Reconfigurable Accelerator for On-Board SAR Imaging Using the Backprojection Algorithm

Rui Policarpo[1,2 [0000−0002−7060−4745]], Helena Cruz[1,3 [0000−0003−2709−867X]], and Horácio Neto[1,4 [000−0002−3621−8322]]

[1] INESC-ID/IST-UL, Lisboa
[2] rui.duarte@tecnico.ulisboa.pt
[3] helena.cruz@tecnico.ulisboa.pt
[4] hcn@inesc-id.pt

Abstract. Synthetic Aperture Radar is a form of radar widely used to extract information about the surface of the target. The transformation of the signals into an image is based on DSP algorithms that perform intensive but repetitive computation over the signal data. Traditionally, an aircraft or satellite acquires the radar data streams and sends it to be processed on a data center to produce images faster. However, there are novel applications demanding on-board signal processing to generate images. This paper presents a novel implementation for an on-board embedded SoC of an accelerator for the Backprojection algorithm, which is the reference algorithm for producing images of SAR sensors. The methodology used is based on a HW/SW design partition, where the most time consuming computations are implemented in hardware. The accelerator was specified in HLS, which allows to reuse the code from the original implementation of the algorithm in software. The accelerator performs the computations using floating-point arithmetic to produce the same output as the original algorithm. The target SoC device is a Zynq 7020 from Xilinx which has a dual-core ARM-A9 processor along with a reconfigurable fabric which is used to implement the hardware accelerator. The proposed systems outperformed the software-only implementation in 7.7× while preserving the quality of the image by adopting the same floating-point representations from the original software implementation.

Keywords: FPGA · Synthetic Aperture Radar · DSP · Backprojection · Zynq · SoC · Reconfigurable Accelerator.

1 Introduction and Motivation

Remote sensing technologies such as Synthetic-Aperture Radar (SAR) have been widely used monitor the surface of the Earth, in particular, ships and oil spills tracking at sea, ice-caps and sea level, terrain erosion, drought and landslides, deforestation, fires, and other types of natural disasters. The main strength of SAR is that it operates even in the presence of clouds, smoke and rain and without a light source, making it a very attractive method of monitoring Earth. A SAR sensor can be mounted on-board flying platforms such as satellites, aircrafts and

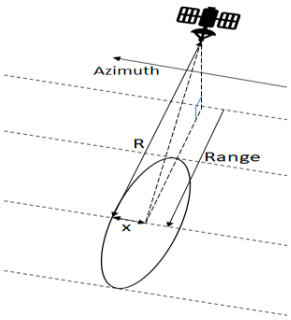

Fig. 1. Illustration of SAR operation and its physical parameters.

drones. Moreover, with the advancements in the technology and signal processing methods, there are increasing business opportunities for satellites and drones equipped with lightweight, small, and autonomous systems for on-board processing and generation of SAR images and subsequent broadcasting them, avoiding the time-consuming processing data at the receivers. However, its implementation in low-power embedded systems is limited to simplified implementations of the algorithm. While they are able to reduce the processing time, they sacrifice the image quality.

At the moment, the reference algorithm for SAR imaging is Backprojection (BP), which computes the contribution of each reflected pulse for each pixel on the resulting image. This process is time consuming as the projections all of the received pulses have to be computed for all pixels in the image. Figure 1 illustrates the parameters involved in the operation of a SAR mounted on a moving platform.

Recent radiation tests [4] on System-on-Chip (SoC) Zynq devices from Xilinx have shown that they provided a good performance under a harsh environment, therefore there is an increasing interest in adopting such systems on-board aircrafts. These devices have a dual-core ARM A9 CPU along with a reconfigurable fabric which is capable of implementing a hardware accelerator to alleviate the computations from the CPU and speedup the overall execution time.

This work introduces a novel accelerator architecture for SAR imaging using the Backprojection image generation algorithm and its evaluation on a SoC device.

This paper is organized as follows. Section 2 presents the background on BP algorithm and existing accelerators. Section 3 is dedicated to the profile of the algorithm, which determines which parts of the implementation require more processing time, and thus be the candidates for hardware acceleration. Section 4 details the implementation of the hardware accelerator using High-Level Synthesis (HLS). Section 5 presents the HW/SW system design, and its performance and resources are discussed in Section 6. Section 7 concludes the paper with the final remarks.

2 Background

2.1 Backprojection

The following nomenclature related to the Backprojection algorithm is adopted in this paper:

- R - Differential range from platform to each pixel at the center of the swath.
- 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.
- $f(x, y)$ - Value of each pixel (x, y).
- θ_k - Aperture point.
- r_k - Range from pixel $f(x, y)$ to aperture point θ_k.
- 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 equation 2).
- $s(n)$ - Wave sample in the previous adjacent range bin.
- $r(n)$ - Corresponding range to the previous adjacent bin.

As aforementioned, the BP algorithm computes the contribution of each reflected pulse for each pixel on the resulting image. The BP algorithm takes the following values as input: number of pulses, location of the platform for each pulse, the carrier wave number, the radial distance between the plane and target, the range bin resolution, the real distance between two pixels and the measured heights. For each pixel and each pulse, the BP algorithm, performs the following steps:

1. Computation of the distance from the platform to the pixel:

$$R = \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} - r_c \tag{1}$$

2. Conversion of the distance to an associated position (range) in the data set (received echoes).
3. Obtain the samples at the computed range via linear interpolation, using equation 2 [5].

$$g_{x,y}(r_k) = g(n) + \frac{s(n+1) - s(n)}{r(n+1) - r(n)} \cdot (r_k - r(n)) \tag{2}$$

4. Scales the sampled value by a matched filter to form the pixel contribution. This value is calculated using equation 3 [5]. dr is calculated using equation 1 [5].

$$e^{i\omega 2|\overrightarrow{r_k}|} = \cos(2 \cdot \omega \cdot dr) + i \sin(2 \cdot \omega \cdot dr) \tag{3}$$

5. Accumulates the contribution into the pixel. The final value of each pixel is given by equation 4 [5].

$$f(x, y) = \sum_k g_{x,y}(r_k, \theta_k) \cdot e^{i \cdot \omega \cdot 2 \cdot |\overrightarrow{r_k}|} \tag{4}$$

The pseudocode to compute the aforementioned steps is shown in algorithm 2. k_u represents the wave number and is given by $\frac{2\pi f_c}{c}$, where f_c is the carrier frequency of the waveform and c is the speed of light, a_k refers to the position of the pixel, and v_p, corresponds to the platform position. The complex exponential $e^{i\omega}$ is equivalent to $\cos(\omega)+i\sin(\omega)$ and, therefore, a cosine and sine computation is implied in the calculation of each pixel, represented in equation 4.

Algorithm 1 Backprojection algorithm pseudocode, from [1].

```
for all pixels k do
    f_k ← 0
    for all pulses p do
        R ← ||a_k − v_p||
        b ← ⌊(R − R0)/ΔR⌋
        w ← ⌊(R − R0)/ΔR⌋ − b
        s ← (1 − w) · g(p, b) + w · g(p, b + 1)
        f_k ← f_k + s · e^{i·k_u·R}
    end for
end for
```

2.2 FPGA Accelerators for Backprojection

There are several accelerators for the BP algorithm, however they often target High Performance Computing (HPC) systems for real-time generation of images. The work in [3] uses OpenCL to program 16 GPUs (with 2048 cores each), receives all signals in 17.7 seconds and takes 1 to produce an image. There are also some implementations of accelerators on FPGA of variations of the BP algorithm such as fast-BP [6] or factorized-BP [7]. Even though they perform faster than the complete BP algorithm the image quality is degraded, therefore they are not useful for comparison with the proposed architecture. Previous work on implementing the BP algorithm targeting SoC devices can be found in [2]. However, the authors focused on acceleration by distributing the load on the two CPU cores and introducing a lightweight software-only fault tolerance mechanism.

3 Algorithm Profiling

The profiling of the BP algorithm running on a single core of the A9 ARM processor of the target Zynq device was required to determine which parts of the algorithm should be accelerated.The implementation of the BP algorithm adopted is available in [1]. The obvious functions to be accelerated in hardware are the square root and the sine and cosine functions from the inner loop. Nevertheless, in this algorithm there is a final accumulation operation at the end of the inner loop, which can be seen as a reduce operation, and thus a scale down in the number of data transfers required.

In the profiling, an image of 512x512 pixels was generated from 512 pulses, with 512 samples for pulse. 512 complex floating-point samples produce a single complex floating-point result, which results in reduction of required throughput.

Table 1 summarizes the processing times of the most time consuming mathematical operations in the BP algorithm. All times are in nanoseconds and were measured for 1000 repetitions of the execution of each operation on the ARM processor, compiled with -O3 compiler optimization.

Table 1. Execution times for the operations in the implementation of BP.

Operation	Time [ns]	% Execution Time
Sqrt	50	1.3%
Sin+Cos	3108	84.3%
Misc	530	15.4%
Total	3688	100.0%

4 SAR Backprojection Accelerator

The accelerator targeted the most time consuming operations of the BP algorithm, and was specified using Xilinx HLS. Using HLS and maintaining the floating-point representation allows to reutilize parts of the source code and guarantees that the images produced will have the same result as the original implementation of the BP algorithm. The accelerator was implemented as a single IP core, where it receives the range values and samples for 512 pulses. The range values are double precision floating-point values whereas the samples are complex single-precision floating-point values. The operations implemented on the accelerator correspond to line 9 of algorithm 2.

In this specification, it is noteworthy the separation of the computations between two loops in the HLS specification. The first loop obtains the data for the range values from the streaming interface, computes their product to serve as input to trigonometric functions and stores the result in local memories. The second loop receives the pulse samples also via the streaming interface, performs the complex multiplication and writes the result to the output streaming interface. Figure 3 illustrates the sequence diagram of the relations between the building blocks of the accelerator.

Table 2 summarizes the FPGA resources required to implement the BP accelerator from the specification. The HLS tool produced a circuit design capable of operating at 100 MHz, resulting in an IP core which requires a minimum of 60 clock cycles in latency, of which 24 cycles are required by the CORDIC IP.

Table 2. Estimate of resources required to implement the BP accelerator reported by Vivado HLS.

Resource	Utilization	% Total on Zynq-7020
BRAM18K	2	1%
DSP48E	34	15%
LUTs	13986	26%

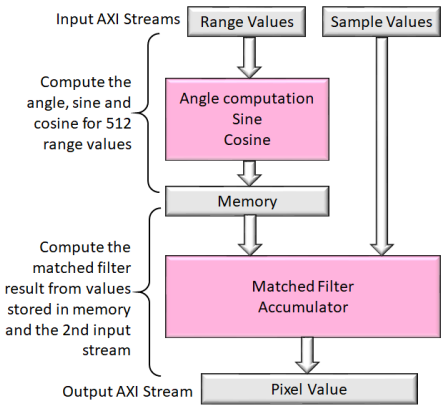

Fig. 2. Organization of the accelerator.

Algorithm 2 HLS accelerator specification.

for all pulses p **do**
  $input \leftarrow inStream.read()$
  $R \leftarrow input.data()$
  $angle \leftarrow 2.R.Ku$
  $s, c \leftarrow hls :: sincos(angle)$
  $mem\_sin[p] \leftarrow s$
  $mem\_cos[p] \leftarrow c$
end for
for all pulses p **do**
  $input \leftarrow inStream.read()$
  $sample.re \leftarrow input.data()$
  $sample.im \leftarrow input.data()$
  $matched\_filter\_result \leftarrow (mem\_cos[p] + imem\_sin[p]) \cdot sample$
  $acc \leftarrow acc + matched\_filter\_result$
  $outStream.write(acc)$ ▷ pixel_val
end for

5 HW/SW Project

The HW/SW project of the BP algorithm follows the partition created for the accelerator of the algorithm. The accelerator was integrated by establishing a connection to the CPU via AXI streaming interface, which is connected through Direct Memory Access (DMA) controller. Figure 3 illustrates the Vivado project containing the hardware blocks. On the software-side, the accelerator is used issuing data transfers between the DMA controller and the memory.

The listing 1.1 shows the simplified code running on the ARM A9 CPU. The initial part of the code corresponds to the initialization of constants [1]. The loops for all pixel computations were changed so that only the range computations are performed in software and the rest of the algorithm in the hardware accelerator. Moreover, the original loop which iterated all the pulse samples was removed as they are computed by the accelerator. The interaction with the accelerator happens through the DMA, before instructing to transfer input values of range and sample values from the DDR to the accelerator, is programmed to wait for the computation of a row of pixels.

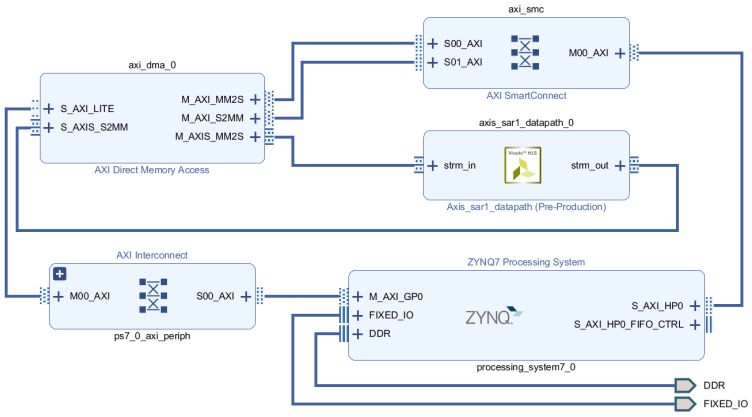

Fig. 3. Hardware project design on Vivado.

Listing 1.1. Backprojection code

```
void Backprojection() {
  sar_constants_calculation();
  for (iy = 0; iy < BP_NPIX_Y; ++iy)     {
    const double py = (-BP_NPIX_Y/2.0 + 0.5 + iy) * dxdy;
    DMA_Transfer(image + iy*row_offset); // ACCL 2 DDR image row
    for (ix = 0; ix < BP_NPIX_X; ++ix)           {
        // calculate pixel contribution
        DMA_Transfer(range); // DDR 2 ACCL
        DMA_Transfer(samples); // DDR 2 ACCL
    } // x
  } // y
} // func
```

6 Results and Discussion

The proposed system was implemented on a Zynq-7020 device installed on a Pynq-Z2 from TUL. The system was tested with two images, a synthetic one provided in the Perfect Suite[1] and a real one from the AFRL dataset, in figure 4. The software was compiled with the -O3 compilation option.

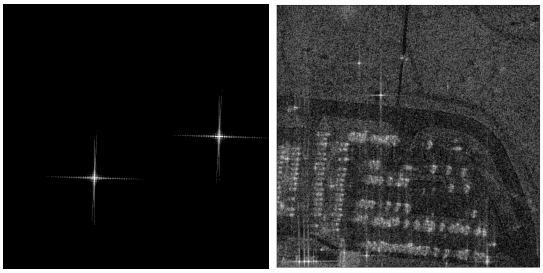

Fig. 4. Synthetic SAR image from the Perfect benchmark suite (left) and real SAR image from the AFRL dataset (right).

6.1 Processing Time

From the original algorithm profiling, it was found the algorithm required 487.5 seconds to generate of a 512x512px image. The processing times for the computations made by the accelerator in software, corresponding to line 9 of the pseudocode, required 1667.3 us, whereas the same computations in the accelerator required only 37.31 us, a reduction of 44.68×. Comparing the total processing times for a 512x512 image, between the original and the accelerated version, the accelerated is 7.7× faster.

6.2 Hardware Resources

The resources required to implement the accelerator on the reconfigurable fabric of the device are dominated by the Digital Signal Processing (DSP) blocks which consume about 64% of the total available on the device. Table 3 summarizes the resources required to implement the accelerator on the Zynq device.

6.3 Energy Consumption

The current consumption was measured using a UM24C USB power meter, connected between the host computer and the Pynq-Z2 FPGA board. Figure 5 shows the power consumption measured, which details the consumption for power-on, configuration of the device and execution of the algorithm with the reconfigurable

Table 3. Summary of resource utilization to implement the accelerator on a Zynq-7020.

Resource	Utilization	% Total on Zynq-7020
LUT	11517	21.65
BRAM	4	2.86
DSP	141	64.09

accelerator. The average power consumption of the whole system is 1.796 W. The power estimate from Vivado provides insight on the on-chip power consumption, which is 1.584 W. The difference between the measurement and the estimate is around 200 mW (12%) and is attributed to other components present on-board which are not taken into account by Vivado. Figure 6 shows the details of the power consumption, where 86% of power is consumed by the CPU (PS7).

The software-only implementation consumes on average 1.72 W. Even though the system with the hardware accelerator requires more 76 mW, finishes 7.1 minutes earlier than the software-only implementation. In comparison with the originalexecution on the CPU, which consumed 241.5 mWh (772.2 J), the system with the hardware accelerator requires 30.4 mWh (109,55 J), which represents 14.18% of the total energy consumption.

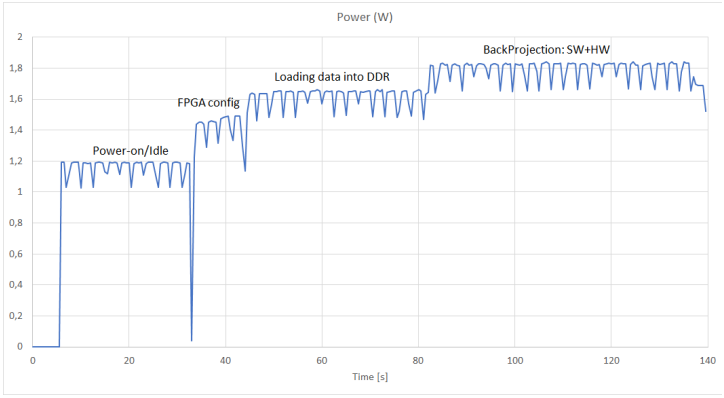

Fig. 5. System current consumption during the different stages of the experiment.

7 Conclusions

The work presented proposes a novel HW/SW implementation of the BP algorithm on an embedded SoC platform for on-board processing of SAR imaging. The creation of the accelerator was facilitated by the adoption of HLS to migrate sets of arithmetic operations from software to hardware. The proposed

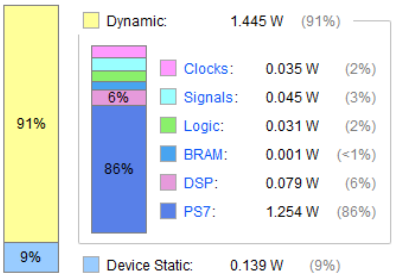

Fig. 6. On-chip power consumption distributed across the different elements.

architecture was able to achieve a speedup of 7.7× over the software-only implementation while preserving the quality of the image. Future work will focus on moving other operation of the BP algorithm into hardware to further improve the performance of the accelerator.

Funding Sources

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2019 and PTDC/EEI-HAC/31819/2017 (SARRROCA).

References

1. Barker, K., Benson, T., Campbell, D., Ediger, D., Gioiosa, R., Hoisie, A., Kerbyson, D., Manzano, J., Marquez, A., Song, L., Tallent, N., Tumeo, A.: PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual. Pacific Northwest National Laboratory and Georgia Tech Research Institute (December 2013), `http://hpc.pnnl.gov/projects/PERFECT/`
2. Cruz, H., Duarte, R.P., Neto, H.: Fault-tolerant architecture for on-board dual-core synthetic-aperture radar imaging. In: Hochberger, C., Nelson, B., Koch, A., Woods, R., Diniz, P. (eds.) Applied Reconfigurable Computing. pp. 3–16. Springer International Publishing, Cham (2019)
3. Gocho, M., Oishi, N., Ozaki, A.: Distributed Parallel Backprojection for Real-Time Stripmap SAR Imaging on GPU Clusters. Proceedings - IEEE International Conference on Cluster Computing, ICCC **2017-Septe**, 619–620 (2017). https://doi.org/10.1109/CLUSTER.2017.64
4. Lentaris, G., Maragos, K., Stratakos, I., Papadopoulos, L., Papanikolaou, O., Soudris, D., Lourakis, M., Zabulis, X., Gonzalez-Arjona, D., Furano, G.: High-performance embedded computing in space: Evaluation of platforms for vision-based navigation. Journal of Aerospace Information Systems **15**(4), 178–192 (2018). https://doi.org/10.2514/1.I010555
5. Pritsker, D.: Efficient global back-projection on an fpga. In: 2015 IEEE Radar Conference (RadarCon). pp. 0204–0209 (May 2015). https://doi.org/10.1109/RADAR.2015.7130996

6. Song, X., Yu, W.: Processing video-SAR data with the fast backprojection method. IEEE Transactions on Aerospace and Electronic Systems **52**(6), 2838–2848 (2016). https://doi.org/10.1109/TAES.2016.150581
7. Wielage, M., Cholewa, F., Riggers, C., Pirsch, P., Blume, H.: Parallelization strategies for fast factorized backprojection SAR on embedded multi-core architectures. In: 2017 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS). pp. 1–6. IEEE (nov 2017). https://doi.org/10.1109/COMCAS.2017.8244770