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

Removed for blind review

No Institute Given

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 presence clouds, smoke and rain and without a light source, hence, making it a very attractive method to generate images of Earth's surface. A SAR sensor can be mounted on-board flying platforms such as satellites, aircrafts and drones. Moreover, with the advancements in the technology and signal processing methods, there are increasing business opportunities for satellites, drones and Unmanned Aerial Vehicles (UAVs) equipped

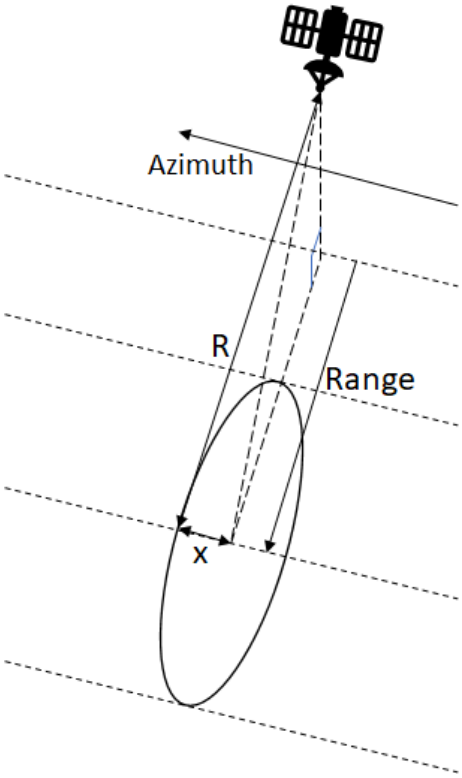

Fig. 1. Illustration of SAR operation and its physical parameters.

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, to the SAR sensor, 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 the pixels in the image. Figure 1 illustrates the parameters involved in the operation of a SAR mounted on a moving platform, and also in the BP algorithm, namely R which is the range of the swath for each pulse, and x, which is the distance to the terrain covered by the pulse, for different azimuth positions.

Recent radiation tests [5] 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 satellites and 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 [6].

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

4. Scales the sampled value by a matched filter to form the pixel contribution. This value is calculated using equation 3 [6]. dr is calculated using equation 1 [6].

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

5. Accumulates the contribution into the pixel. The final value of each pixel is given by equation 4 [6].

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

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 [2].

```
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 [4] uses OpenCL to program 16 GPUs (with 2048 cores each), receives all signals in 17.7 seconds and takes 1 second to produce an image.

There are also some implementations of accelerators on FPGA do variations of the BP algorithm such as: fast-BP [7], factorized-BP [8] or even an incomplete implementation [9]. 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 [3]. However, the authors focused on acceleration by distributing the load on the two cpu cores and introducing a light-weight software-only fault-tolerance mechanism, and don't benefit from any accelerator on the reconfigurable fabric.

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 given that not all parts of a Digital Signal Processing (DSP) algorithm consume the same time to process and the device has limited resources. The implementation of the BP algorithm adopted was the one available in [2], and presented in the listing 1.1.

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;
    for (ix = 0; ix < BP_NPIX_X; ++ix)          {
      complex accum;
      const double px = (-BP_NPIX_X/2.0 + 0.5 + ix) * dxdy;
      accum.re = accum.im = 0.0f;
      for (p = 0; p < N_PULSES; ++p)  {
        xdiff = platpos[p].x-px;
        ydiff = platpos[p].y-py;
        zdiff = platpos[p].z-z0;
        sqrt_aux = xdiff*xdiff+ydiff*ydiff+zdiff*zdiff;
        R = sqrt(sqrt_aux);
        const double bin = (R - R0)*dR_inv;
        complex sample, prod, matched_filter;
        const int bin_f = (int)bin;
        const float w = (float)(bin-(double)bin_f);
        sample.re=(1.0f-w)*data[p][bin_f].re+w*data[p][bin_f+1].re;
        sample.im=(1.0f-w)*data[p][bin_f].im+w*data[p][bin_f+1].im;
        double angle_aux = 2.0 * ku * R;
        signal_template.re = cos(angle_aux);
        signal_template.im = sin(angle_aux);
        matched_filter_result = cmult(sample, signal_template);
        accum.re += matched_filter_result.re;
        accum.im += matched_filter_result.im;
      } // pulse
      image[iy][ix] = accum;
    } // x
  } // y
} // func
```

The obvious functions to be accelerated in hardware are the square root (sqrt) and the trigonometric (sin, cos) functions from the inner loop. The remaining functions in the implementation are sums and multiplications. 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 a set of 512 pulses, with 512 samples for each pulse. For 512 complex floating-point input samples it produces 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 2 presents the processing times for a set for the computation of a single pixel, a set of 512 pixels and complete image of 512x512 pixels. The processing time for a single pixel corresponds to the processing of 512 input pulses.

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

Table 1. Execution times for the mathematical operations in the implementation of the algorithm.

Number of Pixels	Processing Time
1	1.88 ms
512	936 ms
512x512	487,5 s

Table 2. Execution time for sets of pixels.

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. It also guarantees that the images produced will have the same result as 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 numbers whereas the samples are complex single-precision floating-point numbers. The operations implemented on the

accelerator correspond to the ones on line 9 of the algorithm's pseudocode : $f_k \leftarrow f_k + e^{i \cdot k_u \cdot R}$, see 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 R from the streaming interface, computes their product with $2\pi Ku$ to serve as input to sine and cosine operations and stores the results in local memories. The second loop receives the pulse samples also via the streaming interface, performs the complex multiplication with the result from sine and cosine 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.

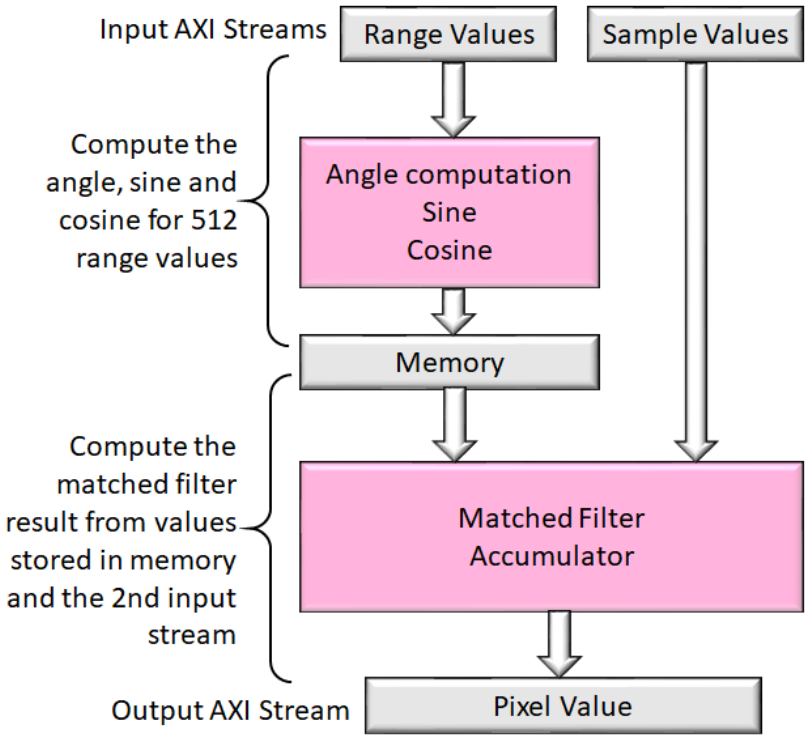

Fig. 2. Organization of the accelerator.

Table 3 summarizes the FPGA resources required to implement the BP accelerator from the HLS specification. The HLS tool was instructed to produce a circuit design capable of operating at 100 MHz. It produced an IP core which requires a minimum of 60 clock cycles in latency, of which 24 cycles are required by the CORDIC IP core instantiated by the HLS.

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

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

Table 3. Estimate of resources required to implement the BP accelerator reported by Vivado HLS.

5 HW/SW Project

The HW/SW project to implement the BP algorithm follows the partition created for the accelerator of the algorithm. The accelerator was integrated in the system by establishing a connection to the CPU via an AXI streaming interface, which is connected through a Direct Memory Access (DMA) controller. Figure 3 illustrates the Vivado project containing the hardware blocks, including the BP accelerator (axis_sar1_datapath_0) and the ARM processor (Processing System). On the software-side, the accelerator is used issuing data transfers between the DMA controller and the memory. The software running on the ARM processor is partially from [2].

The listing 1.2 shows the simplified C code running on the ARM A9 cpu. The initial part of the code corresponds to the initialization of constants as in the original code [2]. The loops for all pixel computations was 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, it is programmed to wait for the computation of a row of (512) pixels.

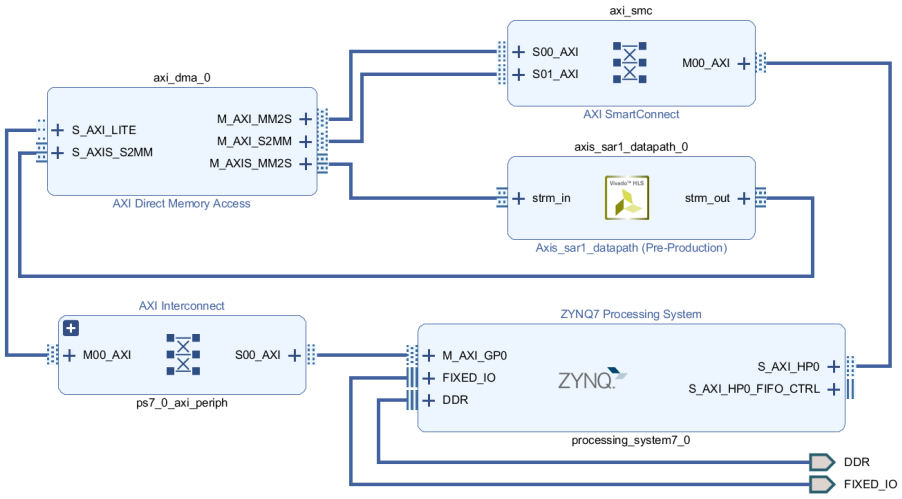

Fig. 3. Hardware project design on Vivado.

Listing 1.2. 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)           {
      complex accum;
      const double px = (-BP_NPIX_X/2.0 + 0.5 + ix) * dxdy;
      accum.re = accum.im = 0.0f;
      for (p = 0; p < N_PULSES; ++p)  {
        xdiff = platpos[p].x-px;
        ydiff = platpos[p].y-py;
        zdiff = platpos[p].z-z0;
        sqrt_aux = xdiff*xdiff+ydiff*ydiff+zdiff*zdiff;
        R = sqrt(sqrt_aux);
        const double bin = (R - R0)*dR_inv;
        complex sample, prod, matched_filter;
        const int bin_f = (int)bin;
        const float w = (float)(bin-(double)bin_f);
        DMA_Transfer(range); // DDR 2 ACCL
        DMA_Transfer(samples); // DDR 2 ACCL
      } // pulse
    } // x
  } // y
} // func
```

6 Results and Discussion

The proposed system was implemented on a Zynq-7020 device xc7z020iclg400-1l installed on a Pynq-Z2 from TUL. The system was tested with two images, a synthetic one provided in the Perfect Suite[2], shown in figure 4 on the left, and

a real one from the AFRL dataset[1], shown in the same figure on the right. Both images are 512x512 pixels. The software for the ARM CPU was compiled with the -O3 optimization compilation option.

6.1 Processing Time

From the original algorithm profiling, it was found that the algorithm required 487.5 seconds to complete the generation of a 512x512 pixels image. The processing times for the computations made by the accelerator in software, which corresponds to line 9 of the pseudocode, required 1667.3 us, whereas the same computations in the accelerator required only 37.31 us, which corresponds to a reduction of 44.68×. Table 4 summarizes the execution times to compute different loops of the algorithm, being the pulses the inner most loop and the Y pixels the outmost loop. An iteration on a loop requires the complete execution of its inner loops.

Loop	Number of Pixels	Processing Time
512 pulses	1	234,5 us
X pixels	512	120.05 ms
Y pixels	512x512	61,46 s

Table 4. Processing time of the proposed HW/SW architecture for different pixel quantities.

Comparing the total processing times for an 512x512 image, between the original software-only implementation and the one with the accelerator, the one with the accelerator is 7.7× faster than the original one.

Amdhal's Law [1] states the theoretical speedup (S) of a program is given that part (p) of it can be accelerated of an amount (s), equation 5.

$$S = \frac{1}{(1-p) + \frac{p}{s}} \tag{5}$$

The accelerated part of algorithm (p) corresponds to 89% of the original execution time, and it is accelerated 44.68 times, hence achieves a theoretical speedup of 7.69, which is validated by the speedup result of 7.75 obtained from the experiment.

6.2 Hardware Resources

The resources required to implement the accelerator on the reconfigurable fabric of the device are dominated by the DSP blocks which consume about 64% of the total available on the device. Table 5 summarizes the resources required to implement the accelerator on the reconfigurable fabric of the Zynq device.

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

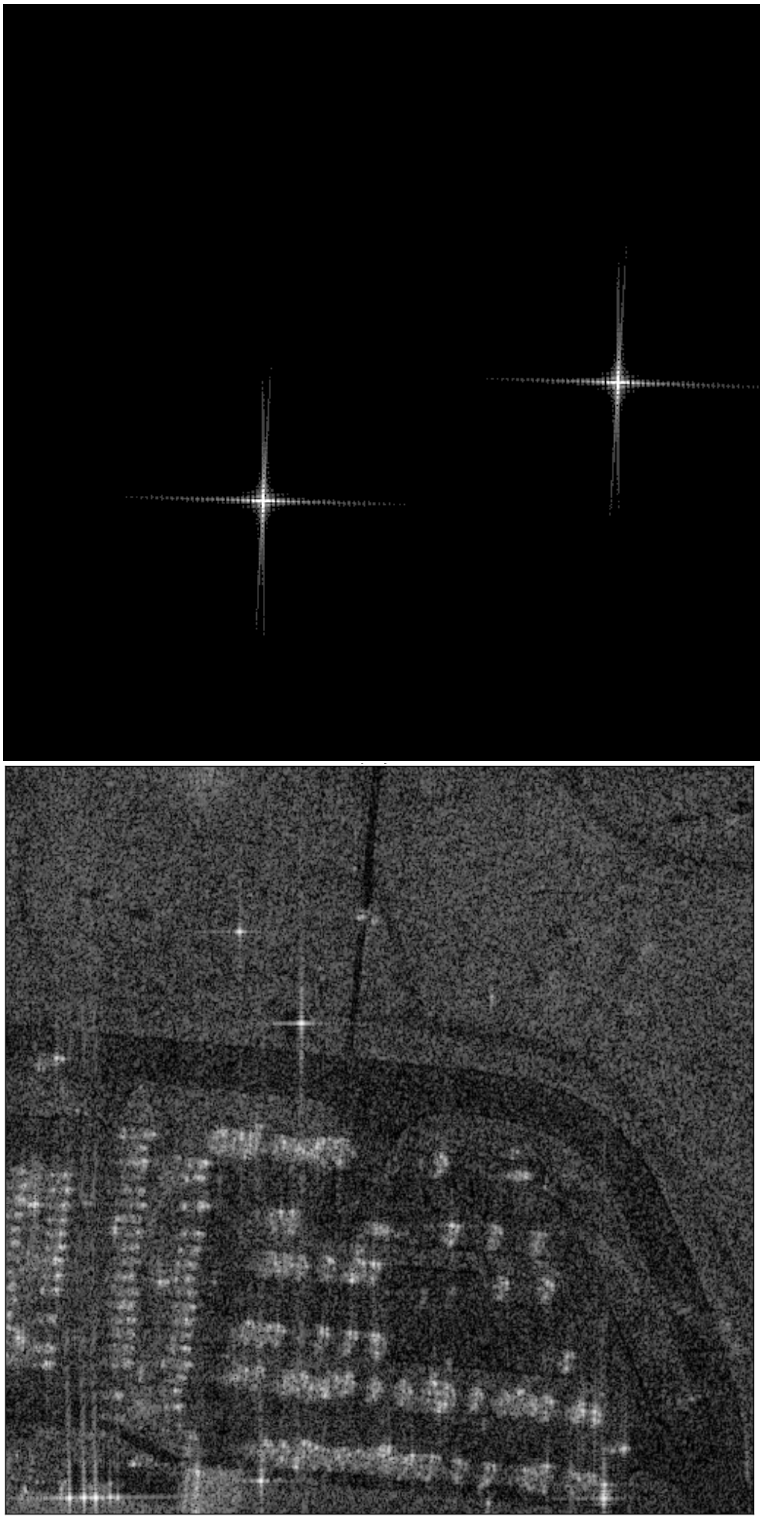

Fig. 4. Synthetic SAR image from the Perfect benchmark suite (top) and real SAR image from the AFRL dataset (bottom).

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

Table 5. Summary of resource utilization to implement the accelerator on a Zynq-7020 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, during the experiments. 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 accelerator. The average power consumption of the whole system during the computation of the BP algorithm 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, such as memories, ethernet controller and LEDs, which are not taken into account Vivado. Figure 6 shows the details of the power consumption, where 86% of the total on-chip power is consumed by the CPU (PS7). The DSP blocks are the elements on the reconfigurable fabric that consume the most as they are the most utilized during the execution of the BP algorithm.

The software-only implementation consumes on average 1.72 W. Even though the system with the hardware accelerator requires more 76 mW of instant power, it finishes 7.1 minutes earlier than the software-only implementation. In comparison with the original software-only execution 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 by the system. For a system supported by batteries as the proposed one, for the same battery charge the proposed system is capable of processing almost 8× more images than the traditional implementation.

7 Conclusions

The work presented in this paper 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 into hardware. The proposed 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.

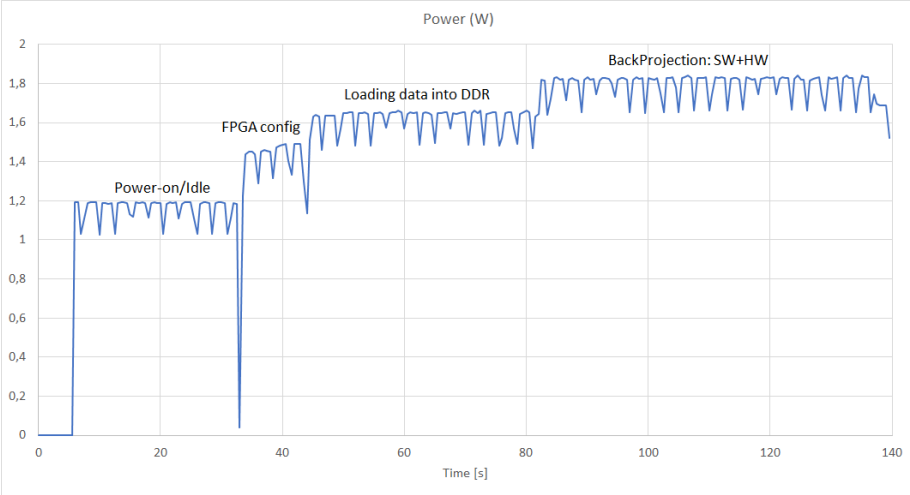

Fig. 5. System current consumption during the different stages of the experiment.

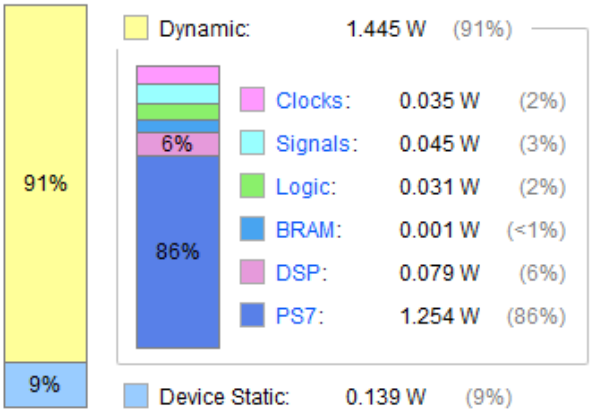

Fig. 6. On-chip power consumption distributed across the different elements.

References

1. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference. pp. 483–485. AFIPS '67 (Spring), ACM, New York, NY, USA (1967). https://doi.org/10.1145/1465482.1465560, http://doi.acm.org/10.1145/1465482.1465560
2. 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/
3. 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)
4. 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
5. 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, https://doi.org/10.2514/1.I010555
6. 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
7. 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
8. 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, http://ieeexplore.ieee.org/document/8244770/
9. Yegulalp, A.F.: Fast backprojection algorithm for synthetic aperture radar. In: Proceedings of the 1999 IEEE Radar Conference. Radar into the Next Millennium (Cat. No.99CH36249). pp. 60–65 (1999). https://doi.org/10.1109/NRC.1999.767270