)

Arq. 1 - SNR = 40.32 dB			
	LUTs	**DSPs**	**BRAMs**
Upsample	7629	40	21
BP	6265	33	22
Total	13445	73	43
Arq. 2 - SNR = 60.56 dB			
Upsample	8155	60	32
BP	6292	33	30
Total	14447	93	62
Arq. 3 - SNR = 80.05 dB			
Upsample	8705	60	32
BP	6412	33	33
Total	15117	93	65
Arq. 4 - SNR = 100.39 dB			
Upsample	9238	60	32
BP	6811	34	34
Total	15149	94	66
Arq. 5 - SNR = 120.63 dB			
Upsample	9812	60	32
BP	6937	50	38
Total	15851	110	70
Arq. 6 - SNR = 136.78 dB			
Upsample	10623	60	32
BP	8289	54	40
Total	17628	114	72

The architectures have a single datapath and a tiling of 8.

TABLE V
CHARACTERIZATION OF THE PROPOSED SAR_IP ARCHITECTURE WITH DIFFERENT TILING FACTORS TL AND DIFFERENT NUMBER OF PARALLEL DATAPATHS (PARALLELISM: DP) FOR AN ARCHITECTURE WITH SNR = 100.6 DB AND IMAGES OF SIZE 512×512

DP = 1				
TL	8	16	32	64
LUTs	15149	15169	15175	15185
DSPs	88	88	88	88
BRAMs	51	68	100	164
DP = 2				
LUTs	21858	21878	21884	21894
DSPs	122	122	122	122
BRAMs	58	74	106	170
DP = 4				
LUTs	35276	35296	35302	35312
DSPs	190	190	190	190
BRAMs	70	86	118	182
DP = 8				
LUTs	62112	62132	62138	62148
DSPs	326	326	326	326
BRAMs	94	110	142	206

C. Performance Analysis of the Hardware/Software Architecture

The architectures with SNR $\approx$ 100 dB were considered for testing in the FPGA with images of size 512×512 and 1024×1024. Results can be obtained for other configurations of the architecture with different SNR and other image sizes by configuring the SAR_IP and resynthesizing the system.

The core was integrated in a SoC with an embedded RISC-V CPU (see Fig. 4).

TABLE VI
CHARACTERIZATION OF THE PROPOSED SAR_IP ARCHITECTURE WITH DIFFERENT TILING FACTORS, TL, AND DIFFERENT NUMBER OF PARALLEL DATAPATHS (PARALLELISM - DP) FOR AN ARCHITECTURE WITH SNR = 100.39 DB AND IMAGES OF SIZE 1024×1024

DP = 1				
TL	8	16	32	64
LUTs	15915	15935	15941	15951
DSPs	94	94	94	94
BRAMs	66	82	114	178
DP = 2				
LUTs	22624	22640	22650	22660
DSPs	128	128	128	128
BRAMs	72	90	124	192
DP = 4				
LUTs	36042	36062	36068	36078
DSPs	196	196	196	196
BRAMs	84	102	136	204
DP = 8				
LUTs	62878	62898	62904	62914
DSPs	332	332	332	332
BRAMs	108	126	160	228

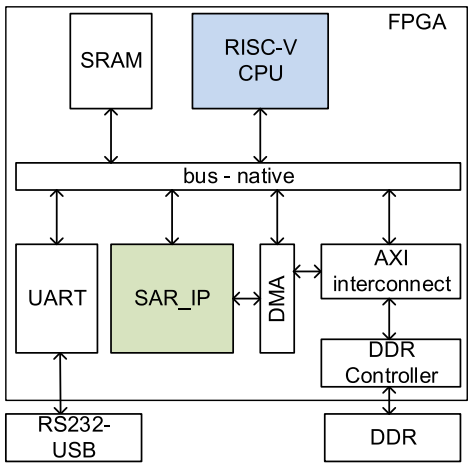

Fig. 4. SoC architecture with an RISC-V CPU designed to integrate and test the SAR_IP core.

The system uses a low-performance RISC-V soft processor to control the memory subsystem and peripherals, including the SAR_IP core. The set of peripherals includes internal memory to store the firmware, and external memory to store the input and output images and one UART. The data communication between the external memory and the core is done with a direct memory access (DMA) through the DDR memory controller. The DMA is configured by the CPU and allows direct access between the core and the memory. The area of individual modules and the full hardware/software architecture area for a configuration with DP=8 and tiling = 8 is given in Table VII).

The most used resources for the configuration with DP = 8 are BRAMs and LUTs. These determine the highest parallel factor. Other DP factors can be considered above 8 to increase the utilization of the available resources and improve the performance. However, DP factors, not powers of two, are still not supported by the configurable core, so they were not considered in this study. This is left for a future upgrade of the core.

TABLE VII
CHARACTERIZATION OF THE PROPOSED SOC ARCHITECTURE WITH THE SAR_IP CORE INTEGRATED WITH THE RISC-V CPU

	LUTs	DSPs	BRAMs
RISC-V	2569	4	0
AXI-Interconnect	3542	0	0
UART	86	0	0
SRAM	60	0	18
DMA	2006	0	9
DDR controller	9697	3	0
Others	481	0	0
SAR_IP	62148	326	206
Total	80589	333	233
Total (%)	60	45	64

The core was designed with a tiling of 64 a with 8 parallel datapaths.

TABLE VIII
EXECUTION TIME OF THE ARCHITECTURE FOR DIFFERENT VALUES OF TILING TL AND DATAPATHS DP IMAGES OF SIZE 512×512 AND 1024×1024 FOR AN SNR $\approx$ 100 DB

	Image size = 512×512			
(TL, DP)	(8,1)	(16,2)	(32,4)	(64,8)
Exec. (s)	1.08	0.54	0.27	0.14
	Image size = 1024×1024			
(TL, DP)	(8,1)	(16,2)	(32,4)	(64,8)
Exec. (s)	8.68	4.34	2.18	1.11

To test the architecture, the DDR memory available in the board was utilized to store the input data. The full SoC system was implemented in the FPGA with a frequency of 125 MHz considering different configurations of the SAR_IP core. From these, the execution times were determined (see Table VIII).

The SAR_IP runs the BP algorithm for images of size 512×512 in a Artix-7 FPGA in 0.14 s. Real-time processing of images with size 512×512 is achieved with all configurations, according to the SAR image capture times referred in [20]. According to Schleuniger *et al.* [20], real-time processing is achieved with 60 000 pixels/s. For images of size 1024×1024, the core takes 1.11 s to run the BP algorithm using eight parallel datapaths.

From Table VIII, it is possible to verify the scalability of the architecture, since the execution time reduces proportionally to the increase of the number of datapaths.

D. Comparison of SAR_IP With Other Computing Devices

The BP algorithm was executed on a PC desktop with a quad-core Intel Core i7-4700MQ processor with 16 GB of RAM on a NVIDIA GeForce RTX 2060 GPU and on the Jetson TX2 module with a 256-core NVIDIA Pascal GPU. The algorithm was run for both image sizes (see results in Table IX).

The fastest platform is the GPU and the slowest is the CPU. However, these are not appropriate for embedded computing since the power is too high. Compared to the embedded GPU, the proposed FPGA architecture is $2.6\times$ faster for images of size $= 512 \times 512$ and $2.5\times$ faster for images of size 1024×1024. Also, the FPGA has an energy consumption $13\times$ smaller than the embedded GPU. This is an important advantage for embedded computing.

TABLE IX
EXECUTION TIME OF THE BP ALGORITHM ON FOUR DIFFERENT PLATFORMS: QUAD-CORE INTEL CORE I7-4700MQ PROCESSOR, NVIDIA GEFORCE RTX 2060 GPU, JETSON TX2, AND ARTIX-7 FPGA, CONSIDERING IMAGES OF SIZE 512×512 AND 1024×1024

Image size = 512×512				
Device	CPU	GPU	eGPU	FPGA
Time (s)	12.9	0.031	0.36	0.14
Power (W)	145	160	15	(2.11+0.81)*
Energy	1871	4.96	5.4	0.41
Image size = 1024×1024				
Time (s)	108.8	0.256	2.75	1.11
Power (W)	145	160	15	(2.11+0.81)*
Energy (J)	15,780	41	41	3.24

*core power + DDR3 power [25].

E. Comparison of SAR_IP With Other FPGA-Based Platforms

The execution time and the energy to process an image of the proposed architecture were also compared with previous FPGA works. Since different FPGA devices were used in previous works, efficiency metrics were also considered for a fair comparison. The efficiency metrics consider the number of pixels/pulses (PP) per LUTs, DSPs, and BRAMs that are processed per second. For example, as given in Table IX, the proposed solution runs $512 \times 512 \times 512$ pp, in 0.14 s with 80 589 LUTs. In this case, the LUT efficiency equals $512 \times 512 \times 512/0.14/80589 = 11,896$ PP/s/LUT. Another metric was considered to determine the energy efficiency (kPP/Energy), that is, the number of processed BP pixels per joule. The comparative results with previous implementations on FPGA are given in Table X.

All previous works consider large FPGAs with high consumption for an embedded system, except the work from Duarte *et al.* [24] that was implemented in a ZYNQ7020. However, it runs partially in the embedded processor, so it takes over 1 min to run the algorithm.

Considering the area efficiency, the proposed solution is the most efficient in terms of LUTs ($1.8\times$ better) and BRAMs ($1.5\times$ better). Since multiplications can be implemented with DSPs or LUTs, a design with a lower ratio of DSPs has a higher ratio of LUTs and vice versa. This means that the proposed solution achieves the highest pixel processing per second per area. Since the architecture is scalable, the ratio closely applies to different picture sizes. The fully optimized pipelined architecture also achieves the best performance efficiency (PP/Time/kLUT) and the best energy efficiency (kPP/Energy) by almost $2\times$ and $4\times$, respectively, compared to the best previous architecture.

The major improvements in performance and energy achieved with the proposed architecture are mainly from the fine customization of the fixed-point representation and a fully pipelined and scalable architecture with configurable parallelism.

The proposed methodology for floating- to fixed-point conversion of the algorithm helped to obtained an architecture with better performance and energy efficiency. The methodology can be applied to other algorithms as an initial design step before hardware design. to improve the final architecture.

The proposed architecture is also scalable, which allows its custom design for FPGAs with different hardware densities and for images with different dimensions. It can also be designed with different levels of parallelism and different external

TABLE X
COMPARISON BETWEEN THE PROPOSED ARCHITECTURE WITH PREVIOUS SOLUTIONS IMPLEMENTED IN FPGA

Work	[20]	[21]	[22]	[23]	[24]	Ours
Device	Virtex-7	Virtex-6	Stratix-V	Arria-V SoC	ZYNQ7020	Artix-7
Image Size	$60k \times 688$	$1.4k \times 1.5k \times 64$	$500 \times 500 \times 1k$	$1k \times 512 \times 512$	$512 \times 512 \times 512$	$512 \times 512 \times 512$
BP_{points}	41 280k	134 400k	250 000k	262 144k	134 218k	134 218k
LUTs	167 000	58 000	495 040	—	11517	80589
BRAMs	600	138	1003	—	4	233
DSPs	240	154	78	—	141	333
Time (s)	0.905	0.35	0.385	0.12	61.46	0.14
Power (W)	10	—	75	26.6	1.6	2.92
Energy (J)	9.05	—	28.88	3.19	98.34	0.41
PP/s/LUT	273	6,612	1,312	—	190	11 896
PP/s/DSP (10^6)	0.2	2.5	8.3	—	0.02	2.88
PP/s/BRAM (10^6)	0.08	2.78	0.65	—	0.55	4.1
kPP/Energy (J)	4.56	—	8.66	82.18	1.37	327.36

PP: Pixel/pulses.

memory bandwidths so that the right balance between communication and computation can be obtained. It also allows the fine-grain customization of fixed-point formats so that solutions with different tradeoffs between performance and accuracy can be generated with high efficiency.

In terms of energy consumption and power, the proposed accelerator has shown high performance with the highest efficiencies compared to other embedded processing solutions and FPGAs. This is particularly relevant for onboard processing of algorithms with high complexity with low energy.

Another relevant aspect of the proposed solution is the integration of both upsampling and SAR image formation in a single custom hardware architecture. This improves the quality of the generated image while optimizing the performance and energy efficiency. Previous works do not consider this design aspect.

The work described in this article has, therefore, contributed to the advance of high-performance onboard processing at low cost and energy.

VI. CONCLUSION

Onboard processing systems have recently emerged in order to overcome the huge amount of data to transfer from the satellite to the ground station. SAR imaging is a remote sensing technology that can benefit of onboard processing. This article proposes an FPGA-based hardware core for onboard processing of SAR images.

The original algorithm was reorganized to improve the accesses to data stored in external memory. The proposed architecture has been designed in a development board with an Artix-7 FPGA. Experimental results indicate that the proposed SAR_IP core can fulfill real-time requirements with low resources in a low-cost FPGA. The architecture is more energy efficient than other computing platforms and previous FPGA implementations.

The proposed core is configurable and, therefore, can be configured for different performance, area, and power requirements.

REFERENCES

[1] J. C. Trinder, "Editorial for special issue 'applications of synthetic aperture radar (SAR) for land cover analysis'," *Remote Sens.*, vol. 12, no. 15, 2020, Art. no. 2428. [Online]. Available: https://www.mdpi.com/2072-4292/12/15/2428

[2] L. A. Gorham and L. J. Moore, "SAR image formation toolbox for MATLAB," in *Proc. Algorithms Synthetic Aperture Radar Imagery XVII*, 2010, pp. 769906–769913. [Online]. Available: https://doi.org/10.1117/12.855375

[3] *Synthetic Aperture Radar (SAR) Imaging Basics*. Hoboken, NJ, USA: Wiley, 2011, ch. 1, pp. 1–22. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118116104.ch1

[4] Y. K. Chan and V. Koo, "An introduction to synthetic aperture radar (SAR)," *Prog. Electromagn. Res. B*, vol. 2, pp. 27–60, 2008.

[5] C. H. Casteel Jr, L. A. Gorham, M. J. Minardi, S. M. Scarborough, K. D. Naidu, and U. K. Majumder, "A challenge problem for 2D/3D imaging of targets from a volumetric data set in an urban environment," in *Algorithms for Synthetic Aperture Radar Imagery XIV*, E. G. Zelnio and F. D. Garber, Eds. Bellingham, WA, USA: SPIE, 2007, pp. 97–103. [Online]. Available: https://doi.org/10.1117/12.731457

[6] K. Naidu and L. Lin, "Data dome: Full k-space sampling data for high-frequency radar research," in *Algorithms for Synthetic Aperture Radar Imagery XI*, E. G. Zelnio and F. D. Garber, Eds. Bellingham, WA, USA: SPIE, 2004, pp. 200–207. [Online]. Available: https://doi.org/10.1117/12.548773

[7] S. M. Scarborough *et al.*, "A challenge problem for SAR-based GMTI in urban environments," in *Algorithms for Synthetic Aperture Radar Imagery XVI*, E. G. Zelnio and F. D. Garber, Eds. Bellingham, WA, USA: SPIE, 2009, pp. 143–152. [Online]. Available: https://doi.org/10.1117/12.823461

[8] K. Barker *et al.*, "PERFECT (Power efficiency revolution for embedded computing technologies) benchmark suite manual," Pacific Northwest Nat. Lab. Georgia Tech Res. Inst., Dec. 2013. [Online]. Available: https://hpc.pnnl.gov/projects/PERFECT/

[9] A. F. Yegulalp, "Fast backprojection algorithm for synthetic aperture radar," in *Proc. IEEE Radar Conf. Radar Next Millennium*, 1999, pp. 60–65.

[10] L. M. H. Ulander, H. Hellsten, and G. Stenstrom, "Synthetic-aperture radar processing using fast factorized back-projection," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 39, no. 3, pp. 760–776, Jul. 2003.

[11] L. Zhang, H.-L. Li, Z.-J. Qiao, and Z.-W. Xu, "A fast BP algorithm with wavenumber spectrum fusion for high-resolution spotlight SAR imaging," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 9, pp. 1460–1464, Sep. 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6725618/

[12] D. L. N. Hettiarachchi and E. Balster, "An accelerated SAR back projection algorithm using integer arithmetic," in *Proc. Asia-Pac. Signal Inf. Process. Assoc. Annu. Summit Conf.*, 2018, pp. 80–88.

[13] A. Fasih and T. Hartley, "GPU-accelerated synthetic aperture radar back-projection in CUDA," in *Proc. IEEE Radar Conf.*, 2010, pp. 1408–1413.

[14] O. Frey, C. L. Werner, and U. Wegmuller, "GPU-based parallelized time-domain back-projection processing for Agile SAR platforms," in *Proc. IEEE Geosci. Remote Sens. Symp.*, Jul. 2014, pp. 1132–1135. [Online]. Available: https://ieeexplore.ieee.org/document/6946629/

[15] C. Stringham and D. G. Long, "GPU processing for UAS-based LFM-CW stripmap SAR," *Photogramm. Eng. Remote Sens.*, vol. 80, no. 12, pp. 1107–1115, 2014.

[16] E. J. Balster, M. P. Hoffman, J. P. Skeans, and D. Fan, "GPGPU acceleration using OpenCL for a spotlight SAR simulator," in *Proc. 5th Int. Workshop OpenCL*, 2017. [Online]. Available: https://doi.org/10.1145/3078155.3078157

[17] B. Ge, L. Chen, D. An, and Z. Zhou, "GPU-based FFBP algorithm for high-resolution spotlight SAR imaging," in *Proc. IEEE Int. Conf. Signal Process. Commun. Comput.*, 2017, pp. 1–5.
[18] M. Wielage, F. Cholewa, C. Fahnemann, P. Pirsch, and H. Blume, "High performance and low power architectures: GPU vs. FPGA for fast factorized backprojection," in *Proc. 5th Int. Symp. Comput. Netw.*, Nov. 2017, pp. 351–357. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CANDAR.2017.101
[19] J. Park, P. T. P. Tang, M. Smelyanskiy, D. Kim, and T. Benson, "Efficient backprojection-based synthetic aperture radar computation with many-core processors," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.
[20] P. Schleuniger, A. Kusk, J. Dall, and S. Karlsson, "Synthetic aperture radar data processing on an FPGA multi-core system," in *Proc. 26th Int. Conf. Archit. Comput. Syst.*, 2013, pp. 74–85. [Online]. Available: https://arcs2013.fit.cvut.cz/
[21] F. Cholewa, M. Pfitzner, C. Fahnemann, P. Pirsch, and H. Blume, "Synthetic aperture radar with backprojection: A scalable, platform independent architecture for exhaustive FPGA resource utilization," in *Proc. Int. Radar Conf.*, Oct. 2014, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/7060455/
[22] F. David, "Implementation of a power efficient synthetic aperture radar back projection algorithm on FPGAs using OpenCL," Master's thesis, Dept. Elect. Eng., Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2017.
[23] D. Pritsker, "Efficient global back-projection on an FPGA," in *Proc. IEEE Radar Conf.*, May 2015, pp. 204–209.
[24] R. P. Duarte, H. Cruz, and H. Neto, "Reconfigurable accelerator for on-board SAR imaging using the backprojection algorithm," in *Proc. 16th Int. Symp. Appl. Reconfigurable Comput. Archit., Tools, Appl.*, 2020, pp. 392–401.
[25] Micron, "System power calculator," 2021. [Online]. Available: https://www.micron.com/support/power-calc

David Mota received the B.S. and M.S. degrees in electronic and telecommunications engineering from Instituto Superior de Engenharia de Lisboa, Lisboa, Portugal, in 2017 and 2021, respectively.

His current interests include the areas of digital signal processing and reconfigurable computing.

Helena Cruz received the B.Sc. and M.Sc. degrees in computer science and engineering in 2016 and 2018, respectively, from Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal, where she is currently working toward the Ph.D. degree.

She is an early-stage Researcher with INESC-ID, Lisbon, where she is researching the optimization of synthetic-aperture radar (SAR) image formation algorithms, implementation of SAR algorithms using reconfigurable devices and fault tolerance mechanisms for SAR.

Pedro R. Miranda received the M.Sc. degree in electrical and computer engineering from Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal, in 2021.

He developed reconfigurable computing architectures for real-time SAR imagery at INESC-ID. His main interest is digital system design, with an emphasis on reconfigurable computing.

Rui Policarpo Duarte received the Ph.D. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 2014.

He is currently a Researcher with the Electronic Systems Design and Automation Research Group, INESC-ID, Lisbon, Portugal. His research interests include reconfigurable computing, and fault-tolerant and low-power architectures.

José T. de Sousa received Ph.D. degree in electronics engineering from Imperial College, London, U.K., in 1998.

He is currently a Lecturer with the Department of Electrical and Computer Engineering, School of Engineering (IST), University of Lisbon, Lisbon, Portugal, and has been a Senior Researcher with INESC-ID, Lisbon, a research institute associated with IST since 1999. He is also a Tech Entrepreneur in the area of semiconductor intellectual property, having founded and managed three companies: Coreworks (2001.2013, cofounder and CEO), IPbloq (2017.2019, cofounder and CEO), IObundle (2018-present, owner and founder). He holds four international patents, is a coauthor of one book, and has authored or coauthored more than 70 technical papers in international journals and conferences. His main research interests include digital systems design and computer architecture, with an emphasis on reconfigurable computing.

Dr. de Sousa was the General Chair of the 2013 Field Programmable Logic and Applications Conference, Co-Editor of its proceedings, and a related Special Issue on the IEEE TRANSACTIONS ON COMPUTERS.

Horácio C. Neto received the Ph.D. degree in electrical and computer engineering from the Technical University of Lisbon, Lisbon, Portugal, in 1992.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, School of Engineering (IST), University of Lisbon. He is responsible for the Electronic Systems Design and Automation Research Group, INESC-ID, Lisbon, a research institute associated with IST. His main research interests include digital systems design and computer architecture, with an emphasis on reconfigurable computing.

Mário P. Véstias received the Ph.D. degree in electrical and computer engineering from the Technical University of Lisbon, Lisbon, Portugal, in 2002.

He is a Coordinate Professor with the Department of Electronics, Telecommunications and Computer Engineering, School of Engineering, Polytechnic Institute of Lisbon, Lisbon, where he is responsible for undergraduate and graduate courses on computer architecture and digital systems design. He is also a Senior Researcher with the Electronic Systems Design and Automation Group, INESC-ID, Lisbon. His research interests include computer architectures and digital systems for embedded reconfigurable computing.