# Back-Projection Algorithm Optimization for On-Board Embedded SAR Imaging System

Extended Abstract of the MSc. Dissertation

Afonso Miguel Soares Fernandes

Departamento de Engenharia Electrotécnica e de Computadores, Instituto Superior Técnico

Supervisors: Prof. José Teixeira de Sousa; Dr. Rui Policarpo Duarte

***Abstract* - The Synthetic Aperture Radar (SAR) is a type of radar capable of high-resolution coherent Imaging. To produce images from SAR data, an image forming algorithm must be used. The Backprojection Algorithm is known for generating high resolution images at the expense of computational intensity. Due to this drawback, the Backprojection Algorithm has never been widely adopted for real time applications or implementation in Small, Weight and Power (SWaP) devices. The first contribution of this work is the analysis and fixed-point conversion of the Backprojection Algorithm while providing high-quality images. The second key contribution of this work is the implementation of the Backprojection algorithm in a System-on-Chip (SoC) Field Programmable Gate Array (FPGA) device. A proof of concept was done on a Zybo Z7-10 board where it was observed the resulting 512x512 image had a signal-to-noise ratio of 99.21 dB. This result was achieved in 959ms, representing a speedup of 500x over the original software implementation.**

## I. INTRODUCTION

This work is integrated in the FCT project "Synthetic Aperture Radar Robust Reconfigurable Optimized Computing Architecture" (SARRROCA, PTDC/EEI-HAC/31819/2017). The overarching goal of this project is to promote mass adoption of Synthetic Aperture Radar (SAR) imagery and its deployment on aircraft by providing reliable, portable and lighter computational on-board systems to produce real-time SAR images. This will be achieved by coupling known optimization techniques with the flexibility, power efficiency and performance of System-on-Chip (Soc) Field-Programable Gate Arrays (FPGA).

The SAR is a complex imaging data collection system with diverse sensing applications. This type of radar uses the relative motion between itself and the target to generate high resolution 2-D or 3-D images[1], making it ideal for mounting on moving platforms such as satellites, aircrafts or drones [2][3].

The Backprojection algorithm is one of the most well-suited for use in non-ideal conditions, as its working principle overcomes most of the obstacles that others can only compensate for. Being a high-quality image formation algorithm also implies being a complex and resource-intensive algorithm [1].

The purpose of this work is to develop a Hardware/Software implementation of the Backprojection Algorithm capable of producing a 512x512 pixels image per second using a small, lightweight, power-efficient device. The optimization process is based on fixed-point conversion, word-length optimization, and algorithm rescheduling coupled with a pipeline architecture.

## II. BACKGROUND AND STATE OF THE ART

### *SAR*

The SAR is a type of coherent radar capable of high-resolution coherent imaging. This type of radar uses the relative motion between itself and the target to generate high resolution 2-D or 3-D images, making it ideal for mounting on moving platforms such as satellites, aircrafts or drones [2]. Due to being an active sensor, i.e. a sensor that provides its own source of illumination, a SAR can operate during day or night. By selecting the operating frequency correctly, the microwave signal can penetrate clouds, haze, rain and fog and precipitation with very little attenuation, allowing SAR to operate in weather conditions that make the use of visible light/infrared systems unviable. SAR has been successfully used over a wide range of applications, including surveillance, forest, sea, snow and ice monitoring, mining, oil pollution monitoring, oceanography and classification of terrain [2][4][3].

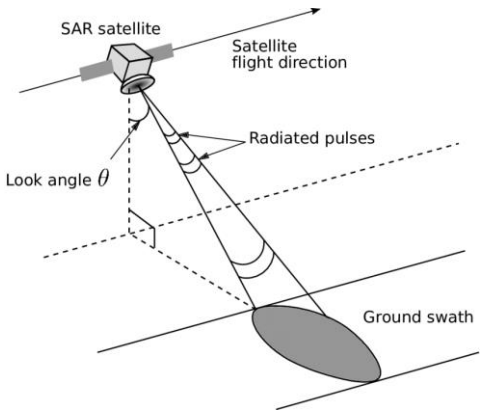


*Figure 1 - SAR's functioning principle [9]*

The SAR works by periodically emitting pulses with a well-defined spectrum and then collecting the echoes reflected back to the antenna at regular time intervals, resulting in samples containing information regarding amplitude and phase [4][3][5]. Allying the samples with the precise time at which they were recorded creates a data set that can be passed to an imaging forming algorithm to obtain the final image.

*Backprojection Algorithm*

To create SAR images an image formation algorithm is necessary, which reads the collected raw data and defines an output image [1]. The Backprojection Algorithm is a time-domain algorithm [2] sometimes referred to as the gold standard in terms of quality for SAR image forming [2][6]. It makes little assumptions and can work in a range of modes and geometries [5][1], with its biggest drawback being the high computational cost [1][2].

The Global Backprojection Algorithm uses the following parameters as inputs: carrier frequency; range from platform to centre of the swath; range bin resolution; image pixel spacing; number of pulses; number of samples per pulse; output image dimensions; platform locations (aperture points); samples from the radar. To calculate the final value for each pixel, the Backprojection Algorithm executes the following steps for every pair (pixel<->pulse) in the order they are presented [2][6]:

1. Calculate the distance between the SAR and the pixel.
2. Convert the previously calculated distance into a position (range bin) in the sample data set.
3. Compute sample value by linear interpolation between the sample of the position obtained before and the sample in the next position. The interpolation can be described by Eq.1:

$$g_{x,y}(r_k) = g(n) + \frac{g(n+1) - g(n)}{r(n+1) - r(n)} * (r_k - r(n)) \quad (1)$$

4. Compute the values for the matched filter described by the Eq.2, with $dr$ calculated according to Eq.3:

$$e^{i.\omega.2.|\overrightarrow{r_k}|} = \cos(2.\omega.dr) + isen(2.\omega.dr) \quad (2)$$

$$dr = \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} - r_c \quad (3)$$

5. Scale the sampled values by the matched filter to obtain the pulse's contribution.
6. Accumulate the contribution for the pixel. End cycle.

The mathematical formulation for this algorithm is described in Eq.4:

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

*Table 1 - Meaning of Backprojection variables*

| | |
|---|---|
| $f(x,y)$ | Output of Algorithm, final value of pixel (x, y) |
| $g(n)$ | Radar sample in range bin |
| $g(n+1)$ | Radar sample in adjacent range bin after |
| $g_{x,y}(r_k, \theta_k)$ | Reflection received by radar at $r_k$ at $\theta_k$ |
| $\omega$ | Angular velocity of radar waveform |
| $r(n)$ | Corresponding range to bin |
| $r(n+1)$ | Corresponding range to adjacent bin after |
| $x, y, z$ | Coordinates of the pixel |
| $x_k, y_k, z_k$ | Coordinates of the radar |
| $\theta_k$ | Aperture point |
| $r_k$ | Range from pixel (x,y) to aperture point $\theta_k$ |

The pseudocode for this implementation is presented in figure 2.

**Algorithm 1** Backprojection pseudocode.

```
for all pixels k do
  y_k = 0
  for all pulses p do
    R = ||a_k − v_p||  /* Distance from platform to voxel */
    bin = ⌊(R − R_0)/ΔR⌋  /* Range bin (integer) */
    if bin ∈ [0, N_BP − 2] then
      w = (R − R_0)/ΔR − bin
      /* Data sampled using linear interpolation */
      s = (1 − w) · x(p, bin) + w · x(p, bin + 1)
      y_k = y_k + s · e^{j·k_u·R}
    end if
  end for
end for
```

*Figure 2 - Pseudocode of the Backprojection Algorithm* [7]

*Image Quality Assessment*

The Signal-to-Noise-Ratio (SNR) metric was chosen to evaluate the quality of the output image and it is calculated using Eq.5 [2].

$$SNR_{dB} = 10log_{10}\left(\frac{\sum_{k=1}^{N}|s_k|^2}{\sum_{k=1}^{N}|s_k - n_k|^2}\right) \quad (5)$$

where the $s_k$ and $n_k$ terms represent the reference and output image values of the k-th element respectively, and N represents the number of values to be compared.

*A. Reconfigurable Hardware*

The family of devices that best combines the flexibility of a generic purpose CPU with the reconfigurable high-speed computing fabrics hardware results from the fusion of a SoC with a FPGA device [2]. The FPGA provides the configurable high-speed computing fabric while the SoC architecture guarantees: tight coupling between components; the existence of a microcontroller or microprocessor; the existence of external memories or connectors for external memories; a range of peripheral ports [8][9].

*Zybo Z7-10*

This work was developed to be implemented in a Zybo Z7-10 board containing a Zynq 7000 device from Xilinx.inc,

an external DDR3 memory with capacity for 1GB and I/O peripherals [10]. The Zynq 7000 is composed of a Processing System (PS), featuring an ARM Cortex-A9 dual-core processor, and a PL block. The PL is a Xilinx 7-series FPGA. The resources available are shown in table 2.

*Table 2 - Main resources in the PL of the Zybo Z7-10*

| LUT Elements | Flipflops | DSP | BRAM | Slice |
|---|---|---|---|---|
| 17 600 | 35 200 | 80 | 60 | 4400 |

The schematic of the APSoC (All Programmable SoC) is shown in figure 3.

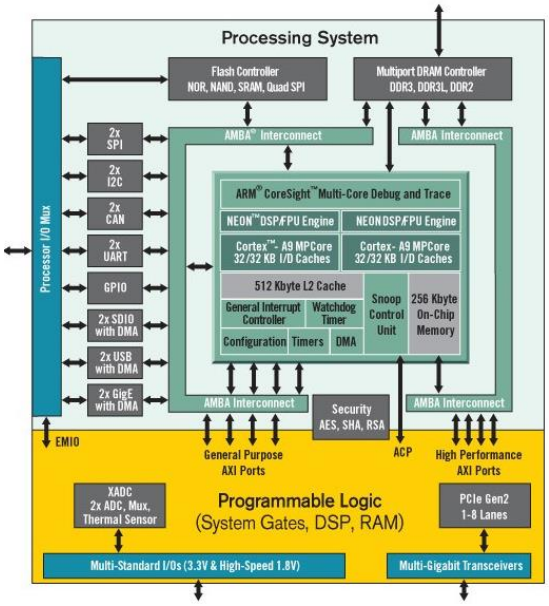


*Figure 3 - Overview of the Zybo's APSoc Architecture* [10]

The board was selected due to its embedded architecture, reconfigurable capabilities, and low power consumption.

### B. *Numeric Formats*

The design of embedded systems usually starts at the algorithmic level where an execution model is created from the general concept of an algorithm. The development at this stage in the design process is made invariably using the Floating-Point numeric format since it provides the best precision of all formats and is natively supported in most Personal Computer (PC) systems [11]. After the initial phase, design constraints change drastically due to the target system changing from a common PC system to an embedded system. In the new target, the use of a fixed-point format is known to allow drastic savings in the traditional cost metrics: silicon area, power consumption and latency/throughput [11][12][13].

### C. *Approximate Computing Techniques*

There are operations in the algorithm that do not have a standard implementation in a fixed-point format. To overcome this limitation, new implementations were made using methods capable of producing approximate results. The methods are studied with two purposes: to choose a method for software implementation and to choose a method for hardware implementation. The operations in question and the methods chosen to approximate them are in table 3.

*Table 3 - Approximation methods used in both implementations*

| | Platform | |
|---|---|---|
| **Operation** | **Software** | **Hardware** |
| Square Root | Newton-Raphson[14] | Binary restoring square root extraction[15] |
| Cosine & Sine | CORDIC[16] | CORDIC |

## III. OPTIMIZATION METHODOLOGY

### A. *Algorithm Profiling*

The first step in the optimization of the Backprojection algorithm is to identify the most time-consuming instructions of the algorithm, as these will be the ones with the higher potential for acceleration. Table 4 details the CPU present in the target device, along with the relevant configuration options

*Table 4 - Target device Specifications and Configuration*

| CPU | Operating system | Multi-Threading | Compilation flags |
|---|---|---|---|
| ARM Cortex-A9 | Bare metal | No | -O3 |

The profiling in the target system was performed using the xil_time.h library functions. The total execution time achieved was of 481s, the rest of the results are presented in figure 4.

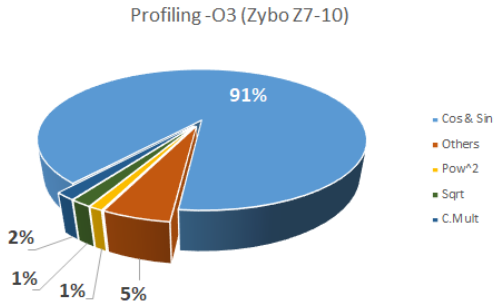


*Figure 4 - Profiling results of the Backprojection Algorithm*

It can be concluded that the Cosine and Sine operations should be the focus of the optimization process, as these are responsible for 91% of the processing time. These findings correspond to the conclusions of previous works [5].

A first proposition for the optimized system architecture is to have the Cosine and Sine operations implemented in the PL of the SoC while the rest of the algorithm still executes in the CPU. However, this architecture's execution time would still be far from the initial goal defined in the introduction of this work. This is due primarily to the fact that a cycle of the algorithm has too many instructions to allow a serial execution in the CPU to be efficient.

### B. Hardware/Software Partition

To achieve the initial goal of total execution time below the one second mark, most, if not all, of the algorithm should be executed in the PL because of the limitations of the PS system. A system partition is proposed with the following characteristics:

*Table 5 - Proposed system partition*

| | |
|---|---|
| ***PS*** | Initialization of AXI-DMA modules;<br>Control of the read and write operations to the DDR memory. |
| ***PL*** | Calculus of the Distance between the Satellite and the pixels;<br>Calculus of the matched filter values relative to the distances calculated before;<br>Sample selection using the calculated Distances;<br>Sample linear interpolation;<br>Complex multiplication between the samples and the matched filter;<br>Accumulation of the products of the complex multiplication. |

Now it is possible to pipeline all the algorithm's operations, achieving a higher throughput. The working frequency was chosen to allow the target system to achieve the desired execution time of 1 second. Solving eq. 30 and 31:

$$Execution\ Time = (Nº\ Pipeline\ stages + Nº\ Cycles)/Frequency \quad (6)$$

$$Execution\ Time = (0^{?} + 134\ 217\ 728)/x = 1s \quad (7)$$

We obtain a minimum frequency of 135 MHz. However, the closest possible clock frequency allowed in the target device is 140 MHz. Although the number of pipeline stages is unknown at this point, we can disregard it for these calculations, has its impact has no significance. Initially, for a clock frequency of 140MHz the circuit did not achieve the timing requirements. In the combinatorial logic part of the circuit, these issues were solved though the placement of pipeline stages, leaving the communications and memory accesses as the potential limiting factors.

### C. Word-Length Optimization

To optimize a circuit, it is fundamental to choose the right fixed-point format for its signals as this decision has a great impact over the resources allocated and in achieving the timing requirements of the circuit.

The original program is executed while collecting information about the interval of values each variable can take. With these results a final number of bits can be defined for the integer part of each signal. To choose the right length for the decimal part of the format it was necessary to develop a new software implementation of the Backprojection algorithm using fixed point. Due to the complexity of the relation between the attained SNR and the length of the decimal part of the variables, this part of the study was made by trial and error. Considering that every variable's fixed-point format can be deducted from the variable's relation to the input variables, the latter were split in two groups:

- Variables belonging to the calculation of the distance between the Satellite and a pixel. (satellite positions, pixel positions, Pythagorean theorem variables, sample selection variables);
- Variables belonging to the Filter and Samples (samples, matched filter, interpolation coefficients, complex multiplication variables, accumulation variables).

This new implementation of the algorithm was executed several times using different size combinations for the variables. The integer part of the variables was fixed while the decimal part was chosen to occupy the remaining bits. It was concluded that to achieve the quality minimum of 100dB of SNR while minimizing resource utilization, the best size combination is:

*Table 6 - Word-length of variables after optimization*

| Var | Size | Format |
|---|---|---|
| Distance | 40 | 15Q25 |
| Data/Filter | 24 | 2Q22 |

This combination achieves an SNR of 100.469dB. The chosen fixed-point formats all use rounding as the quantization mode. The need for an overflow mode was overcome with the selected choices for the word-lengths.

### D. Resource Utilization Study

The Zybo Z7-10 board used has limited resources and their use adds an associated power consumption. The goal of this study is to provide a relation between the size of the operators (inputs and output) and the resources allocated for that IP core to obtain the information needed to identify potentially expensive areas and estimate the values of the resources allocated by the future design. After gathering the information needed to make the circuit estimates, two estimations were, one for a hypothetical circuit made based on 64 bit wide input variables (Estimate A) and other made using the formats in table 6 (Estimate A). The estimates include only the circuit responsible for performing the mathematical operations of the Backprojection algorithm, excluding the square root.

*Table 7 - Resources required by Estimate A*

| **Estimate A: Total Hardware Resources** | | |
|---|---|---|
| **LUT** | **Register** | **DSP** |
| 16 823 | 21 691 | 226 |

*Table 8 - Resources required by Estimate B*

| **Estimate B: Total Hardware Resources** | | |
|---|---|---|
| **LUT** | **Register** | **DSP** |
| 8 997 | 11 452 | 69 |

As per tables 7 and 8, the resource consumption is much lower using optimized word lengths, validating this method as a valuable optimization procedure. From the analysis of this study it is expected that the design will fit the chosen target device.

Another resource study was made to evaluate the proposed architecture, this one focused on the memory types chosen to store the input and output values, as well as the memory capacity required by this architecture. To achieve the overall goals of this work the memory access operations must have a maximum latency of 1 clock cycle. This is a necessity because for every cycle of the algorithm a new address for reading input values is calculated. To meet the aforementioned latency requirements while maintaining the algorithm's schedule, the only viable option is to hold all the data in the Block RAM (BRAM) cells present in the PL fabric of the target device.

*Table 9 - Estimate of BRAM required to store all input/output values*

| Data | 36Kb BRAM cells |
|---|---|
| Satellite Position: | 3 |
| X | 1 |
| Y | 1 |
| Z | 1 |
| Satellite Samples | 3648 |
| Output Image | 0 |
| Board Resources | 60 |

Since the total size of the samples (in bits) is larger than the total capacity of the BRAMs in the board, all solutions considered from this point forward for the storage of the sample values are based on an architecture with 2 equal BRAMs, where one is being read while the other is updated with the next values. This dynamic happens throughout the execution of the system.

### E. Algorithm Rescheduling

The original schedule requires more BRAM than the available in the board, as shown in table 9. To overcome this limitation, a combination of memory management and algorithm rescheduling was used. The cycles of the Backprojection algorithm are independent from one another with the accumulation phase being the only part where the dependency between cycles becomes relevant. Three schedules were studied:

***Pixel Computation*:** for each pixel calculate and accumulate the contribution of all pulses, then go to the next pixel. This was the original schedule.

***Pulse Computation*:** for each pulse calculate its contribution to every pixel, and then go to the next pulse.

***Pixel Region Computation*:** for each pulse calculate its contribution to every pixel in a region, then go to the next pulse. After the contribution of all the pulses has been calculated, go to the next region. A region is considered a parcel of the whole image under formation.

The pseudocode of the organization of each schedule is presented in table 10, with the details presented in table 11.

*Table 10 - Pseudocode of the loop hierarchy for the schedules studied*

| Schedule | Pseudo Code |
|---|---|
| Pixel Computation | For every **Pixel** do:<br>For every **Pulse** do: |
| Pulse Computation | For every **Pulse** do:<br>For every **Pixel** do: |
| Pixel Region Computation | For every **Region** do:<br>For every **Pixel** in the **Region** do:<br>For every **Pulse** do: |

## IV. PROPOSED SYSTEM ARCHITECTURE

The proposed architecture for Hardware Accelerator implemented in the PL system is presented and explained in detail. The circuit was designed in Vivado due to the program belonging to the same manufacturer as the FPGA. All the custom-built modules and circuits were debugged using the Vivado Simulator, except the montage of the whole system (that includes the PS+PL system), that was debugged using the Integrated Logic Analyser (ILA). The architecture was divided by function into three parts, as shown in figure 5.

1. Memory (Purple box + Pink box)
2. Algorithm Execution (Orange box)
3. System Control (light purple box)

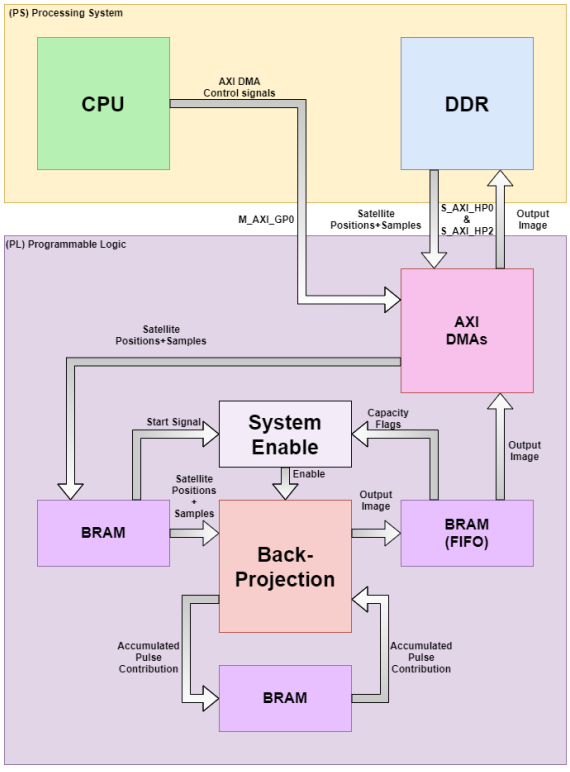


*Figure 5 - Proposed system architecture, discriminating memory units and inter-system communication.*

*Table 11 - Overview of the schedules' requirements and implementation details*

| | **Schedules** | | |
|---|---|---|---|
| | Pixel Computation | Pulse Computation | Pixel Region Computation |
| **Input Requirements** | Every cycle:<br>-X, Y and Z values for the new satellite position;<br>-Real and imaginary parts of every sample from the next pulse; | Every 262,144 cycles:<br>-X, Y and Z values for the new satellite position;<br>-Real and imaginary parts of every sample from the next pulse; | Every REGION SIZE cycles:<br>-X,Y and Z values for the new satellite position;<br>-Real and imaginary parts of every sample from the next pulse; |
| **Intermediate values Requirements** | Hold 2 values at a time; | Hold 524,288 values at a time; | Hold 2*REGION SIZE values at a time; |
| **Output Requirements** | Every 512 cycles:<br>-2 values to be stored in the DDR; | End of Execution:<br>-524,288 values to store in the DDR; | Every REGION SIZE * 512 cycles:<br>-REGION SIZE values to be stored in the DDR; |
| **Advantages** | - Output: very low throughput; is the final value for each pixel.<br>- Bandwidth requirements for storing output values: low; achievable with the Zybo board's resources | -Updates: the BRAM holding the samples only needs to be updated every 262,144 cycles. | Updates: the BRAM holding the sample values needs to be updated every REGION SIZE cycles.<br>To work the REGION SIZE needs to be bigger than 4096 (amount of values to update). |
| **Disadvantages** | - Unfeasible input requirement regarding sample values.Too much Bandwidth required. | - Unfeasible intermediate values requirement. Not enough BRAM available | -Region size depends on BRAM available.<br>-Region size cannot be smaller than the number of samples per pulse. |
| **Implementation Requirements** | - Does not require BRAMs in the output. | -Does not need BRAMs for the Satellite positions; the time between the need for new positions allows other solutions. | - |
| **36Kb BRAM** | 3651 | 924 | 45 |
| **18Kb BRAM** | 0 | 3 | 4 |
| **Resources** | 36Kb BRAMs: 60, each one can be split into two 18Kb BRAMs. | | |
| **Feasibility** | No | No | Yes |

### *A. Memory Units*

*DDR*

The DDR memory is used to hold all the input data files and the output image. The inputs consist of the Satellite position file, containing all the positions of the Satellite in the format *24Q40*, and the Satellite sample file, containing all the samples in the format *10Q22*.

The AXI DMA IP core coordinates the accesses made by the circuit to the DDR memory, due to the high performance achievable using the HP ports and DMA feature. All the interfaces used to read/write data feature a 64-bit wide data channel allowing for the processing of a word per clock cycle, in any direction. The use of the module is further incentivised by the fact that the connections to the circuit are made using the AXI Stream protocol. Two instances of this IP core are used, the first being responsible for reading satellite positions and writing the real part of the output image and the second being responsible for reading the satellite samples and writing the imaginary part of the output image.

*BRAM*

The BRAM modules are used to hold all the Satellite positions, all the Samples from 2 pulses at a time and the intermediate/final values of the pixels in a region. Table 12 presents the configurations for these modules along with their contents.

*Table 12 - BRAM modules configurations*

| ID | Content | Format | Word Depth | Mode | 36Kb BRAM cells |
|---|---|---|---|---|---|
| B1.1 | X coordinate (position) | 1 value per word | 512 | Simple Dual Port | 1 each |
| B1.2 | Y coordinate (position) | | | | |
| B1.3 | Z coordinate (position)* | | | | |
| B2.1 | Samples from pulse *X* | -Real part[63:32] | 4096 | True Dual Port | 7.5 each |
| B2.2 | Samples from pulse *X+1* | -Imaginary part[31:0] | | | |
| B3.1 | Real part of Output Image | 1 value per word | 8192 | Simple Dual Port | 14.5 each |
| B3.2 | Imaginary part of Output Image | 1 value per word | | | |
| B4.1 | Output FIFO | 1 value per word | 2048 | X | 4 each |
| B4.2 | | | | | |

*This BRAM module was supressed in the final system due to lack of BRAM available and because the data set used had fixed Z coordinate for the platform in all pulses.

All the BRAMs feature a memory controller interface and an AXI Stream interface along with a finite state machine to control them. The modules also feature a word counter to manage the addressing of writing operations and the memory switching while writing. The controllers are now divided by the 4 different memory modules.

*C. Algorithm Execution Circuit*

To facilitate the comprehension of this work, the presentation of the circuit is divided into 5 parts, where each part has a well-defined purpose and functionality. The circuit is divided as follows:

1. **Distance** -> Calculates the distance between the satellite and a given pixel.
2. **Sample** -> Computes the sample values for each pair: pulse[i] ↔ pixel[x, y].
3. **Filter** -> Calculates the values of the Matched Filter for the distance provided.
4. **MultC** -> Computes the complex multiplication between the sample and the matched filter.
5. **Accumulator** -> Accumulates the contribution of the pulses for each pixel.

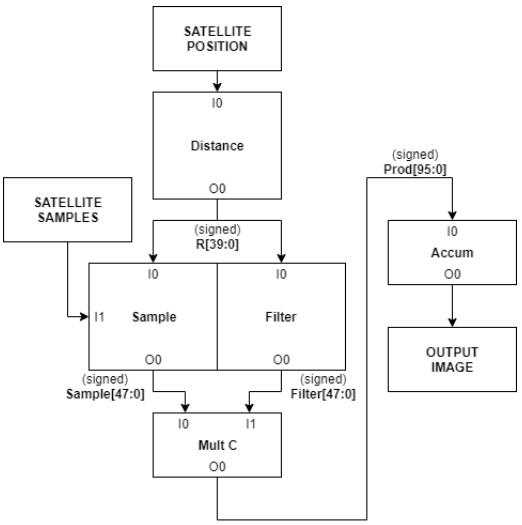


*Figure 6 - Hardware Accelerator Top-Level Architecture*

*Distance Module*

The Distance module is composed of the B1 memories and the modules that perform the arithmetic operations responsible for calculating the distance between the Satellite platform and the Pixel coordinates. The module reads the satellite position from the B1 memories and receives the pixel position from the Pixel Position module, calculating the first part of the Pythagorean theorem (Radicant), described in the following equations:

$$X_1 = (plat_x - p_x)^2 \quad (8)$$

$$Y_1 = (plat_x - p_y)^2 \quad (9)$$

$$Radicant = X_1 + Y_1 + Z_1 \quad (10)$$

The result is then passed to the Square Root module to obtain the final value of the distance. Since the Z coordinate of the satellite ($plat_z$) is constant and the z coordinate of the image is always zero, the optimization decision of cutting the Z coordinate part of the circuit was made. This decision allowed to free BRAM cells.

*Pixel Position Module*

The Pixel Position module was implemented using binary counters, logic gates and one adder to produce the correct sequence of pixel positions, calculating a new value every clock cycle. It is also responsible for calculating the reading addresses for the B1 memories. Due to the schedule chosen for the algorithm's execution the pixels' positions are calculated by the equations presented next:

$$Ry = 16 * region;\ region \in [0,32]; \quad (11)$$

$$Offset_x = (-255,5 * dxdy); \quad (12)$$

$$Offset_y = (-255,5 * dxdy) + (Ry * dxdy); \quad (13)$$

$$p_x = Offset_x + (ix * dxdy);\ ix \in [0,511]; \quad (14)$$

$$p_y = Offset_y + (iy * dxdy);\ iy \in [0,15]; \quad (15)$$

The Pixel Position module is also responsible for controlling one important flag, the Sample memory switch flag. This flag signals the Sample module when to switch memories for reading, meaning the next iterations are

pertaining contributions of the next pulse. This flag is set to one every time the Pixel Position module reaches the position of the final pixel of a region.

*Square Root Module*

The Square root module is responsible for performing the square root operation that ends the calculus of the distance between the Satellite and the Pixel. This operation is performed by a custom module built to perform the square root using the Binary restoring square root extraction method. Because this method uses two input bits to calculate one bit of the output per layer, the total latency of the module is half the number of bits in the output rounded up, which in this case is thirty-nine cycles.

*Samples Module*

The Sample module is composed by the B2 memory, the WBin module, the Interpolation module, an adder belonging to the B2 memory controller and a Shift Register. The shift register was introduced to compensate for the difference in latencies in the Datapath of the Bin signal and the Datapath of the W1 and W2 signals. The adder is used to compute the address of the next sample of the pulse by adding one to the address of the current sample.

*WBin Module*

The WBin module calculates the address (Bin) where the correct sample for a given distance is, as well as the interpolation coefficients (W_1 and W_2) to be used by the Interpolation module. This module was implemented with basic arithmetic operation modules and the concatenation module.

The operations performed by this module are described in the following equations:

$$Bin = \lfloor (R - R_0) * dR_{inv} \rfloor \quad (16)$$

$$W_2 = ((R - R_0) * dR_{inv}) - Bin \quad (17)$$

$$W_1 = 1 - W_2 \quad (18)$$

For efficiency purposes, in the implemented circuit the multiplication by the coefficient $dR_{inv}$ was replaced by an arithmetic left shift due to $dR_{inv}$ being a constant whose value is a positive power of two (32 = 2^5). The subtraction was also replaced by simply splitting the bin_0 bus to separate the integer from the decimal part. This module has a latency of one cycle due to the buffer on the outputs.

*Interpolation Module*

The Interpolation module performs the linear interpolation described by the equations:

$$Sample_{Re} = (data1_{Re} * W_1) + (data2_{Re} * W2) \quad (19)$$

$$Sample_{Im} = (data1_{Im} * W_1) + (data2_{Im} * W2) \quad (20)$$

The interpolation is done between the samples from the Bin address and the Bin+1 address of the B2 memory, using the coefficients calculated by the Wbin module.

*Filter Module*

The Filter module is responsible for calculating the matched filter values from the distance value received. The module was built using basic mathematical IP cores, the CORDIC IP core, a Shift Register, a custom multiplexer and support cores, like the concatenation IP core. This module can be divided into three parts: Argument calculator, Trigonometric stage, and Quadrant normalizer.

The Argument calculator is the circuit responsible for converting the value of the distance (in meters) received into the value of an angle (in radians). After the conversion, the quadrant is stored in a shift register and the angle is rotated to the first quadrant.

The Trigonometric stage employs the circuit composed by the CORDIC IP core and it is responsible for calculating the sine and cosine of the angle received as input. The configuration of the CORDIC IP core results in an IP core with a latency of 28 cycles.

The Quadrant normalizer follows the Trigonometric stage and is responsible for correcting the effects of the angle rotation described in the first stage. This module was implemented by providing the sine and cosine values for all the possible 90 degrees angle rotations (sin(a), -sin(a), cos(a), -cos(a)) and using the quadrant number to select the correct option.

*Complex Multiplication Module*

The MultC module is responsible for performing the complex multiplication between the samples and the matched filter. It was implemented using only basic mathematical IP cores, performing the operations detailed in eq.52 and 53:

$$Prod_{Re} = (Sample_{Re} * Filter_{re}) - (Sample_{Im} * Filter_{Im}) \quad (21)$$

$$Prod_{Im} = (Sample_{Re} * Filter_{Im}) + (Sample_{im} * Filter_{Re}) \quad (22)$$

*Accumulator Module*

This module performs the accumulation and stores the results in the B3 memory for every cycle, with only one exception pertaining to the last accumulation of a pixel, where the result is passed to the FIFO in the next module. The operations performed by this module are detailed in eq.23 and 24:

$$Accum_{Re_{i+1}} = Accum_{Re_i} + Prod_{Re_i};\ i \in [0; 511] \quad (23)$$

$$Accum_{Im_{i+1}} = Accum_{Im_i} + Prod_{Im_i}\ ;\ i \in [0;511] \qquad (24)$$

*D. System Control*

The accelerator has an enable signal that runs through the whole Datapath, going from the System control module to the Accumulator modules. With this signal it is possible to know if a value stored in a pipeline stage is valid or not. The need for a control system stems from the existence of an initialization phase to fill the BRAM modules, because all the positions of the satellite and all the samples from the first pulse must be written in the BRAMs before the execution can start. Also, due to the output values being sent for storage in bursts, a FIFO had to be added to solve the lack of synchronization between the AXI DMAs and the Accumulator modules.

## V. RESULTS

*A. SNR*

The Hardware/Software implementation produced an image with a SNR value of 99.210 dB. The result is below the projected SNR value of 100.469 dB by 1.259 dB and below the threshold of 100dB proposed in the beginning of this work by 0.79 dB. Figure 7 shows the image taken as the golden reference and the image formed by the fixed-point Hardware/Software implementation, respectively.

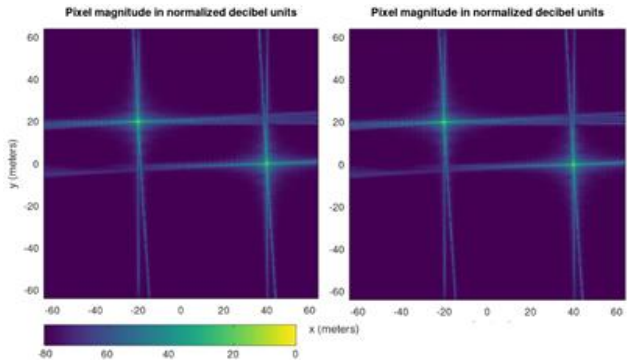


*Figure 7 – Golden reference (left), Output of the final Hardware/Software implementation (99.21 dB; right)*

This difference was not predicted but was expected considering that the exact implementation of the CORDIC algorithm in the IP core is hidden to the user and is protected by Xilinx.

*B. Resources*

The resources allocated by the final design can be found in their entirety in the target device. When comparing the resource allocation with estimations made we can conclude that the predicted savings made through the word length study are valid predictions and that the study allowed for some optimization in that area.

*Programmable Logic Resources*

Table 13 shows the amount of resources from the PL system allocated by the design after the implementation phase performed in Vivado.

*Table 13 - Resources used in PL by the whole system*

| Name | LUTs | Registers | DSPs | BRAM |
|---|---|---|---|---|
| Distance | 2292 | 2807 | 0 | 2 |
| Sample | 398 | 356 | 10 | 13 |
| Filter | 2198 | 2491 | 25 | 0 |
| MultC | 0 | 68 | 10 | 0 |
| Accum | 219 | 231 | 0 | 14.5 |
| Init | 2 | 2 | 0 | 0 |
| Axi DMA | 1636 | 2407 | 0 | 3 |
| Axi-Interconnect (Axi-Stream) | 543 | 650 | 0 | 0 |
| Axi-Interconnect (Axi-Lite) | 505 | 657 | 0 | 0 |
| Axi Data FIFO | 60 | 88 | 0 | 4 |
| FIFO Transfer Control | 8 | 20 | 0 | 0 |
| Processor System Reset | 16 | 33 | 0 | 0 |
| **TOTAL** | 10438 (59.3 %) | 13304 (37.7 %) | 55 (68.8 %) | 58 (96.7 %) |

The rescheduling of the algorithm proved to be one of the biggest contributions to optimization in the design of the circuit, by lowering the BRAM resources needed by the implementation enough to fit the target device. This alteration reduced the BRAM needed by a factor of 77 times.

*CPU*

The final design only used one core of the Dual-core ARM Cortex-A9 CPU.

*C. Timing and Latency*

The total execution time of the system was 0.96 seconds. This time was achieved with a working frequency of 140 MHz on the PL clock.

Table 14 shows the latency of the top-level modules of the system.

*Table 14 - Latency of the Top-Level modules*

| Name | Latency (cycles) |
|---|---|
| Distance | 46 |
| Sample | 6 |
| Filter | 44 |
| MultC | 3 |
| Accum | 1 |
| **TOTAL** | 100 |

The timing objective was fulfilled. The total execution time obtained represents a speedup of 500x over the original software only implementation, executed in the PS system of the target device. The pipelined architecture and the rescheduling of the algorithm allowed the overheads from

accessing the external memory to be hidden without impacting the execution time.

### D. Energy Consumption

The energy consumption estimate provided by Vivado appears in table 15.

*Table 15 - Power consumption discriminated by component of the target device*

| Component | Power (Watts) | Power (%) |
|---|---|---|
| CPU | 1.406 | 75 |
| Signals | 0.129 | 7 |
| BRAM | 0.118 | 6 |
| Logic | 0.101 | 5 |
| DSP | 0.067 | 4 |
| Clocks | 0.064 | 3 |

From the results, it can be concluded that the CPU is by far the most power consuming component used. When relating the power consumption profile presented in table 15 with the Hw/Sw partition employed in the final implementation, a large discrepancy between the CPU's workload and its consumption can be found due to this component's static consumption.

## VI. CONCLUSION

The aim of this work was to design, develop and evaluate an optimized system to compute the imaging forming algorithm for a SAR system. The design was developed around two main ideas: the advantage offered by flexibility of an FPGA regarding parallel and pipeline architectures and the advantages offered by the fixed-point format. Both the initial studies performed on the Backprojection algorithm and their conclusions are generic enough to be used in the study of almost every other algorithm, with the exception of the rescheduling that presents a reasoning very specific to the characteristics of the algorithm. The results obtained in the software fixed-point implementations are one more proof that the methods used today to approximate mathematical functions can provide a high-quality solution, opening a lot of opportunities to the adoption of fixed-point architectures in optimization efforts.

The fact that the device used in this work is low-end should emphasise the potential of the SoC FPGA family in optimization efforts. The energy consumption could be greatly improved. Due to the Hw/Sw partition present in the design, the CPU was left with very few instructions to perform. Furthermore, the instructions that the CPU still performs are only pertaining the control of the AXI-DMA blocks and can be easily implemented by a finite state machine. Floor-planning is outside the scope of this work, however it could be an effective method to achieve a faster final circuit. This opinion is based on the high impact of the Net Delay values in the Total Delay values of the slowest paths in the circuit.

All these improvements could result in a useful real-time application in a device with good portability characteristics.

## VII. ACKNOWLEDGEMENTS

This thesis is integrated in the FCT project "Synthetic Aperture Radar Robust Reconfigurable Optimized Computing Architecture" (SARRROCA, PTDC/EEI-HAC/31819/2017).

## References

[1] M. I. Duersch, "Backprojection for Synthetic Aperture Radar," *J. Phys. A Math. Theor.*, vol. 44, no. 8, pp. 1689–1699, 2011, doi: 10.1088/1751-8113/44/8/085201.

[2] H. Cruz, R. P. Duarte, and H. Neto, *Fault-Tolerant Architecture for On-board Dual-Core Synthetic-Aperture Radar Imaging*, vol. 93, no. 6. 2006.

[3] Y. K. Chan and V. C. Koo, "An introduction to Synthetic Aperture Radar (SAR)," *Prog. Electromagn. Res. B*, vol. 2, pp. 27–60, 2008, doi: 10.2528/pierb07110101.

[4] H. Balzter, "Forest mapping and monitoring with interferometric synthetic aperture radar (InSAR)," *Prog. Phys. Geogr.*, vol. 25, no. 2, pp. 159–177, 2001, doi: 10.1191/030913301666986397.

[5] J. Park, P. T. P. Tang, M. Smelyanskiy, D. Kim, and T. Benson, "Efficient backprojection-based synthetic aperture radar computation with many-core processors," *Sci. Program.*, vol. 21, no. 3–4, pp. 165–179, 2013, doi: 10.3233/SPR-130372.

[6] D. Pritsker, "Efficient Global Back-Projection on an FPGA," *IEEE Natl. Radar Conf. - Proc.*, vol. 2015-June, no. June, pp. 204–209, 2015, doi: 10.1109/RADAR.2015.7130996.

[7] K. B. Pnnl *et al.*, "PERFECT Benchmark Suite Manual," vol. 5, pp. 0–32.

[8] C. Zhang, "Optimizing FPGA-based Accelerator Design for Deep.pdf," *ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays(FPGA)*, pp. 161–170, 2015.

[9] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," *FPL 09 19th Int. Conf. F. Program. Log. Appl.*, pp. 126–131, 2009, doi: 10.1109/FPL.2009.5272532.

[10] Digilent, "Zybo Z7 Board Reference Manual," *Digilent, Inc.*, pp. 1–30, 2017, [Online]. Available: www.store.digilent.com.

[11] P. Belanović and M. Rupp, "Automated floating-point to fixed-point conversion with the fixify environment," *Proc. Int. Work. Rapid Syst. Prototyp.*, pp. 172–178, 2005, doi: 10.1109/rsp.2005.15.

[12] I. Engineering, "Fixed-Point Optimization Utility for C and C+ based Digital Signal Processing Programs," pp. 197–206, 1995.

[13] C. Shi and R. W. Brodersen, "An automated floating-point to fixed-point conversion methodology," *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2, pp. 529–532, 2003, doi: 10.1109/icassp.2003.1202420.

[14] I. Newton and I. Newton, "Historical development of the newton-raphson method," vol. 37, no. 4, pp. 531–551, 1995.

[15] K. Turkowski, "Fixed-Point Square Root," *Graph. Gems V*, no. 96, pp. 22–24, 1995, doi:10.1016/b978-0-12-543457-7.50011-5.

[16] J. Volder, "The CORDIC computing technique," *Proc. West. Jt. Comput. Conf. IRE-AIEE-ACM 1959*, pp. 257–261, 1959, doi: 10.1145/1457838.1457886.