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

Afonso Miguel Soares Fernandes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. José João Henriques Teixeira de Sousa,

Dr. Rui António Policarpo Duarte

January 2021

Acknowledgments

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

With thanks to Dr.Rui Duarte for his guidance, and Prof.Horácio Neto for always finding time for my curiosities.

Resumo

O Radar de Abertura Sintética (SAR) é um radar capaz de criar imagens coerentes de alta resolução. Para produzir imagens a partir de informação recolhida por um SAR é necessário usar um algoritmo de formação de imagem. Backprojection é um algoritmo conhecido por gerar imagens de alta qualidade com a desvantagem de ser computacionalmente intensivo. Devido a esta desvantagem o Backprojection nunca foi adoptado em aplicações em tempo real ou em dispositivos de baixo consumo energético e dimensões e poder computacional reduzido.

O primeiro contributo deste trabalho é o estudo do algoritmo Backprojection e a sua conversão para fixed-point. A caracterização temporal foi feita através de profiling. Para fazer a conversão foram conduzidos dois estudos de forma a escolher os parâmetros do formato fixed-point.

O segundo contributo é a implementação do algoritmo Backprojection num dispositivo System-on-Chip (SoC) Field Programmable Gate Array (FPGA). A implementação deve ser capaz de produzir uma imagem de 512 x 512 pixeis por segundo. A optimização é focada na aplicação e uma arquitectura pipeline.

Usando uma placa Zybo Z7-10 foi possível implementar o sistema de processamento capaz de produzir uma imagem com um SNR de 99.21 dB em apenas 959ms. Sendo o dispositivo escolhido da gama mais baixa, este trabalho comprova o enorme potencial que esta família de dispositivos tem no futuro do processamento de imagem em tempo real.

Palavras-chave: Synthetic Aperture Radar, Backprojection, FPGA, ponto fixo, projecto HW/SW, optimização

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 time characterization was done by means of profiling. To perform the fixed-point conversion two studies were conducted in order to set the parameters of the fixed-point format.

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. The implementation should be capable of producing an image with 512 per 512 pixels, per second. The optimization is focused on the development of a pipeline architecture.

Using a Zybo Z7-10 board it was possible to achieve a successful design, capable of producing an image with a signal-to-noise ratio of 99.21 dB in 959ms. With the target system being an entry-level device this works proves that these devices have an immense potential in the future of image processing for real-time applications.

Keywords: Synthetic Aperture Radar, Backprojection, FPGA, fixed-point, optimization, Hw/Sw system design

Table of Contents

List of Figures

List of Tables

Acronyms

AMBA – Advanced Microcontroller Bus Architecture

APSoC – All Programmable SoC

ASIC – Application Specific Integrated Circuit

AXI – Advanced eXtensible Interface

BRAM – Block RAM

CPU – Central Processing Unit

CORDIC – Coordinate Rotation Digital Computer

DDR – Double Data Rate

DMA – Direct Memory Access

DSP – Digital Signal Processor

FIFO – First In, First Out

FPGA – Field Programable Gate Array

IEEE - Institute of Electrical and Electronics Engineers

ILA – Integrated Logic Analyser

IP – Intellectual Property

IST – Instituto Superior Técnico

GP – General Purpose

GPU – Graphics Processing Unit

HP – High-Performance

LSB – Least Significant Bit

LUT – Look-Up Table

MSB – Most Significant Bit

PC – Personal Computer

PL – Programmable Logic

PLL – Phase-Locked Loop

PS – Processing System

RAM – Random Access Memory

SAR – Synthetic Aperture Radar

SLAR – Side-Looking Aperture Radar

SNR – Signal-to-Noise Ratio

1 Introduction

This work was developed in the scope of a master's thesis to conclude the IST Integrated Master's in Electrical and Computer Engineering course. The thesis is integrated in the Synthetic Aperture Radar Robust Reconfigurable Optimized Computing Architecture (SARRROCA) project (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. It captures several target parameters, providing unique information that complements standard optical remote sensing methods, which can use a wide range of image formation algorithms. 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. For these reasons, the optimization of the Backprojection algorithm is an active field of research [1].

The main goal of this work is the optimization of the Back-Projection Algorithm for On-Board Embedded SAR Imaging System.

1.1 Motivation

Radar data has historically been used to advance the Earth study science, with applications in areas ranging from resource identification and monitoring, to landscape mapping and monitoring of the Earth's natural changes.

However, radar data is also expensive and complicated to use. The advances in the past decades in this area have broadened the scope of radar uses, allowing satellite remote sensing datasets to be used in a series of applications as an option for sustainable and replicable methods

Technological advances in electronic appliances have unveiled unique opportunities that take advantage of the enhanced data processing capacity of simple devices, like FPGA, allowing the exploitation of algorithms operating in real time applications using a small, lightweight, mobile, energetically efficient and battery-powered device, which is an ever growing need [2].

The advantages of coupling data collection with image formation techniques in the devices is the reduced time consumption for image processing at the receivers [1]. This allows the widespread use of these devices and technology in applications such as surveillance radars and mapping of natural resources and geological structures, among others.

1.2 Scope and Objectives

The Backprojection Algorithm is an algorithm capable of generating high quality images of the Earth's surface. However, it is one of the most computationally intensive image formation algorithms which has led to it being overlooked for real time applications (especially in small, light and power efficient platforms) [3].

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. At the same time, this work aims to build a generalized methodology for optimizing algorithms.

The developed architecture implements the Backprojection Algorithm for image formation using SAR data samples. The architecture is composed of a hard-core Central Processing Unit (CPU) coupled with FPGA fabric, forming a Soc device. The target device used in this work is the Zybo Z7-10 (Digilent.inc, Washington, USA) which has a Zynq 7000 FPGA containing a dual-core ARM Cortex-A9 processor and Programmable Logic (PL) in which custom hardware will be implemented. The advantages brought by the target device chosen coupled with the use of fixed-point arithmetic will allow the implementation to surpass the previous software implementations without compromising too much the quality of the output. This work will accompany the development of this system with all the studies performed to guarantee reproducibility for optimization studies of a range of algorithms.

1.3 Contributions

The main contributions of this thesis are:

- Wordlength study of the Backprojection algorithm;
- Conversion of the algorithm from floating-point to fixed-point;
- Development of a methodology for algorithm optimization
- Hardware/Software (Hw/Sw) implementation of the Backprojection algorithm in a SoC device.

1.4 Structure and Organization

This thesis is structured in six chapters:

Chapter 1: Motivation, objectives, main contributes and structure of this work are described.

Chapter 2: Presents context and the basic knowledge about SAR, Backprojection Algorithm, Reconfigurable Hardware and numeric formats needed in the scope of this work. It concludes by presenting the approximate techniques used in the acceleration of the algorithm.

Chapter 3: Presents the studies performed to the Backprojection Algorithm and its implementation in the C language, along with a study about the target system. This chapter includes a profiling study, a word-length study, a resource study and a schedule study.

Chapter 4: The full implemented architecture is described in detail. The chapter focus is on the Datapaths, the resources used and the timing considerations of the modules.

Chapter 5: Presents the results obtained in terms of output quality, energy consumption, performance and resources used, and the respective analysis.

Chapter 6: Presents the conclusions drawn from the work and discusses future work.

2 Background and State of the Art

This chapter introduces the state of the art for this work. It begins by introducing the SAR technology in terms of main characteristics, functioning principle, image formation algorithms and image quality evaluation, with emphasis on the image formation algorithm chosen for this work, the Backprojection algorithm. The chapter continues by presenting the reconfigurable hardware technology, presenting its advantages and disadvantages when compared to common computer systems and Application Specific Integrated Circuits (ASICS). Following is the numeric format section, in which the floating-point and fixed-point representations are presented and compared in the context of this work. The chapter finishes with an explanation of the approximate computing techniques studied for use on the implementation of the square root operation and trigonometric functions.

2.1 SAR

The SAR is a type of coherent radar capable of high-resolution coherent imaging [4][5]. 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 [1][4][6]. In development since the 1950's, it has since then evolved into a powerful and valuable tool for monitoring the Earth’s environment, among other uses [7][8][6].

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 [4][5][7][6]. 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 [1][4][5][7][9].

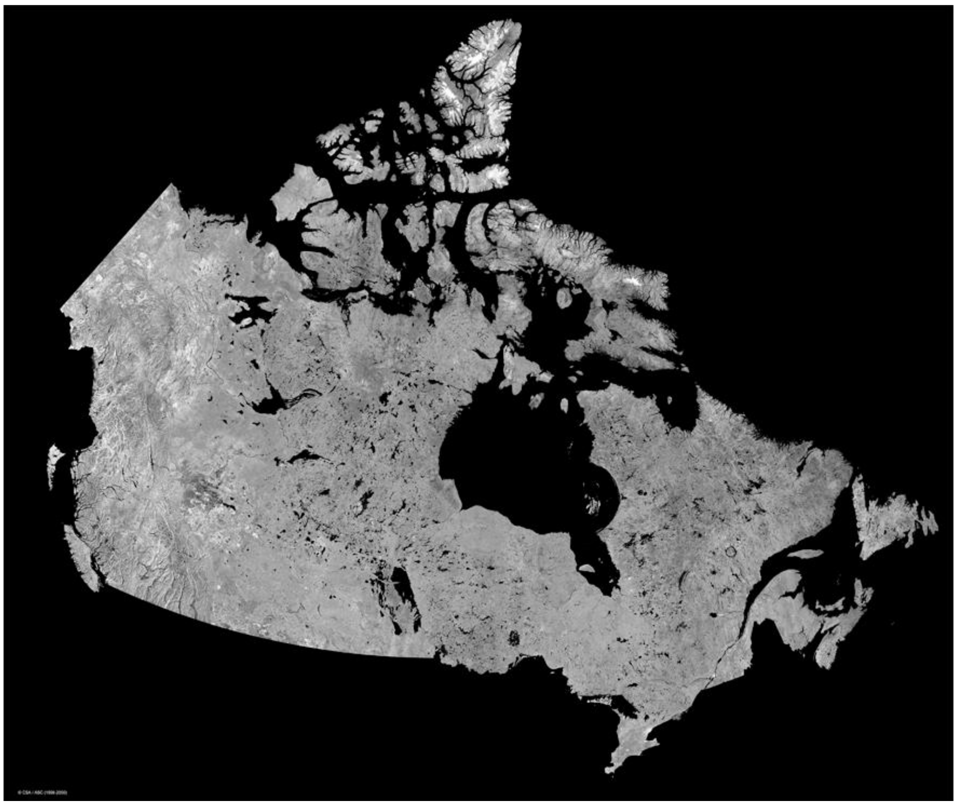


Figure 1-RADARSAT-1 100-metre resolution mosaics of the Canadian provinces over a brief seven-day period in January 1999 (https://www.nrcan.gc.ca/earth-sciences/geomatics/satellite-imagery-and-air-photos/sensors-and-methods/synthetic-aperture-radar/10968).

SAR has been successfully used over a wide range of applications, including surveillance [3], forest, sea, snow and ice monitoring, mining, oil pollution monitoring, oceanography and classification of terrain [4] [7] [10]. Figure 1 shows an example of these applications.

2.1.1 Functioning Principles

The SAR gets its name from the signal processing method it uses, emulating an antenna with big aperture using a moving antenna of smaller aperture[2] [7]. This represents the main difference to its predecessor, the Side-Looking Aperture Radar (SLAR), that had poor azimuth resolution and could only solve the problem by using shorter wavelength signals (that have high attenuation in the atmosphere) or using an impractically long antenna [7].

In a SAR system an antenna is usually mounted on an airborne or spaceborne moving platform [2][5]. The platform moves in a straight line with the antenna pointing in a direction perpendicular to the azimuth, while illuminating the desired surface at an angle, the Slant Range (with 0º being the direction straight down from the platform) [9][10] .

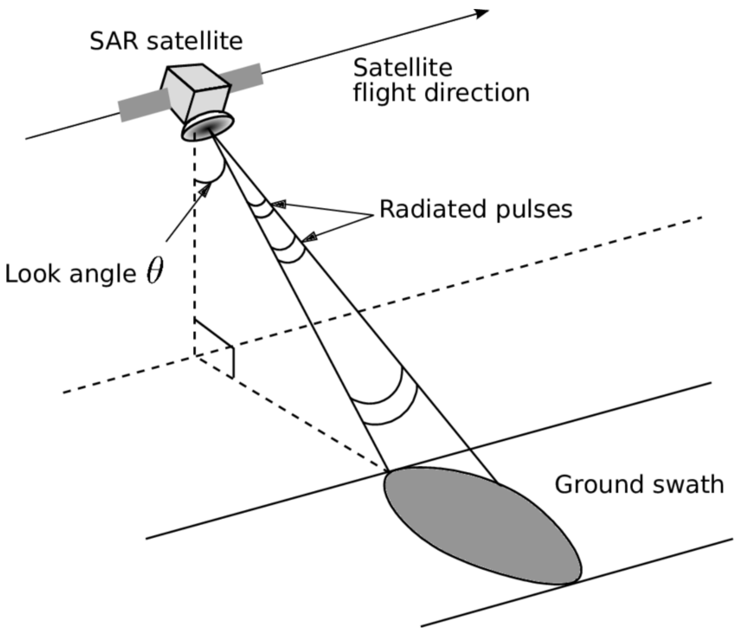


Figure 2 - SAR's functioning principle [10]

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 [2][3][4][8][11][12]. 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 [12].

2.1.2 Image Formation Algorithms

To create SAR images an image formation algorithm is necessary [1][3]. The algorithm reads the collected raw data and, through several calculations defines an output image, as schematically presented in Figure 3 [5].

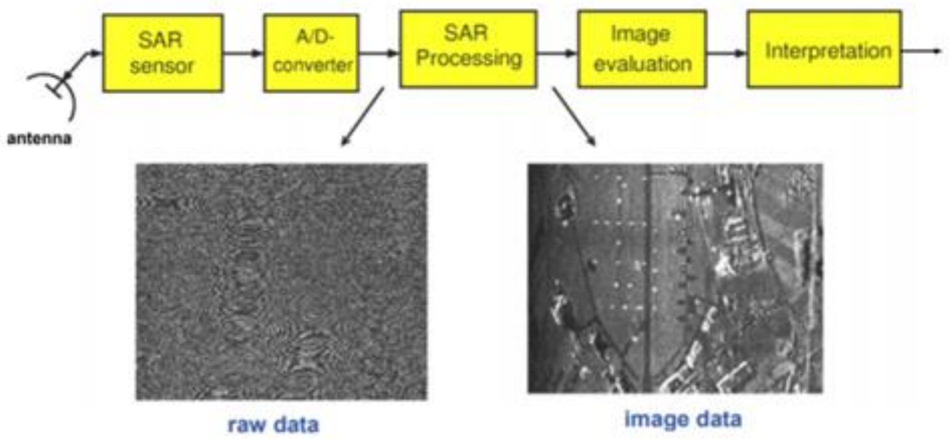


Figure 3 - SAR data flow [9]

Image formation algorithms can be divided in two groups: frequency-domain algorithms and time-domain backprojection algorithms [13][5]. Due to their computational efficiency, frequency-domain algorithms are the most widely used [5][12]. Despite this advantage, assumptions regarding squint, range curve mitigation, the planarity of both the surface and the wavefront and the idealized trajectory of the radar platform, among others, reduce the computation costs but degrade the quality of the image formed [2][5][13]]. Compensation techniques can be applied for deviations from these assumptions, but image quality degrades as the deviation increases [2].

The Backprojection Algorithm is a time-domain algorithm [1] capable of generating images with superior quality than those generated with frequency-domain algorithms, being sometimes referred to as the gold standard in terms of quality for SAR image forming [3][6]. It makes nearly no assumptions and can work in a range of modes and geometries [3][5][8], with its biggest drawback being the high computational cost [1][3]. This algorithm will be described in greater detail in the next section.

2.1.2.1 Backprojection Algorithm

This work uses the Global Backprojection Algorithm[3] as an example to show the generic functioning of these algorithms. The following parameters are needed as inputs [2][5][6]:

- 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 [1][3][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:

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

   $g(n)$ – Radar sample in range bin

   $g(n+1)$ - Radar sample in adjacent range bin after

   $r(n)$ – Corresponding range to bin

   $r(n+1)$ - Corresponding range to adjacent bin after

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

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

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

   x, y, z – Coordinates of the pixel.

   x_k, y_k, z_k – Coordinates of the radar.

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:

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

$f(x,y)$ – Output of Algorithm, final value of pixel (x, y)

θ_k - Aperture point

r_k – Range from pixel (x,y) to aperture point θ_k

ω – Angular velocity of radar waveform

$g_{x,y}(r_k, \theta_k)$ – Reflection received by radar at r_k at θ_k

The C language implementation of the Backprojection Algorithm present in the PERFECT suite [14] differs from the algorithm above in the use of the range from the platform to the centre of the swath [15]. In this implementation it is replaced by R0, representing the minimum distance for which the radar collected samples [15]. Distances inferior to R0 do not have a corresponding sample in the data set.

The pseudocode for this implementation is presented in figure 4:

Algorithm 1 Backprojection pseudocode.

```
 1: for all pixels $k$ do
 2:     $\mathbf{y}_k = 0$
 3:     for all pulses $p$ do
 4:         $R = ||\mathbf{a}_k - \mathbf{v}_p||$   /* Distance from platform to voxel */
 5:         $\mathrm{bin} = \lfloor (R - R_0)/\Delta R \rfloor$   /* Range bin (integer) */
 6:         if $\mathrm{bin} \in [0, N_{BP} - 2]$ then
 7:             $w = (R - R_0)/\Delta R - \mathrm{bin}$
                /* Data sampled using linear interpolation */
 8:             $s = (1 - w) \cdot \mathbf{x}(p, \mathrm{bin}) + w \cdot \mathbf{x}(p, \mathrm{bin} + 1)$
 9:             $\mathbf{y}_k = \mathbf{y}_k + s \cdot e^{j \cdot k_u \cdot R}$
10:         end if
11:     end for
12: end for
```

Figure 4 - Pseudocode of the Backprojection Algorithm [15]

The value *Ku* present in the pseudo-code is defined as $\frac{2\pi f_c}{c}$, where *c* is the speed of light in vacuum and $f_c$is the carrier frequency.

2.1.2.2 *Image Quality Assessment*

The Signal-to-Noise-Ratio (SNR) metric was chosen to evaluate the quality of the output image. The SNR provides the relation between the power of the desired signal and the power of the background noise, expressed in dBs and calculated using the Eq.5 [1][15],

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

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. The noise is defined as the difference between the reference and the result obtained. This reference is provided in the PERFECT suite. Eq.5 is computed by the pseudocode in figure 5, where $test$ represents the output image of the Backprojection algorithm and ref represents the reference image. The re and im fields represent the real and imaginary parts of the value of each pixel.

```
Algorithm 2 SNR calculator pseudocode.
 1: for all pixels $p$ do
 2:     $den \leftarrow den + (ref[p].re - test[p].re)^2$
 3:     $den \leftarrow den + (ref[p].im - test[p].im)^2$
 4:     $num \leftarrow num + (ref[p].re)^2 + (ref[p].im)^2$
 5: end for
 6: if $den == 0$ then
 7:     return $140.0$
 8: else
 9:     return $10.0 \cdot \log_{10} \frac{num}{den}$
10: end if
```

Figure 5 - Pseudocode to compute the SNR of an Image [16]

Large positive numbers of SNR represent a high degree of similarity between the two images. Each 20 dB of SNR correspond roughly to a single digit of accuracy between values [15]. The PERFECT suite manual sets 100 dB as the reasonable correctness threshold [15].

2.2 Reconfigurable Hardware

Reconfigurable computing refers to the use of a computer architecture that combines the flexibility of software with the high performance of very flexible high-speed computing fabrics hardware [1]. The family of devices that best embodies this philosophy results from the fusion of a SoC with a FPGA device. The FPGA part of the device provides the configurable high-speed computing fabric while the SoC architecture guarantees a tight coupling between components, the existence of a microcontroller or microprocessor, the existence of external memories or connectors for external memories and a range of peripheral ports [17].

The most important difference between these devices and regular microprocessors is the ability to make substantial changes to the Datapath [18]. Modern microprocessors are complex, mainly sequential devices designed to be applicable for solving virtually any mathematical problem and as such are not fully optimized to solve simple calculations [18]. The FPGA, on the other hand, allows the Datapath to be designed with only the necessary elements while offering a parallelization-friendly environment [19]. By fusing the two systems a new one is created with a generic CPU and connectors granting high portability and compatibility, tightly coupled with a programmable fabric allowing the design of high-performance modules that perform specific operations considerably faster than the CPU [20].

The Graphics Processing Unit (GPU) based accelerators are usually preferred over their CPU-based counterparts due to their improved throughput, but the high-power consumption of GPU has brought up concerns regarding cooling in data centres [19][21]. When comparing these to a SoC-FPGA device, a considerable difference lies on power consumption [21][20]. This topic has been a case study over the last decade, with some studies reporting an increase in energy efficiency of 75 times in the most favourable situations and an increase of 9.5 times when working in the mode most favourable to the GPU architecture [18][21].

Another type of devices that can offer this granularity in the design of the Datapath are the ASICs. What the ASICs offer in specificity, they lose in flexibility. These devices are built to perform only very specific operations offering no means of re-configuration [20]. FPGA devices have clear advantages in design revisions and development costs by making available a high performance system from the get-go with well-known available resources, unlike the ASICs that must make a lot more decisions regarding the chip composition and production [22].

There are several accelerators for the Backprojection algorithm. Some target High performance systems [23], while others target embedded systems using variants of the algorithm like the works in [24] and [25]. Previous work on implementations of the Backprojection algorithm using a SoC device can be found in [1] and [6]. The work in [6] presents an accelerator, developed with the High-Level-Synthesis tool, to compute the more time consuming operations identified through profiling, using floating-point.

2.2.1 Zybo Z7-10

This work was developed to be implemented in a Zybo Z7-10 board [26] containing a Zynq 7000 device from Xilinx.inc (Califórnia, USA), an external DDR3 memory with capacity for 1GB and I/O peripherals. The Zynq 7000 is composed of a Processing System (PS) and PL block. The main components in the PS are the Dual-core ARM Cortex-A9, containing cores identified as CPU0 and CPU1, the memory controller and the Advanced Microcontroller Bus Architecture (AMBA) Interconnect bus. The ARM processor has two levels of cache, the L1 cache being composed of a 32KB memory for data and another 32KB memory for instructions and the L2 cache having 512KB and being shared between the cores. Additionally, it features a 256KB On-Chip-Memory split into two continuous address spaces, the Random Access Memory (RAM) 0 and 1, with 192KB and 64KB respectively. The ARM processor has a clock frequency of 667MHz and a dedicated Phase-Locked Loop (PLL) capable of generating up to four reference clocks, each with settable frequencies, that can be used to clock custom logic implemented in the PL.

The AMBA Interconnect bus links all the peripherals as slaves to the processor and contains readable/writable control registers (reg) that are addressable in the processors' memory space. The main connections used in this project are the High-Performance (HP) ports and the General Purpose (GP) ports. Although it is not a physical component is important to mention the Advanced eXtensible Interface (AXI) protocol, that is the Intellectual Property (IP) interconnect standard chosen by Xilinx. It provides not only improved performance and flexibility to build custom IP cores, but also compatibility with the Xilinx IP-catalog and a worldwide community of ARM Partners.

The PL is a Xilinx 7-series FPGA. The resources available are shown in table 1.

Table 1 - Zybo Z7-10 PL main resources. [26]

LUT elements	**Flipflops**	**DSP**	**BRAM**	**Slice**
17 600	35 200	80	60 (270KB)	4400

The schematic of the APSoC (All Programmable SoC) is shown in figure 6.

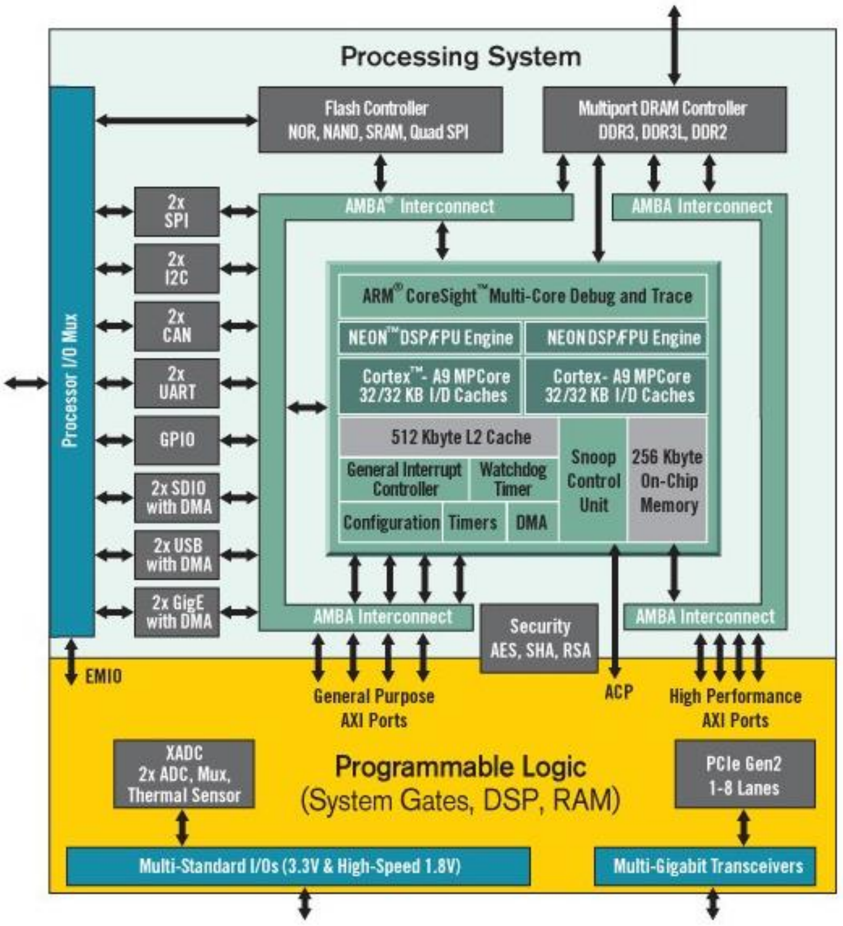

Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

Figure 6 - Overview of the Zynq APSoc Architecture [26]

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

The implementation, that is detailed in chapter 4, uses primarily:

- the Double Data Rate (DDR) memory to store the input files and the final output;
- HP ports with Direct Memory Access (DMA) capabilities and high bandwidth to link the DDR memory to the Hardware Accelerator;
- the GP ports used by the CPU for controlling the AXI-DMA modules that manage the channels between the Hardware Accelerator and the HP ports;
- the CPU for controlling the DDR memory accesses through the control of the AXI-DMA modules;
- the PL fabric to implement the Hardware Accelerator. The PL uses only one clock, from the four references made available by the PS.

2.3 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 [27].

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 [27][28].

In this section both types of numeric formats are presented, followed by a comparison based on the differences observed in the multiplication operation and the impact they have in an FPGA implementation.

2.3.1 *Floating-Point*

The floating-point format stores in a variable three numbers to represent a value. These numbers are unsigned integers and have a specific position in the word, i.e. each value is interpreted by the position it occupies. The size of these numbers (in bits) differs between floating point formats. The floating-point representation is composed by the following fields, in the order presented below from Most Significant Bit (MSB) to the Least Significant Bit (LSB):

- the sign field (0 = positive, 1= negative);
- the exponent field;
- the significant field.

To categorize a floating-point format, it is also necessary to know the base and the bias value used, with 2 and 127 being the values adopted in the Institute of Electrical and Electronics Engineers-754 (IEEE) for the 32-bit single precision floating-point numbers [29]. Figure 7 provides a graphical representation of the standard 32-bit single precision floating-point format.

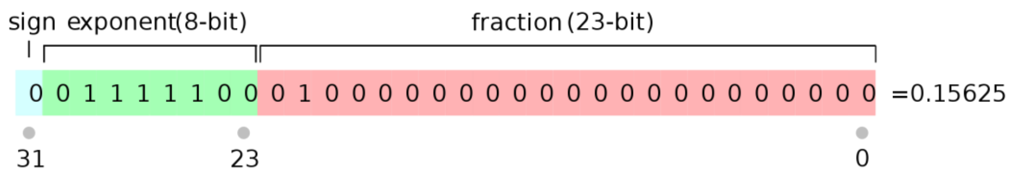


Figure 7 - Word structure of a number in Floating-Point format (https://courses.physics.illinois.edu/cs357/sp2020/notes/ref-4-fp.html)

A value stored in a floating-point format can be converted to a real value by performing the operations in Eq. 6, where σ represents the signal, $\sigma = \pm 1$.

$$X = \sigma * 2^{(exponent-bias)} * significant \tag{6}$$

The bias was introduced to allow the representation of very small numbers, allowing the use of negative powers of two without the need to use a sign bit in the exponent field. Although the floating-point format allows the representation of real numbers, a single floating-point format cannot represent all real numbers in any given interval.

2.3.2 *Fixed-Point*

The Fixed-Point formats store a value in a variable in 2's complement [30]. The name fixed-point stems from the fact that in a word there is a fixed number of bits to represent the integer part of the value and a fixed number of bits to represent the decimal part of the value, creating an imaginary binary-point in the number.

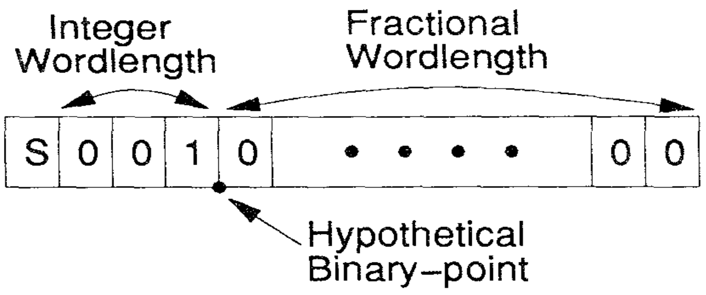


Figure 8 - Word structure of a number in Fixed-Point format [30]

To interpret the true value represented by a fixed-point variable it is necessary to know the scale factor used, as its value will determine the location of the binary-point. The scale factors are usually positive powers of two, enabling the use of bit shift operations as a replacement for multiplications whenever the scale of a variable needs to be adjusted. To store a value in a fixed-point format the following operation is performed:

$$FixP = Value * Scale\ Factor; \tag{7}$$

To interpret the actual value stored in a variable the complementary operation is performed:

$$Value = \frac{FixP}{Scale} Factor; \tag{8}$$

2.3.3 Numeric Format Comparison

This section presents a brief comparison between the two previously described numeric formats. The comparison focuses on the differences between implementations of basic arithmetic operations and the impact they have on hardware implementations. This analysis does not regard precision since the target system chosen for this work supports fixed-point operations with a maximum input size big enough to achieve higher precisions than required.

The multiplication operation will be used to exemplify the differences with greater impact in the choice between the two formats. Figure 9 shows the architecture of a floating-point multiplication which includes chain of operations, starting with the multiplication of the significands and composed of four stages, that is required to maintain the precision of the result. This becomes a critical path in the Datapath.

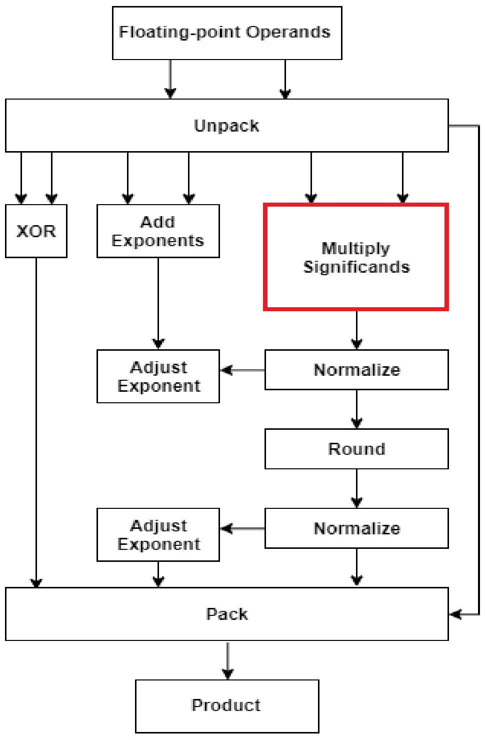


Figure 9 - Floating-Point Multiplication Architecture [29]

While the entirety of figure 9 represents the blocks needed to perform a floating-point multiplication, only the highlighted block is necessary for a fixed-point multiplication. The designer must, however, be mindful of the increase of the scale of the output. It can be concluded that a fixed-point implementation incurs in less complex operations than a floating-point operation, allowing for a greater performance and lower resource allocation. The scale adjustments in the fixed point operations do not pose a constraint because they can be performed by bitwise operations that in hardware can be implemented by splitting a bus or concatenating 2 buses (The architecture “performs” the operations, not the logic cells).

2.4 Approximate Computing Techniques

In this section the methods studied for alternative implementations of mathematical operations are presented in detail. As stated in the introduction of this chapter, 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. These methods can be implemented in hardware and are resource-effective, albeit producing less precise computations than the standard library implementation of these operations in C language.

Due to the context of this work, the methods are studied with two purposes: the first is to choose a method for software implementation; the second is to choose a method for hardware implementation. The software implementation is not essential but can be useful for the initial studies of the algorithm. The performance of the methods varies greatly with the platform were it is being executed and because software and hardware implementations are fundamentally different in nature, methods chosen for each case can be different.

This section is split into two parts, one regarding the square root operation and the other regarding the trigonometric functions.

2.4.1 Square Root

This section presents the methods studied for the fixed-point implementation of the square root operation.

2.4.1.1. Newton-Raphson

Named after Isaac Newton and Joseph Raphson, the Newton Raphson method is an iterative root-finding algorithm, meaning it is used to solve algebraic equations of the form [31]:

$$F(x) = 0 \tag{9}$$

The method needs an initial guess to produce the first iteration while the successive iterations are calculated using the eq.10:

$$X_{n+1} = X_n - \frac{F(X_n)}{F'(X_n)} \tag{10}$$

The definition of $F(X_n)$ chosen to approximate the square root function of Y using this method is presented in the following equations:

$$F(X) = Y - X^2 \tag{11}$$

This definition yields the following derivative in X:

$$F'(X) = -2X \tag{12}$$

Using eq.11 and 12, the adapted formula for the method results in:

$$X_{n+1} = X_n + \frac{Y - {X_n}^2}{2X_n} \tag{13}$$

The resulting formula can now be optimized in terms of the operation composition, to reduce the costs of a hardware implementation (Some optimizations are fixed point exclusive). By application of the distributive property on the division and subtraction, the of eq.13 can be changed to eq.14:

$$X_{n+1} = X_n + \left(\frac{Y}{X_n} - X_n\right)/2 \tag{14}$$

The new form produces the same output but has the advantage of performing no multiplications (1 less than the original formula) and of replacing the division by 2 with an arithmetic shift left (Due to the use of the fixed point format).

The stop criteria used for the method is defined in eq.15:

$$\frac{-F(X_n)}{F'(X_n)} > 0 \tag{15}$$

2.4.1.2 Binary restoring square root extraction

The binary restoring square root extraction algorithm finds the square root x of a nonnegative integer number, R, satisfying the following constraints [32]:

$$x^2 <= R \ AND \ (x+1)^2 > R \tag{16}$$

This method takes 2 bits of the Radicand to produce 1 bit of the result in each iteration, resulting in a complexity of O(*n/2)*, with n being the size in bits of the radicand [33]. Below is presented a C language implementation of this algorithm [32]:

```
#define FRACBITS 30;
#define ITERS (15 + (FRACBITS >> 1));
typedef long TFract; /* 2 integer bits, 30 fractional bits */
TFract
FFracSqrt(TFract x)
{
        register unsigned long root, remHi, remLo, testDiv, count;
        root = 0; /* Clear root */
        remHi = 0; /* Clear high part of partial remainder */
        remLo = x; /* Get argument into low part of partial remainder */
        count = 30; /* Load loop counter */
        do {
                remHi = (remHi<<2) | (remLo>>30); remLo <<= 2; /* get 2 bits of
        arg */
                root <<= 1; /* Get ready for the next bit in the root */
                testDiv = (root << 1) + 1; /* Test radical */
                if (remHi >= testDiv) {
                        remHi -= testDiv;
                        root++;
                }
        } while (count-- != 0);
        return(root);
}
```

From the code presented it can be deduced that this algorithm does not use any multiplication or division and has a simple loop composed only of shift-add operations that execute quickly in software and especially in digital hardware, making it a competitive solution for the computation of the square root.

2.4.2 Trigonometric Functions

This section presents the algorithms studied for the fixed-point implementation of the trigonometric functions.

2.4.2.1 CORDIC

The Coordinate Rotation Digital Computer algorithm (CORDIC) is a hardware-efficient iterative method especially suitable for solving trigonometric functions and hyperbolic functions [34]. CORDIC's main concept is the usage of a series of optimized rotations to achieve the input rotation. The great advantage of this algorithm is that it belongs to the shift-add family, meaning it does not need multipliers to be implemented. The CORDIC's main working principles are explained in this section.

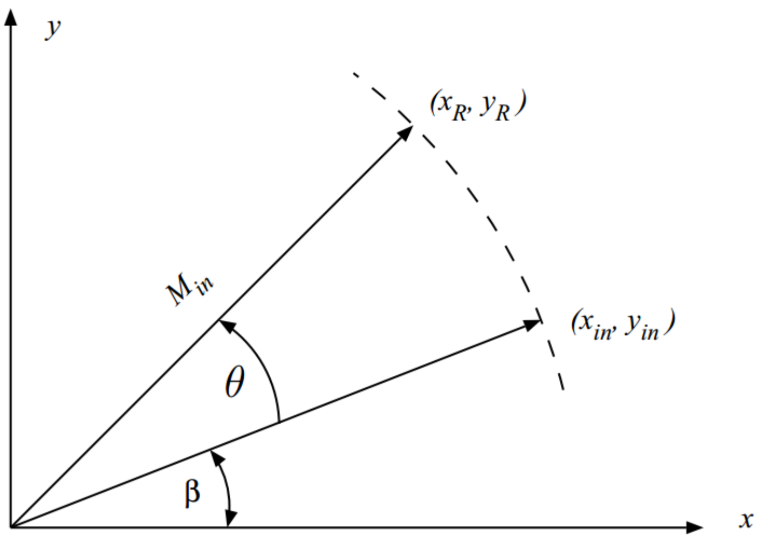


Figure 10 - Rotation of a vector by an arbitrary angle

(https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/)

A rotation around the origin in the 2-D plane (shown in figure 10) can be defined by eq.17 and 18:

$$X_R = X_{in} * cos(\theta) - Y_{in} * sin\ (\theta) \tag{17}$$

$$Y_R = X_{in} * sin(\theta) + Y_{in} * cos\ (\theta) \tag{18}$$

Re-writing eq.17 and 18 into the matrixial form yields eq.19:

$$\begin{bmatrix} X_R \\ Y_R \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin\ (\theta) \\ sin\ (\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} X_{in} \\ Y_{in} \end{bmatrix} \tag{19}$$

Eq.19 can be simplified into eq.20.

$$\begin{bmatrix} X_R \\ Y_R \end{bmatrix} = cos(\theta) \begin{bmatrix} 1 & -tan\ (\theta) \\ tan\ (\theta) & 1 \end{bmatrix} \begin{bmatrix} X_{in} \\ Y_{in} \end{bmatrix} \tag{20}$$

There are two fundamental ideas behind the CORDIC's implementations. The first is that rotating a vector by an arbitrary angle θ is the same as rotating the same vector by several smaller angles; the second is that it is possible to compute the rotations by substituting the multiplications by $\tan(\theta)$ with simple bitwise shift operations, by choosing these smaller angles in a way that eq.21 is satisfied,

$$\tan(\theta_i) = 2^{-X} \tag{21}$$

where X is a positive integer. A CORDIC interpretation of an angle composed by the sum of 3 smaller angles is explained in eq.22:

$$\begin{bmatrix} X_R \\ Y_R \end{bmatrix} = cos(\theta_0)cos\,(\theta_1)\cos\,(\theta_2) \begin{bmatrix} 1 & -2^0 \\ 2^0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{bmatrix} \begin{bmatrix} X_{in} \\ Y_{in} \end{bmatrix} \tag{22}$$

The only multiplication left to perform can be ignored until the end of the process because the product defined by eq.23 converges to the value in eq.24:

$$K(n) = \prod_{k=0}^{n-1} \cos\,(\theta_k) \tag{23}$$

$$\lim_{n\to\infty} K(n) = 0.60725293500 \tag{24}$$

Leaving the iterations of the algorithm to be defined by eq.25:

$$\begin{bmatrix} X[i+1] \\ Y[i+1] \end{bmatrix} = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} X[i] \\ Y[i] \end{bmatrix} \tag{25}$$

Where σ_i represents the direction of each rotation, $\sigma_i = \pm 1$. As stated before, it is still necessary to perform the final multiplication by the factor K.

2.4.2.2 Taylor Series

The Taylor Series is a method of approximating functions through an infinite sum of terms that are expressed in terms of the function's derivatives at a single point. The Taylor Series of a function f(t) around point t=tr can be defined by the equations 26 and 27 [35]:

$$f(t) = \sum_{i=0}^{\infty} \frac{f^{(k)}(t_r)}{i!} * (t - t_r) \tag{26}$$

$$f^{(k)}(t) = \left(\frac{\partial^{(k)} f(t)}{\partial t^{(k)}} \right) \tag{27}$$

The Taylor series applied to the trigonometric functions cosine and sine takes the forms in eq. 28 and 29 respectively:

$$\cos(x) = \sum_{n=0}^{\infty} (-1)^n * \frac{x^{2n}}{(2n)!} \tag{28}$$

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n * \frac{x^{2n+1}}{(2n+1)!} \tag{29}$$

Although this method can produce precise results, the need for multiplications makes it a less ideal candidate for implementation in FPGA fabric when CORDIC can produce results with the same precision [1], without the use of multiplications.

3 Optimization Methodology

The design of the optimized Hw/Sw system begins with the study of the algorithm as along with the environment of the new implementation to understand what new tools and functionalities are available and how and where they can be used.

This chapter presents the optimization methodology, composed by the studies of the C language implementation of the Backprojection Algorithm, the studies of the target environment and the design decisions they support.

The chapter begins with the profiling study followed by the Hw/Sw partition. The word-length study is then presented, followed by the hardware resource utilization study guided by the previous study's conclusions. This chapter concludes with the algorithm's re-scheduling that aims to further improve the optimized system's architecture.

3.1 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. To this end a profiling study was developed to quantify the time resources required by each operation.

This section presents the profiling of the C language implementation of the algorithm provided in the PERFECT suite [15]. The first step of this study was performed in a PC to obtain an initial estimate using the already fully functional available materials. Table 2 details the main specifications of the PC used along with relevant configuration options.

Table 2 - Computer Specifications and Configuration

CPU	Operating system	Multi-Threading	Compilation flags
Intel I7-7700K	Ubuntu 20.04.1	No	default

This provided a basis to compare the forthcoming results from the algorithm execution in the target system's CPU. Table 3 details the CPU present in the target device, along with the relevant configuration options.

Table 3 - Target device Specifications and Configuration

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

The profiling study of the implementation using the PC was made using the gprof tool available in the Ubuntu 20.04.1 LTS release[1]. The profiling was made using an image set that has the sample data of 512 pulses and produces an output image of 512 x 512 pixels.

The next step of the process is the profiling of the algorithm's execution in the target system. The C code used in this stage is the same as the one used in the PC, except for adaptations made regarding the reading operations of the input files. Profiling in the target system was performed using the xil_time.h library functions. The results obtained are detailed in table 4.

Table 4 - Profiling results of the C language implementation of the Backprojection Algorithm

Algorithm Operation	Desktop Computer (-O3)	Zybo Z7-10 (-O3)
Pythagorean Theorem	0.00425 us	0.12 us
Pow ^2	0.00365 us	0.04 us
Square Root	0.00059 us	0.06 us
Cos & Sin	0.00231 us	3.40 us
Complex Multiplication	0.00618 us	0.06 us
Total	6.97 s	480,729,250.04 us (481 s)

The differences observed between the profiling results obtained in the PC and in the target system can be explained by the different architectures used to manufacture the CPU of the two systems. From this point onward the results obtained regarding the target system will be used as reference for future comparisons.

[1] https://ubuntu.com/download/desktop

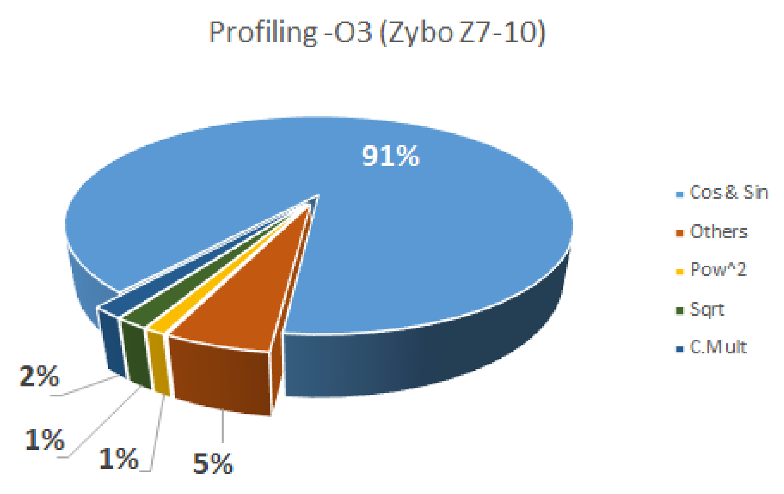


Figure 11 - Graphic representation of the profiling results of the zybo board from table 4

From analysis of the results, 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 [1][2][6].

A first proposition for the optimized system architecture that can emerge from the conclusion explained above 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 (instructions comprised in the calculations of a single pulse's contribution to a single pixel) has too many instructions to allow a serial execution in the CPU to be efficient, even if we consider the Cos and Sin execution time is 0 and that the CPU's clock is 4 to 5 times faster than the fastest clock that a circuit implemented in the PL can achieve (CPU frequency = 667MHz; PL frequency = 150MHz). In light of these considerations, it becomes easy to understand that the goal of a total execution time of one second requires the system's architecture to take maximum advantage possible from the pipelining and parallelization opportunities made available by the PL system.

3.2 Hardware/Software Partition

The target's system family devices were presented in this work as the fusion of two independent systems. This section is focused on the partition of the system's execution between the PS and PL systems.

To achieve the initial goal of total execution time below the one second mark, the system needs to take advantage of the reconfigurable hardware's characteristics in terms of pipelining and parallelization. In this case it means that most, if not all, of the algorithm should be executed in the PL because of the limitations of the PS system when handling the serial execution of a program with many cycles (134,217,728 cycles in this case). This idea is further supported by the work in [6], that registered a speedup of 7.7x, achieved by implementing the more time consuming operations in reconfigurable hardware.

A system partition is proposed, based on the observations made in the previous paragraph, with the following characteristics:

Table 5 - Hardware / Software partition

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

This proposal is made assuming the following:

- Due to the size of the input data files, they are initially stored in the DDR memory;
- The hardware accelerator will access the DDR memory in DMA mode, through the HP ports (using a well-known xilinx IP core, the AXI-DMA);
- The PL and PS will also communicate using the GP ports.

Figure 12 illustrates the proposed partition.

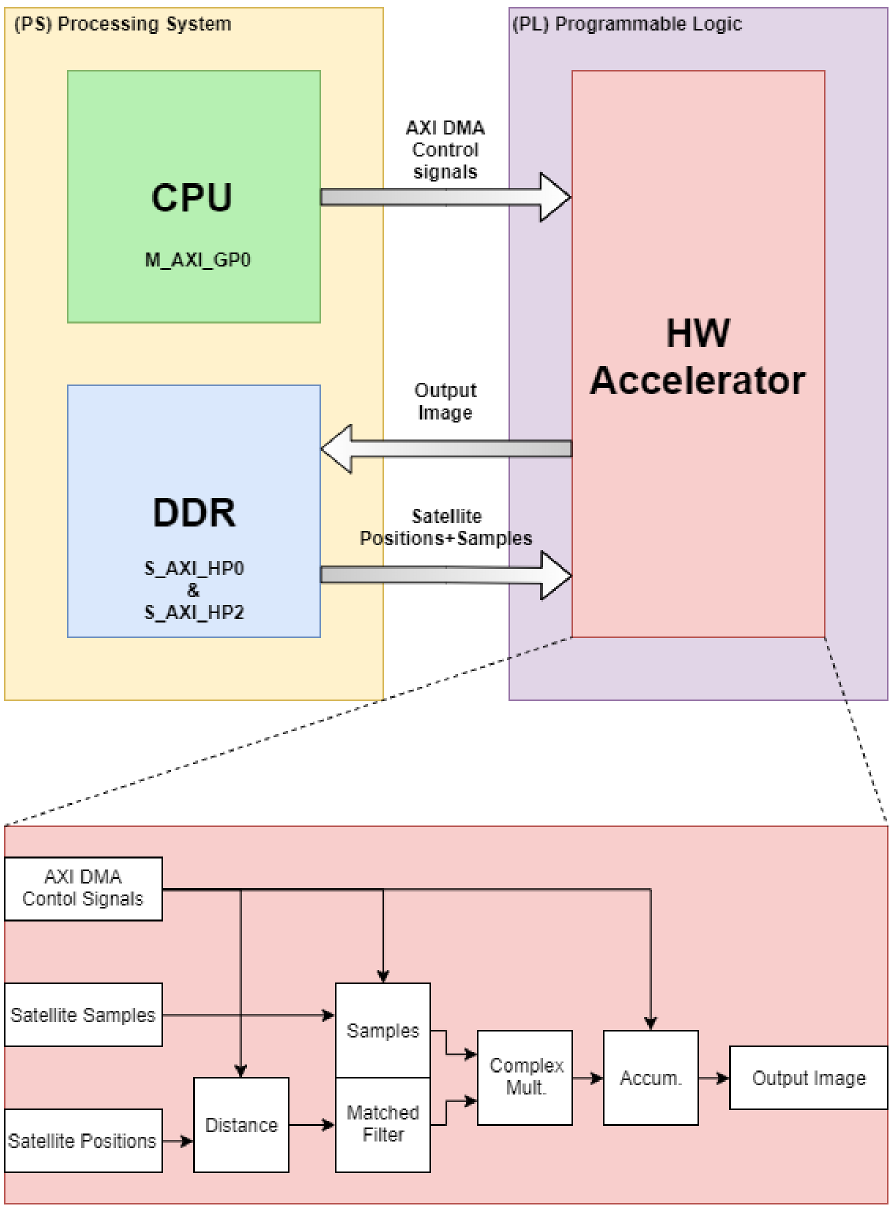


Figure 12 - Overview of the proposed system's partition

With this architecture the optimization can go much further than the first proposal made in the previous section in terms of execution time, because now we can 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:

$$Execution\ Time\ = (\ Nº\ Pipeline\ stages\ +\ Nº\ Cycles)\ /\ Frequency \qquad (30)$$

$$Execution\ Time\ = \ (0^{?}\ + 134\ 217\ 728\ )/\ x = 1s \qquad (31)$$

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 consider it almost zero, because the number will always be small when compared with the number of iterations in the algorithm.

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, which are addressed in sections 3.4 and 3.5.

This is only a proposal, as it is still necessary to evaluate its feasibility in terms of resources used and the performance of these resources (not all resources all fully customizable).

3.3 Word-Length Optimization

To design an optimized circuit, it is fundamental to choose the right size for its signals as this decision has a great impact over the resources allocated and in achieving the timing requirements of the circuit.

This section presents the word-length study made on the Backprojection Algorithm for implementation of the algorithm using the fixed-point numeric format. This study has two parts: the first aims to provide the length, in bits, that the integer part of every signal needs in order to accommodate its biggest possible absolute value avoiding overflow; the second part aims to provide the length that the decimal part of every signal needs in order to attain precision that will guarantee the overall quality performance of the system of 100dB of SNR [28].

The first part begins by executing the original program 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, that will be able to hold every value needed and avoid overflow of operations. These results do not consider the bit used for the signal because this is inferred case by case.

The requirements for the word lengths found by this study are presented in the tables in appendix A.

To perform the second part of this study 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.

The new implementation was developed taking into account that after the loss of precision occurring in the conversion of the input values to the fixed-point format, no other loss of precision is present (meaning all significant bits of the output values of mathematical operations are carried forward). 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).

Some operations don't have an implementation in the standard libraries for operators bigger than 64 bits and some operations like the square root and the Cosine and Sine don’t have an implementation in fixed point. As such, new implementations were made for those cases, implementing the square root and trigonometric functions using the chosen methods among those detailed in section 2.4.2. Although the mathematical operations do not cause loss of precision, the method used for the trigonometric functions does not guarantee precision to the LSB. 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 with the help of the results of the first part of this study while the decimal part was chosen to occupy the remaining bits. The different sizes varied in a 1-byte interval starting at 24 bits and going up to 64. Table 6 shows the SNR (in dB) obtained for each combination. The size of the variables from the Data group vary between lines and the size of the variables from the Distance group vary between columns.

Table 6 - SNR (dB) obtained for each word-length combination studied

Data/Distance	**64**	**56**	**48**	**40**	**32**	**24**
64	182.876	183.976	151.595	103.129	54.967	9.252
56	182.876	183.976	151.595	103.129	54.967	9.252
48	182.876	183.976	151.595	103.129	54.967	9.252
40	182.819	183.904	151.595	103.129	54.967	9.252
32	153.005	153.007	149.326	103.129	54.967	9.252
24	103.609	103.609	103.610	**100.469**	54.965	9.252

From the results in table 6, it can be concluded that to achieve the quality minimum of 100dB of SNR while minimizing resource utilization, the best size combination is:

- Distance - 40 bits.
- Data/Filter - 24 bits.

This combination achieves an SNR of 100.469dB and results in the following fixed-point formats:

- Distance [39:0] - 15Q25 (signed).
- Data/Filter [23:0] - 2Q22 (signed).

To further corroborate the conclusion drawn here, this part of the study was extended by conducting an extra experiment with the chosen sizes and the chosen sizes minus 1 bit. The results yielded by this experiment in terms of SNR (in dB) are presented in table 7.

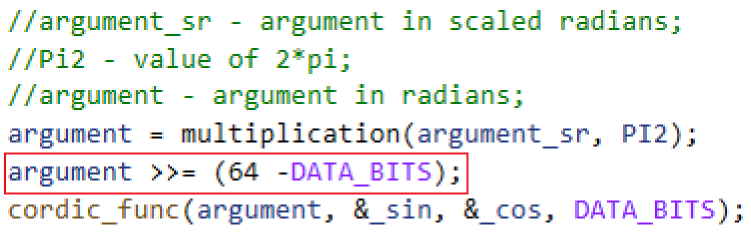
Table 7 - SNR (dB) obtained for the word-length combinations closest to the threshold

Data/Distance	**40**	**39**
24	**<u>100.469</u>**	96.359
23	96.543	94.401

From these results it is concluded that the best possible combination of formats that can be found while using the methodology followed in this work is the previously elected one, highlighted in tables 6 and 7.

It should be noted that in the implementation developed for the second part of this study some operations incurred in precision loss. This loss occurs in the calculation of the argument for the trigonometric operations, where a right shift is executed on the output of a multiplication that converts a value in scaled radians to a value in radians. This operation is performed to reduce the bit width necessary to represent the value of the argument. The CORDIC algorithm performs an iteration for each bit of the input, therefore reducing the input size reduces the number of iterations needed as well.

Figure 13 presents the code that performs the detailed operations, with the operation responsible for the precision loss highlighted.



```
//argument_sr - argument in scaled radians;
//Pi2 - value of 2*pi;
//argument - argument in radians;
argument = multiplication(argument_sr, PI2);
argument >>= (64 -DATA_BITS);
cordic_func(argument, &_sin, &_cos, DATA_BITS);
```

Figure 13 - Snippet of code responsible for performing the error inducing operation.

From the results presented in table 6 it can be concluded that, in the ranges considered for the bit width of the signals, the width of the Distance group variables has a greater impact on the output image quality than the width of the Data group variables. This conclusion further supports the choice of a precise method for the implementation of the square root operation: the Binary restoring square root extraction.

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. The full list of the fixed points formats for each signal in the circuit will be shown in the next section 4.1.2.

3.4 Resource Utilization Study

The Zybo Z7-10 board used in this work has limited resources and the use of those resources adds complexity to the circuit and has an associated power consumption. From an optimization perspective, the possibility of freeing unnecessary resources that could otherwise have a very negative impact in the efforts to meet the timing constraints offers room for improvements. 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 make estimates on the values of the resources allocated by the future design. Although this work was developed with a specific target system in mind, the results from this type of study can inform the choice of a suited device when working in projects that start with an undefined target system.

The study was developed by making a synthesized design in Vivado 2019.2 of the main IP cores used, with varying operator sizes, varying pipeline stages and with the clock frequency set to 140MHz (estimate frequency needed to achieve final timing goal). The variations in pipeline stages only occur until the design meets the timing requirements. The results are presented in the tables in appendix B.

The number of IP cores needed to implement the circuit that performs the mathematical operations described in the C language implementation of the Backprojection algorithm can be estimated to further assess the total resources of the top-level modules.

Since the use of extra IP cores and their configurations in this project was expected, some were studied to get better estimates. The referred IP cores are the AXI DMA, the AXI Interconnect and the AXI Data FIFO. Table 8 shows the resources used in the implementation of these support IP cores.

*Table 8 - Resources used by support IP cores**

IP core	**Quantity**	**Unit Cost**	**Total Cost**
AXI DMA	2	LUT: 1954 Reg: 2570 F7 MUX:5 BRAM 36Kb: 3	LUT: 3908 Reg: 5140 F7 MUX: 10 BRAM 36Kb: 6
AXI Interconnect (HP ports)	2	LUT: 684 Reg: 783	LUT: 1368 Reg: 1566
AXI Interconnect (GP port)	1	LUT: 551 Reg: 694	LUT: 551 Reg: 694
AXI Data FIFO	2	LUT: 61 Reg: 89 BRAM 36Kb: 4	LUT: 122 Reg: 178 BRAM 36Kb: 8

* The configurations of these IP cores are detailed in chapter 4.

With the information needed to make the circuit estimates gathered, two estimations were, one for the circuit made using the formats chosen in the previous section (Estimate A) and other for a hypothetical circuit made based on 64 bit wide input variables (Estimate B). The estimates include only the circuit responsible for performing the mathematical operations of the Backprojection algorithm, excluding control, support and memory related IP cores. The estimates also exclude the square root operation due to the fact that this operation is implemented by a custom module with fixed input size. Tables 9 and 11 compile all the basic arithmetic IP cores that need to be used to compute the algorithm

Estimate A

Table 9 - Breakdown of resource estimate for system with input variables of width of 64 bits

IP core	Signal + Operator size [inA,inB,Out]	Quantity
Subtractor	S + [64,64,64]	3
Multiplier	S + [64,64,128]	3
Adder	U + [128,128,128]	2
Adder	U + [14,1,14]	1
Multiplier	S + [64,64,128]	4
Adder	S + [64,64,64]	2
Subtractor	U + [64,64,64]	2
Multiplier	U + [20,6,26]	1
Multiplier	U + [64,64,128]	2
CORDIC	S + [48,96]	1
Multiplier	S + [64,64,128]	2
Multiplier	S [64,64,128]	4
Subtractor	S + [128,128,128]	1
Adder	S + [128,128,128]	1
Adder	S + [128,128,128]	2

Table 10 - Resources required by Estimate A

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

Estimate B

Table 11 - Breakdown of resource estimate for system with input variables of optimal width

IP core	Signal + Operator size[inA,inB,Out]	Quantity
Subtractor	S + [40,40,40]	3
Multiplier	S + [40,40,80]	3
Adder	U + [79,79,79]	2
Adder	U + [14,1,14]	1
Multiplier	S + [24,24,48]	4
Adder	S + [24,24,24]	2
Subtractor	S + [40,40,40]	1
Multiplier	U + [20,6,26]	1
Subtractor	U + [26,25,26]	1
Multiplier	U + [64,64,128]	2
Cordic	S + [25,24]	1
Multiplier	S + [24,24,48]	2
Multiplier	S [24,24,48]	4
Subtractor	S + [48,48,48]	1
Adder	S + [48,48,48]	1
Adder	S + [64,48,64]	2

Table 12 - Resources required by Estimate B

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

As per tables 9 and 11, 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.

Following the discussion about the Hw/Sw partition proposed in section 3.2 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. Before values are calculated, a theoretical analysis can offer information to inform decision-making regarding memory types. As mentioned earlier, 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 highlight the importance of this restriction, it can be inferred by equations 30 and 31 that a simple increase in latency of 1 cycle would cause the execution time to almost duplicate. 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 13 shows the estimated memory capacity required to implement the algorithm with the original schedule using the Hw/Sw partition proposed earlier and the word lengths chosen in section 3.3.

Table 13 - Estimate of memory capacity required to store all input values in BRAMs in the PL system

Input Type	Nº of values	Nº of sets	Bit width	Packaging	Word Size Used (bits)	Word Depth	Total (Kbits)
Satellite Positions	3	512	40	1	64	1536	98
Satellite Samples	2	4096 x 512	24	2*	64	2,097,152	131,072

*The real part and imaginary part are packed together.

Table 14 - Estimate of memory capacity required to store all output values in BRAMs in the PL system

Output Type	Nº of values	Bit width	Packaging	Word Size Used (bits)	Word Depth	Total (bits)
Output Image	2	59	1	64	2	128

Converting the values of tables 13 and 14 to resources of the target system, table 15 is obtained.

Table 15 - Estimate of BRAM cells required to store all input/output values

Data	18Kb BRAM cells	36Kb BRAM cells
Satellite Position:	0	3
X	0	1
Y	0	1
Z	0	1
Satellite Samples	0	3648
Output Image	0, registers can be used;	0
Board Resources	120**	60

**The board has 60 tiles of 36Kb BRAMs and each can be split into two 18Kb BRAMs for more flexibility. The total size of the BRAMs in the board is 270KB (36Kb *60 = 270Kb), not the sum of the types of BRAMs.

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.

3.5 Algorithm Rescheduling

Due to the characteristics of the Backprojection algorihm, several schedules can be used when implementing it. This section presents the study of the schedule used in the original software implementation, as well as two new schedules. This study was motivated by the results presented in table 15, showing that the original schedule cannot be implemented in the chosen target system when considering the Hw/Sw partition proposed in section 3.2. To overcome this limitation, a combination of memory management and algorithm re-scheduling was used. As mentioned previously, the cycles of the Backprojection algorithm are independent from one another, so the outcome is not affected by the order in which they are performed. These characteristic increases flexibility in the re-scheduling, given that the accumulation phase is the only part where the dependency between cycles becomes relevant. To this end 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 16.

Table 16 - Pseudocode of the loop hierarchy for the schedules studied

Schedule	Pseudo Code
Pixel Computation	For every **Pixel** do: For every **Pulse** do:
Pulse Computation	For every **Pulse** do: For every **Pixel** do:
Pixel Region Computation	For every **Region** do: For every **Pixel** in the **Region** do: For every **Pulse** do:

Table 17 presents an overall analysis of these schedules.

Table 17 - Overview of the schedules' requirements and implementation details

	Schedules		
	Pixel Computation	Pulse Computation	Pixel Region Computation
Input Requirements	Every cycle: -X, Y and Z values for the new satellite position; -Real and imaginary parts of every sample from the next pulse;	Every 262,144 cycles: -X, Y and Z values for the new satellite position; -Real and imaginary parts of every sample from the next pulse;	Every REGION SIZE cycles: -X,Y and Z values for the new satellite position; -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: -2 values to be stored in the DDR;	End of Execution: -524,288 values to store in the DDR;	Every REGION SIZE * 512 cycles: -REGION SIZE values to be stored in the DDR;
Advantages	- Output: very low throughput; is the final value for each pixel. - Bandwidth requirements for storing	-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.

	output values: low; achievable with the Zybo board's resources		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. -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

It is important to mention that any of the three approaches could result in an optimized system if the constraints in timing and latency imposed thus far were more flexible.

Due to the constraints applied, the third approach proved to be the only option viable.

4 Proposed System Architecture

In this chapter the proposed architecture for Hardware Accelerator implemented in the PL system is presented and explained in detail. The presentation is focused on the IP cores and their configurations along with small but important implementation details that represent some finer optimizations performed and decisions made regarding the control system employed in this work. The circuit was designed in Vivado due to the program belonging to the same manufacturer as the FPGA device used, which makes it the less error-prone option. 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 ILA was used both because the test bench to simulate the whole system would be too complex and because the ILA provides a direct reading of the values passing through the circuit instead of a theoretical simulation (where the effects of the chip temperature and overall degradation of the components cannot be accounted for).

The architecture was divided by function into three parts to facilitate the comprehension of each part and the whole system architecture, schematically represented in figure14.

A. Memory (Purple box + Pink box)
B. Algorithm Execution (Orange box)
C. System Control (light purple box)

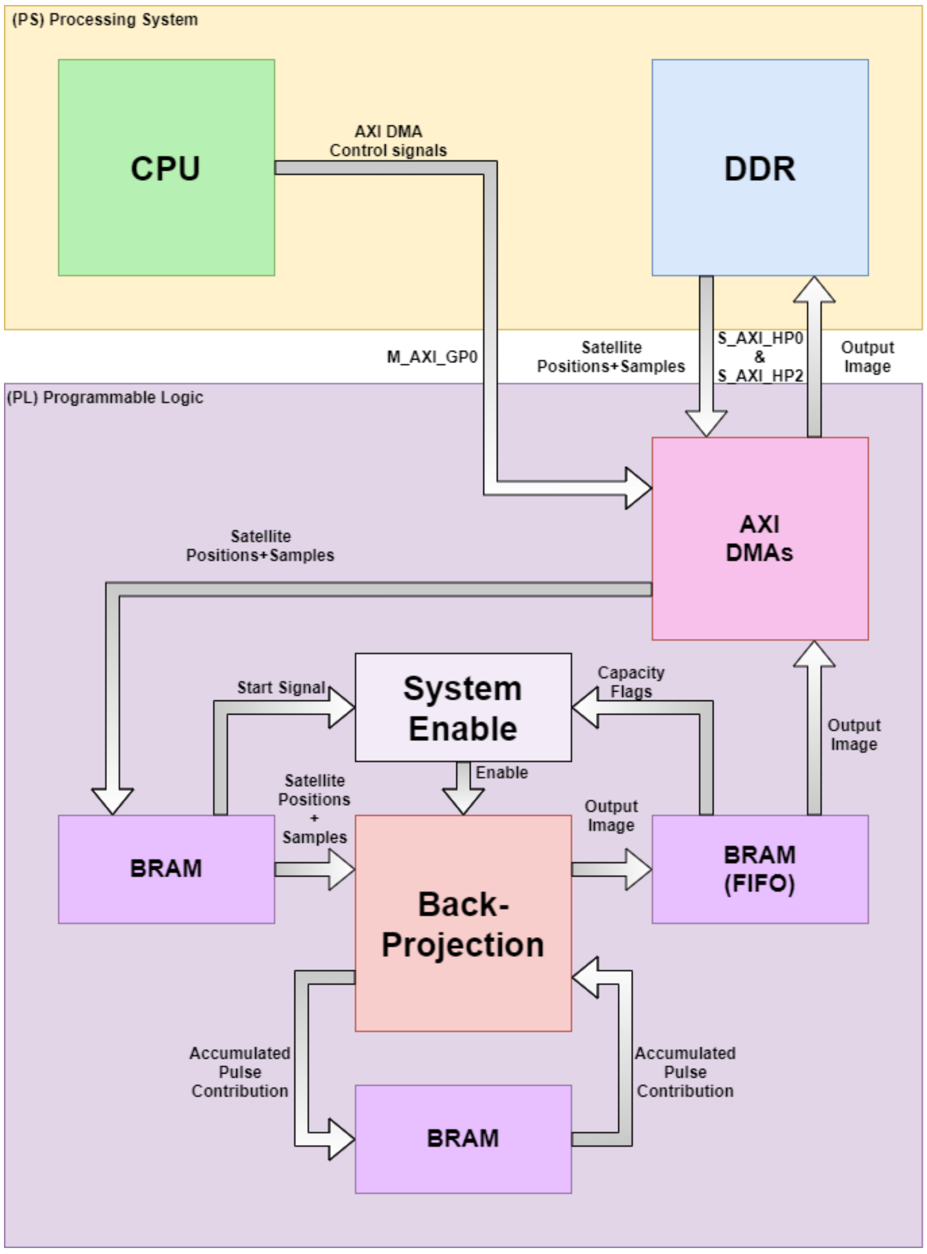


Figure 14 - Proposed system architecture, discriminating memory units and inter-system communication.

4.1 Memory

This section presents in detail all the memory related modules used in the Hardware accelerator, divided into two parts, the first presenting the memory units used and the second explaining the modules responsible for accessing those units. This explanation will be further divided into the considerations about the DDR and the considerations about the BRAMs.

As stated in section 3.5, the algorithm's execution will follow a different schedule than the original implementation due to resource limitations. After designing part of the circuit according to the new schedule, decisions were made regarding the memory units and the size of the region to use:

- The circuit will feature BRAM modules[36], that will store all the satellite positions (one for each coordinate);
- The region size is set to 8192 (16 * 512);
- The output FIFO's size will be decided based on the BRAM cells left unused in the device [37].

Although the chosen schedule's requirement on receiving new satellite positions enables the use of methods that do not need BRAM cells, such as querying the CPU through GP ports and AXI protocols, the use of the BRAMs can make the circuit more easily implemented by improving synchronization. This was achieved by filling the BRAMs with all the positions needed before the algorithm starts execution, keeping the circuit's execution more organized.

The region size used in the implementation of the circuit is 8192, that corresponds to 16 rows out of the 512 in the final output image. This decision comes with the following consequences:

- There are 32 regions in an image, meaning that during execution all the samples will be read from the DDR and stored in the BRAMs 32 times;
- The BRAMs holding the positions of the satellite will be fully read 32 times;
- The BRAMs in the accumulators will need to hold 8192 words;
- The AXI DMAs [38] have 8192 clock cycles to update the BRAMs that hold the samples with 4096 values.

With this analysis the decisions needed to design the memory units can be made.

4.1.1 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 variable types used, along with the sizes of the files, are detailed in table 18.

Table 18 - Input files format and size

File	Format (bits)	Size in KBytes (Hex)
Satellite Positions	int (64)	12 (0x000C)
Satellite Samples	int (32)	16 384 (0x4000)
Output Image	int (64)	4096 (0x1000)

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 19 presents the configurations for these modules along with their contents.

Table 19 - BRAM modules configurations

ID	Content	Format	Word Size	Word Depth	Mode	36Kb BRAM cells
B1.1	X coordinate (position)	1 value per word	64	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]	64	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	64	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	64	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.

4.1.2 Memory Accesses

DDR

As stated in section 3.2, the design of the circuit was made based on some initial decisions. One of them was to use the AXI DMA IP core to coordinate the accesses made by the circuit to the DDR memory, based on the high performance achievable using the HP ports and DMA feature. This core was used due to being a reliable module and the standard module for this type of memory accesses. Designing an alternative module is out of the scope of this work.

The AXI DMA features a writing channel and a reading channel with the following interfaces:

- M_AXI_MM2S, used to read value from the DDR;
- M_AXI_S2MM, used to write values to the DDR;

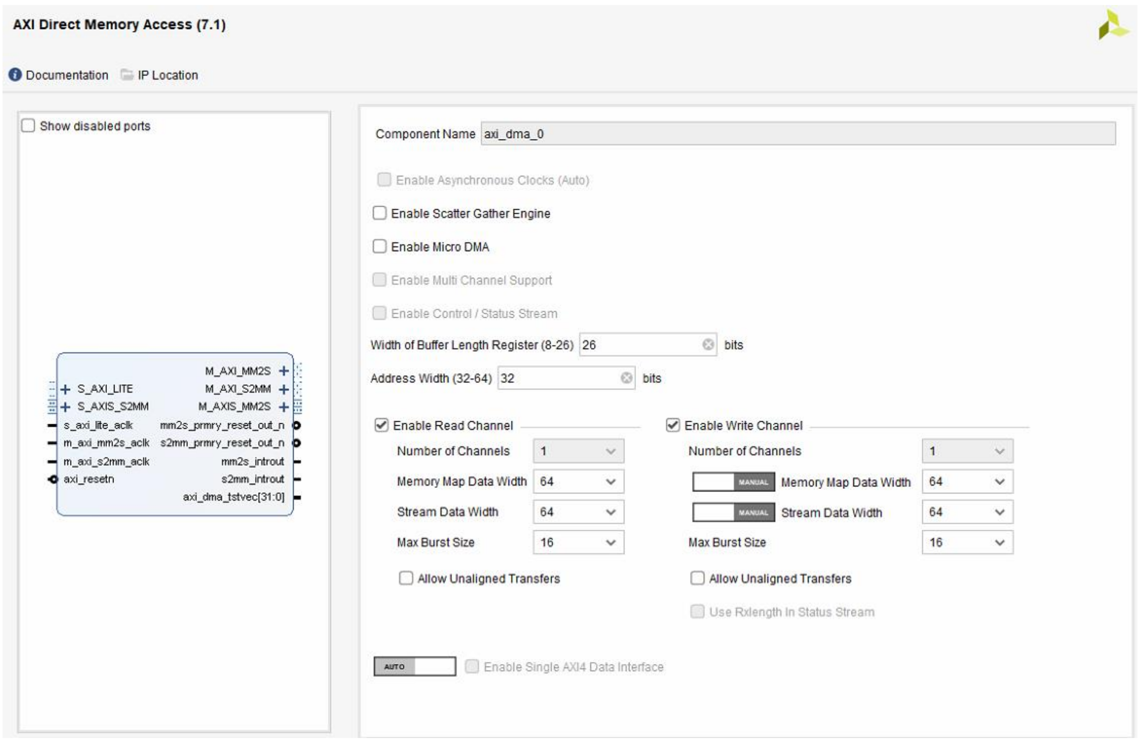
- M_AXIS_MM2S, used to send values to the circuit;
- S_AXIS_S2MM, used to receive values from the circuit;
- S_AXI_LITE, used by the CPU to communicate orders to the module.

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. This protocol is the lightest protocol of the AXI standard in terms of number of channels and protocol messages, making it easy to implement the circuit to which the AXI DMA is connected (the protocol doesn't use addresses, this is communicated to the DMA by the CPU).

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.

Figure 15 shows the configurations of the AXI DMA modules. The values for Max Burst Size are the program's defaults.



AXI Direct Memory Access (7.1)

☐ Show disabled ports

Component Name: axi_dma_0

☐ Enable Asynchronous Clocks (Auto)
☐ Enable Scatter Gather Engine
☐ Enable Micro DMA
☐ Enable Multi Channel Support
☐ Enable Control / Status Stream

Width of Buffer Length Register (8-26): 26 bits
Address Width (32-64): 32 bits

☑ Enable Read Channel		☑ Enable Write Channel	
Number of Channels	1	Number of Channels	1
Memory Map Data Width	64	Memory Map Data Width	64
Stream Data Width	64	Stream Data Width	64
Max Burst Size	16	Max Burst Size	16
☐ Allow Unaligned Transfers		☐ Allow Unaligned Transfers	
		☐ Use Rxlength In Status Stream	

☐ Enable Single AXI4 Data Interface

Figure 15 - AXI DMA IP core configuration in Vivado Environment

BRAMS

All the BRAM modules 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:

B1 - memory module that holds the satellite position values;

B2 - memory module that holds the satellite sample values;

B3 - memory module that holds the intermediate values of the accumulation;

B4 - Output FIFOs.

B1

The B1 controller is responsible for controlling the writing operations on the B1.1, B1.2 and B1.3 memories, featuring a memory controller interface and a Slave 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.

It waits for the signal from the AXI DMA to start receiving the coordinate values, which are then sent one coordinate at a time so that the memories are filled sequentially, facilitating the design of the module.

The reading operations performed on these memories are controlled by a counter used for addressing. This counter is part of the Pixel Position module (detailed in the next section, 4.2).

B2

The B2 controller is responsible for controlling the reading and writing operations performed on the B2.1 and B2.2 memories, featuring a memory controller interface and a Slave 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 B2 modules are configured with True Dual Port mode to enable the controller to read from both ports concurrently. This configuration enables the reading of the samples from two indexes at a time (as stated in section 2.1.2.1, the Backprojection algorithm reads samples from two indexes per cycle). The address of the reading operations is received by the controller from the Wbin module where it is incremented to obtain the second address needed. The memory switching for the reading operations is controlled by a flag that originates in the Pixel Position module.

B3

The B3 memory controller is responsible for controlling the reading and writing operations performed in the B3.1 and B3.2 memories. This controller has a finite state machine but unlike the other controllers so far, it has 2 memory controller interfaces - one for reading and one for writing - and one Master AXI Stream Interface connected to one of the Output FIFOs. The controller features a word counter to manage the reading and writing addresses, and to switch between operation mode.

The module has two operation modes. In the first mode the module reads a value from memory on one port and writes the result of the accumulation on the other port. In the second mode of operation the module reads a value from one port and writes the value zero on the other port, while sending the result of the accumulation through the AXI Stream interface. This second mode enables the initialization of memory between calculations from two different regions of the output image and is only used during the cycles of the calculations of the last pulse’s contribution.

B4

The B4.1 and B4.2 memory modules are used as part of another module, the AXI Data FIFO IP core. This IP core has one Slave AXI Stream Interface to receive the output values and one Master AXI Stream Interface to send the output values.

4.2 Algorithm Execution Circuit

This section presents the design of the part of the Hardware accelerator responsible for the execution of the Backprojection algorithm itself. 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. This starts by explaining the module’s composition, followed by an explanation of its functioning, finishing with observations on the details of the implementation. The tables in appendix C detail all the signals from the figures in this section and their corresponding fixed-point format.

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: $\mathrm{pulse}[i] \leftrightarrow \mathrm{pixel}[x,y]$.
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.

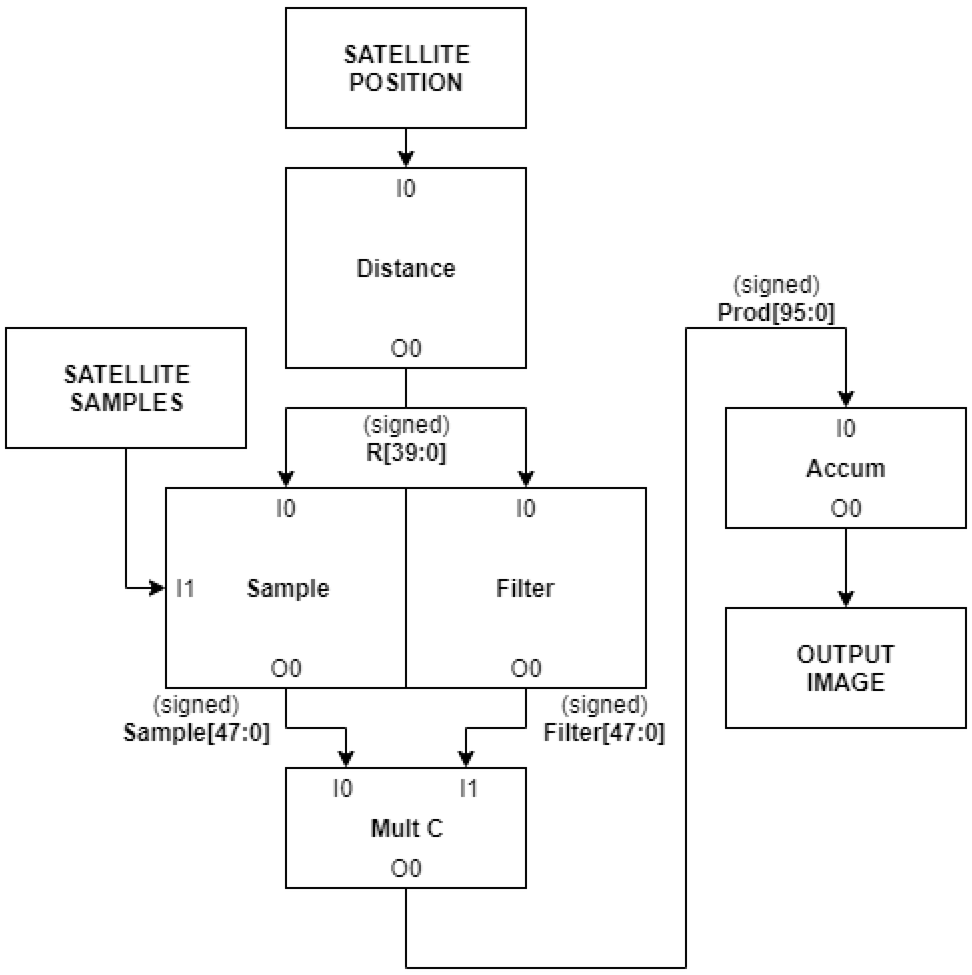


Figure 16 - Hardware Accelerator Top-Level Architecture

4.2.1 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.

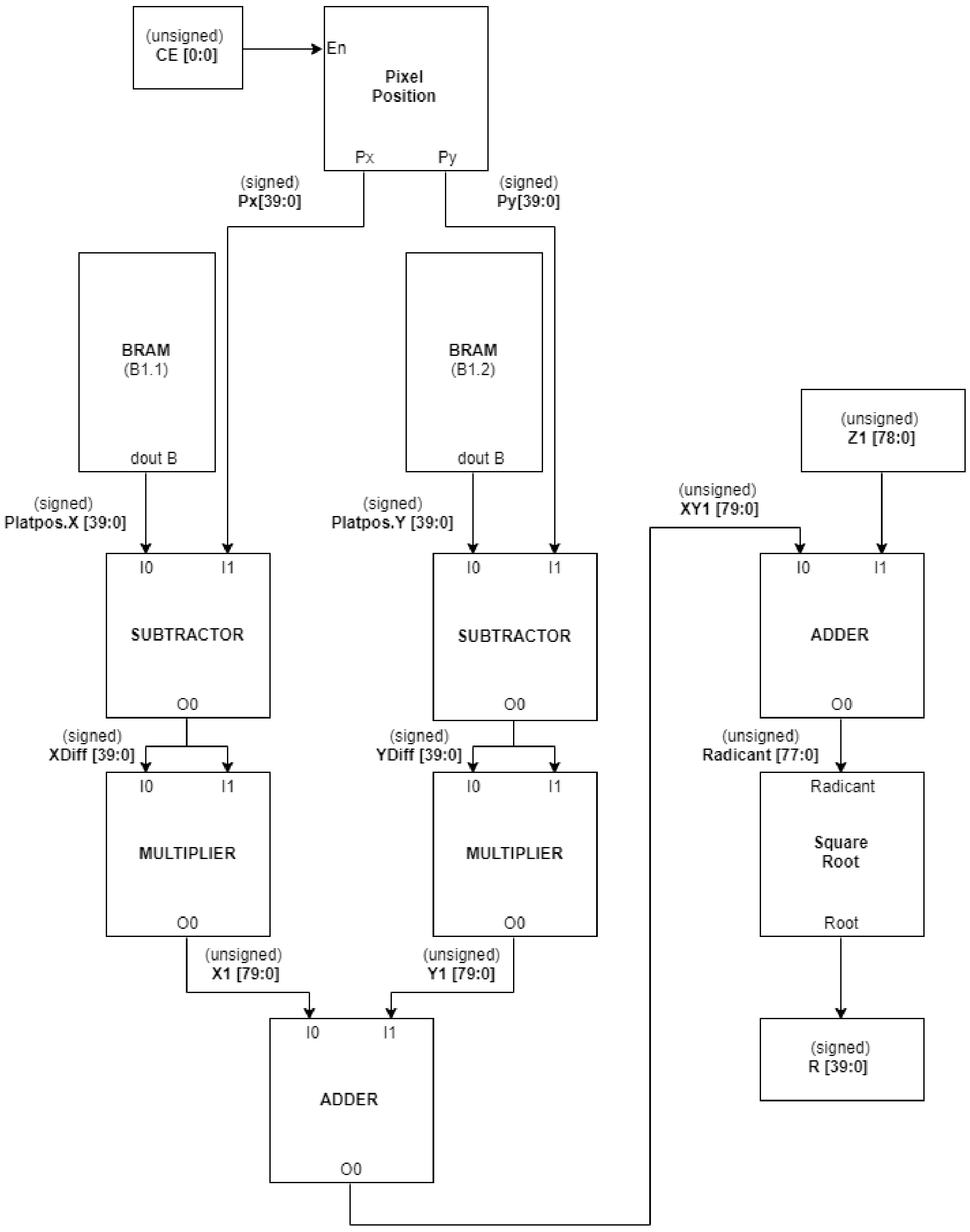


Figure 17 -Simplified Distance module block design.

To better organize the circuit, two sub modules were created: one for calculating the coordinates of the pixels in the sequence compatible with the schedule chosen for the algorithm's execution and another for calculating the square root of a number using the Binary restoring square root extraction method, as described in section 2.4.1.2.

Table 20 presents the Distance module composition in terms of basic IP cores, available from the free license of Vivado 2019.2.

Table 20 - Distance module composition in terms of basic arithmetic IP cores

IP core	Signal + Operator size[inA,inB,Out]	Latency	Quantity
Subtractor [39]	S + [40,40,40]	0	2
Multiplier [40]	U + [39,39,78]	5	2
Adder [39]	U + [78,78,78]	0	1
Adder [39]	U + [78,78,78]	1	1

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:

$$X_1 = (plat_x - p_x)^2 \tag{32}$$

$$Y_1 = (plat_x - p_y)^2 \tag{33}$$

$$Radicant = X_1 + Y_1 + Z_1 \tag{34}$$

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, which is the most used resource, allowing some flexibility during design.

Next, the pixel position and square root sub modules used will be presented in detail.

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.

The calculus of the pixel positions was original performed with the following equations:

$$\boldsymbol{p_x} = (\boldsymbol{-256} + \boldsymbol{0,5} + \boldsymbol{ix}) * \boldsymbol{dxdy};\ \boldsymbol{ix} \in [\boldsymbol{0,511}]; \tag{35}$$

$$\boldsymbol{p_y} = (\boldsymbol{-256} + \boldsymbol{0,5} + \boldsymbol{iy}) * \boldsymbol{dxdy};\ \boldsymbol{iy} \in [\boldsymbol{0,511}]; \tag{36}$$

But due to available options given by the technology used, the pixel positions can now be calculated using eq.38, 39 and 40:

$$Offset = (-255{,}5 * dxdy); \tag{38}$$
$$p_x = Offset + (ix * dxdy);\ ix \in [0, 511]; \tag{39}$$
$$p_y = Offset + (iy * dxdy);\ iy \in [0, 511]; \tag{40}$$

The new equations are a direct consequence of the use of the synchronized init feature of the binary counters. Due to the schedule chosen for the algorithm’s execution the equations are transformed into the new equations:

$$Ry = 16 * region;\ region \in [0{,}32]; \tag{41}$$
$$Offset_x = (-255{,}5 * dxdy); \tag{42}$$
$$Offset_y = (-255{,}5 * dxdy) + (Ry * dxdy); \tag{43}$$
$$p_x = Offset_x + (ix * dxdy);\ ix \in [0, 511]; \tag{44}$$
$$p_y = Offset_y + (iy * dxdy);\ iy \in [0, 15]; \tag{45}$$

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.

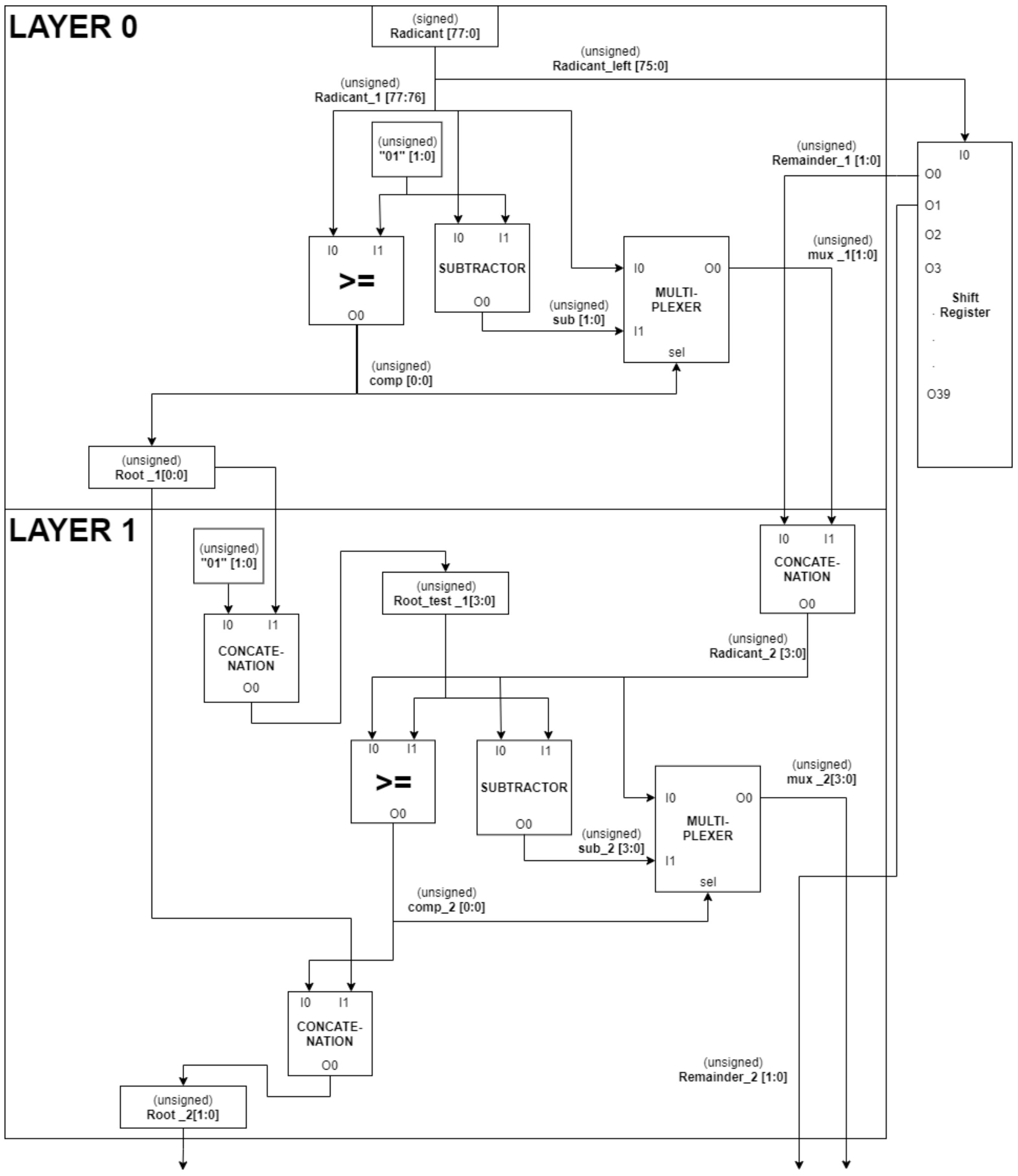


Figure 18 - Simplified Square Root module block design.

Table 21 shows the correspondence between the variables of the C code present in section 2.4.1.2 and the signals in figure 18:

*Table 21 - Correspondence between C code variables and circuit signals for the square root implementation**

Varibles	Signals
remLo	Radicant_left
remHi	Radicant_x
(remHi << 2)	Mux_x
(remLo >> 30)	Remainder_x
testDiv	Root_test
(remHi >= testDiv)	Comp_x
remHi -= testDiv	Sub_x
root	Root_x

*some signals represent the result of comparisons or other operations that do not have a correspondent variable in software.

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 takes the value of thirty-nine cycles.

4.2.2 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.

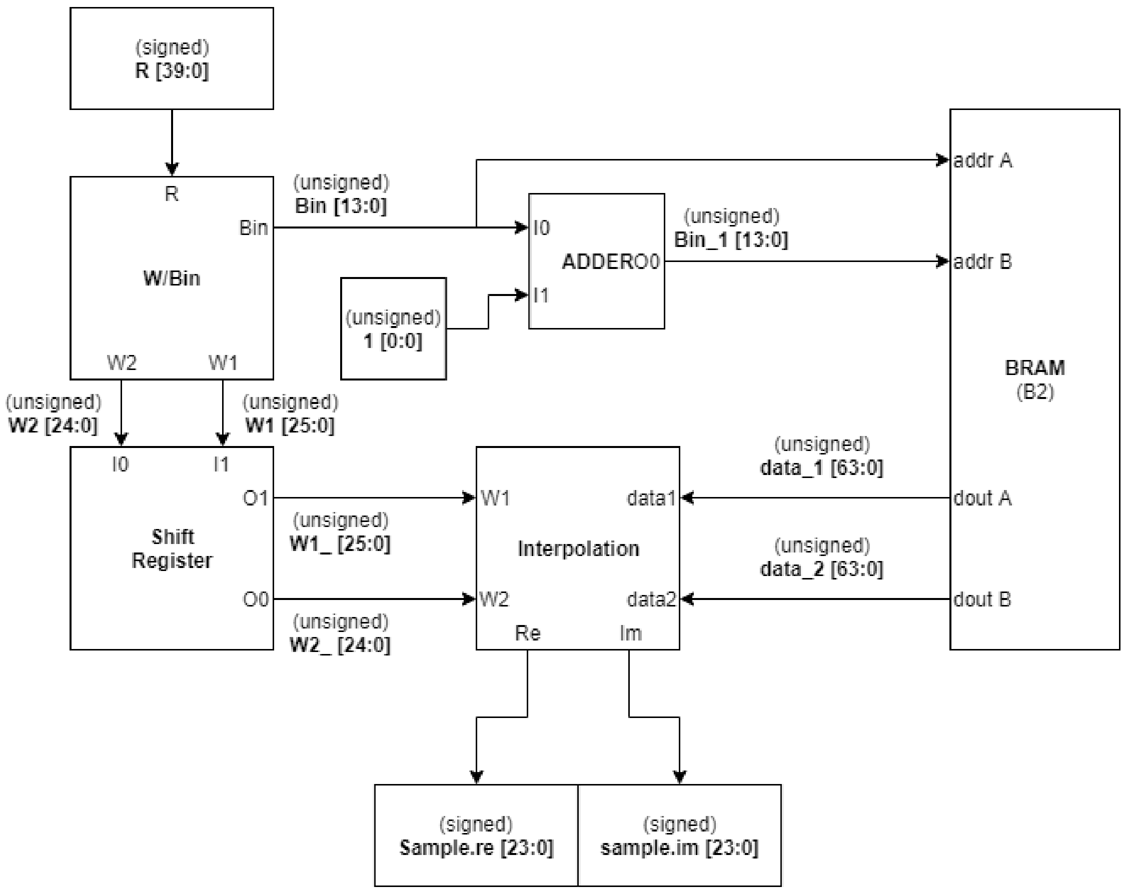


Figure 19 - Simplified Sample module block design.

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. The data1 and data2 signals have 64 bits because the real and imaginary parts are grouped in one signal.

The sub modules will be presented next.

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.

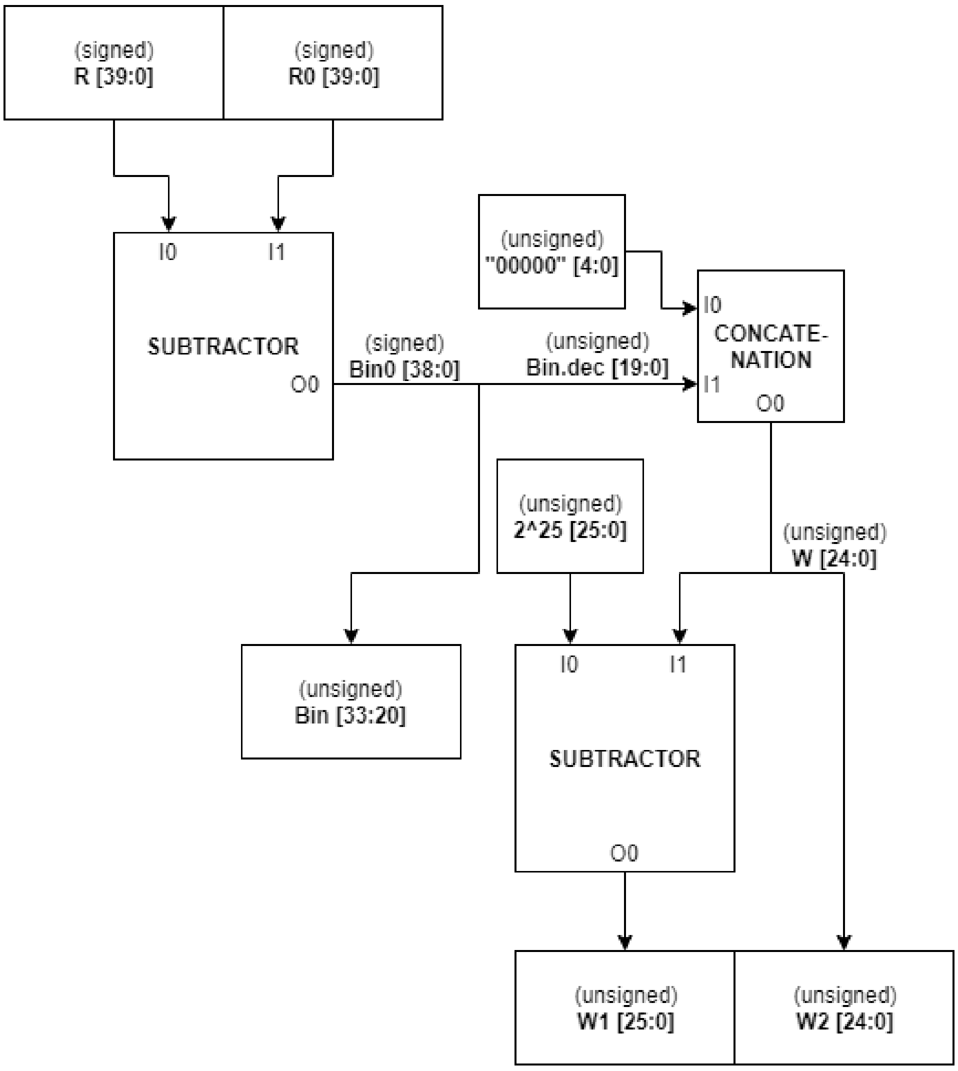


Figure 20 - Simplified WBin module block design.

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

$$Bin \ = \ \lfloor (R - R_0) * \ dR_{inv} \rfloor \tag{44}$$

$$W_2 \ = ((R - R_0) * \ dR_{inv}) - Bin \tag{45}$$

$$W_1 \ = 1 - W_2 \tag{46}$$

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:

$$Sample_{Re} = (data1_{Re} * W_1) + (data2_{Re} * W2) \quad (47)$$

$$Sample_{Im} = (data1_{Im} * W_1) + (data2_{Im} * W2) \quad (48)$$

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.

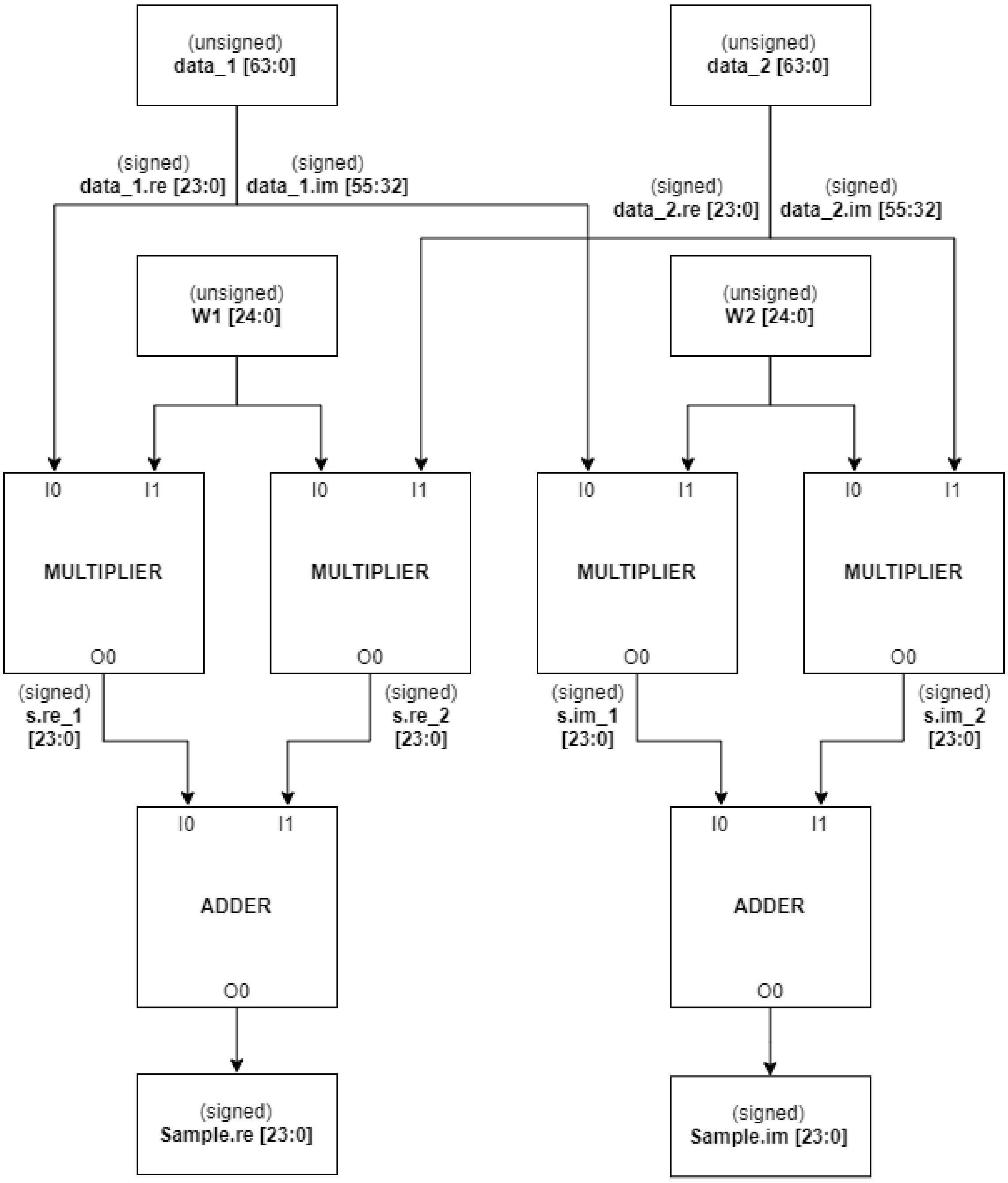


Figure 21 - Simplified Interpolation module block design.

This module was built using basic mathematical IP cores. Table 22 presents their configuration.

Table 22 - Interpolation module composition in terms of basic arithmetic IP cores

IP core	Signal + Operator size[inA,inB,Out]	Latency	Quantity
Multiplier	S + [24,24,24]	2	2
Multiplier	S + [24,25,24]	2	2
Adder	S + [24,24,24]	1	2

4.2.3 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.

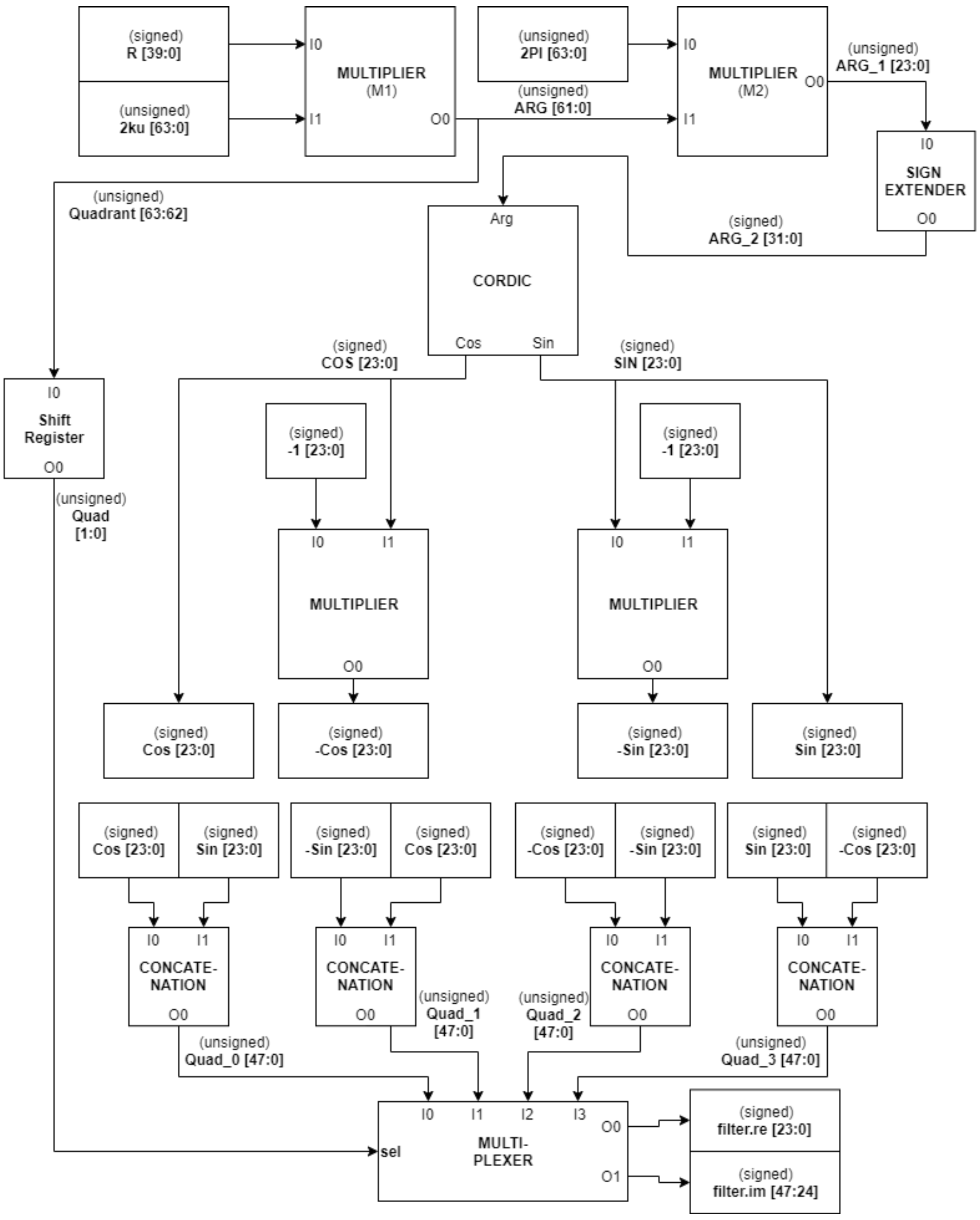


Figure 22 - Simplified Filter module block design.

Its composition in terms of basic modules is shown in table 23.

Table 23 - Filter module composition in terms of basic arithmetic IP cores

IP core	Signal + Operator size [inA,inB,Out]	Selected Output	Latency	Quantity
Multiplier	S + [24,24,24]	[23 : 0]	1	2
Multiplier M1	U + [40,64,64]	[81 : 18]	7	1
Multiplier M2	U + [62,64,24]	[126 : 103]	7	1

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 following the software implementation. Looking at table 23 it can be seen that contrary to other basic arithmetic modules, the M1 multiplier has a special configuration regarding the width of his output signal, as it does not represent the full output of the operation. This output bit selection is done to avoid a large increase in the bit width of signals.

The multiplier M1 performs a multiplication whose output represents an angle in scaled radians (meaning that to obtain the actual value in radians of the angle a further multiplication of the result by 2Pi must be performed), as shown in equations 50 and 51.

The operation performed by the M1 module is the following:

$$R * 2Ku \ = Arg \tag{49}$$

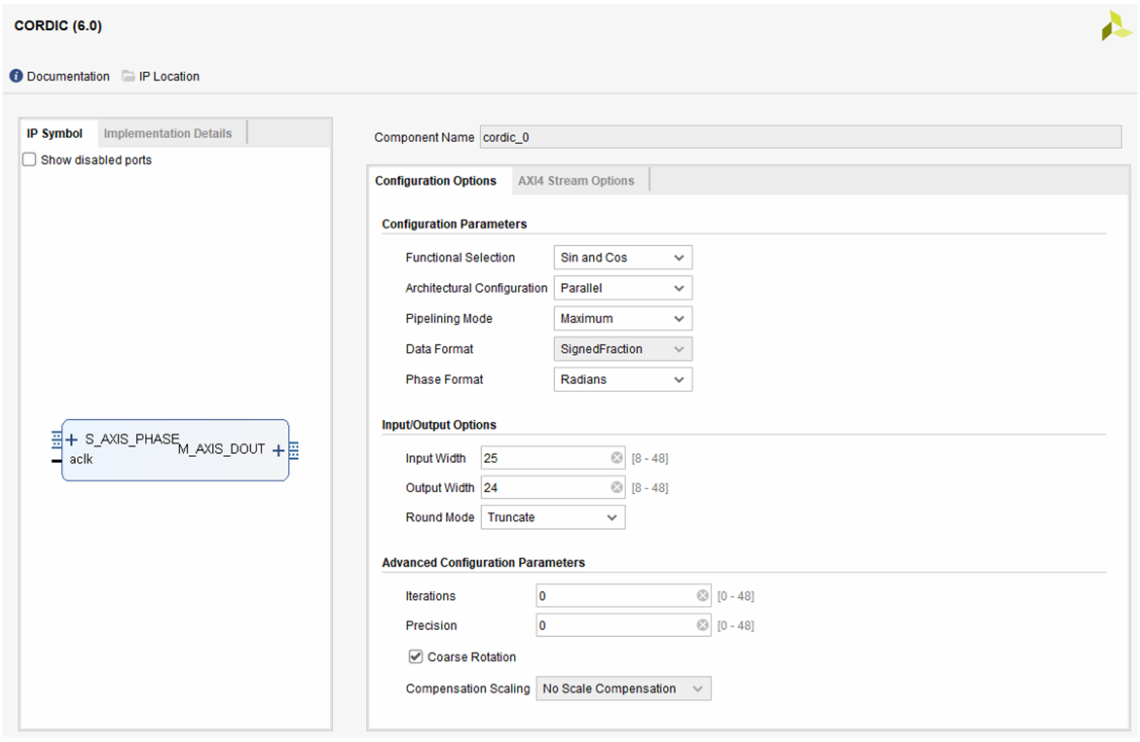
Transforming this equation by replacing variables with their units yields the following:

$$\begin{aligned} meters \ * \ 2\ (Laps\ around\ the\ trigonometric\ circle\ per\ meter) \\ = Laps\ around\ the\ trigonometric\ circle \end{aligned} \tag{50}$$

$$Laps\ around\ the\ trigonometric\ circle = Scaled\ Radians \tag{51}$$

Combining this information with the periodicity of the trigonometric functions it is possible to conclude that the integer part of this value is redundant and can be disregarded without losing precision. The output bits chosen for the module are the result of this conclusion and the effort to maintain the maximum signal width close to 64 bits. This approach differs from the original software implementation in the value stored in the Ku variable, as the original implementation stored a value in radians per meter. This difference is supported by the fact that an early conversion of the Ku value to radians per meter would only increase the signal's width without offering any advantage.

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. Figure 23 shows the configuration used on the CORDIC IP core [41].



Component Name: cordic_0

Configuration Parameters

Functional Selection	Sin and Cos
Architectural Configuration	Parallel
Pipelining Mode	Maximum
Data Format	SignedFraction
Phase Format	Radians

Input/Output Options

Input Width	25	[8 - 48]
Output Width	24	[8 - 48]
Round Mode	Truncate	

Advanced Configuration Parameters

Iterations	0	[0 - 48]
Precision	0	[0 - 48]

[✓] Coarse Rotation

Compensation Scaling: No Scale Compensation

Figure 23 - CORDIC IP core configuration.

This 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. As previously stated, it uses a shift register to compensate for the difference in latencies in the Datapath of the ARG signal and the Datapath of the Quadrant signal. 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.

4.2.3.1 Alternative Filter Module

The filter module was designed so that de CORDIC IP core would receive an angle in radians rotated to the first quadrant, mostly to follow the fixed-point software implementation of the filter and reduce the impact of the unknown internal architecture of the CORDIC IP core. This module can be redesigned to use less resources by performing simple alterations. Setting the input of the CORDIC IP core to scaled radians allows the removal of the M2 multiplier from the circuit. By converting the input angles in the interval from [0,2] to the interval [-1,1] (1 = π radians) we can achieve a circuit without quadrant normalization while maintaining the resource consumption and latency of the CORDIC IP core. This idea was not part of the final design because Vivado could not implement it in a working circuit. The module can achieve the desired timing, but Vivado could not find a working implementation for the whole system. Figure 24 shows the new version of the filter module.

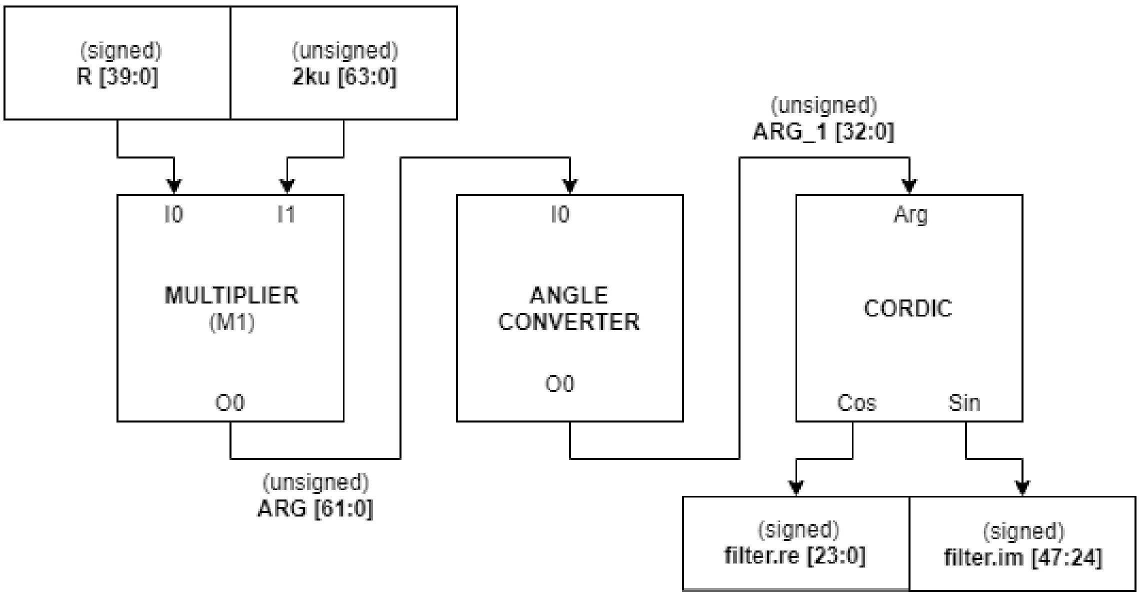
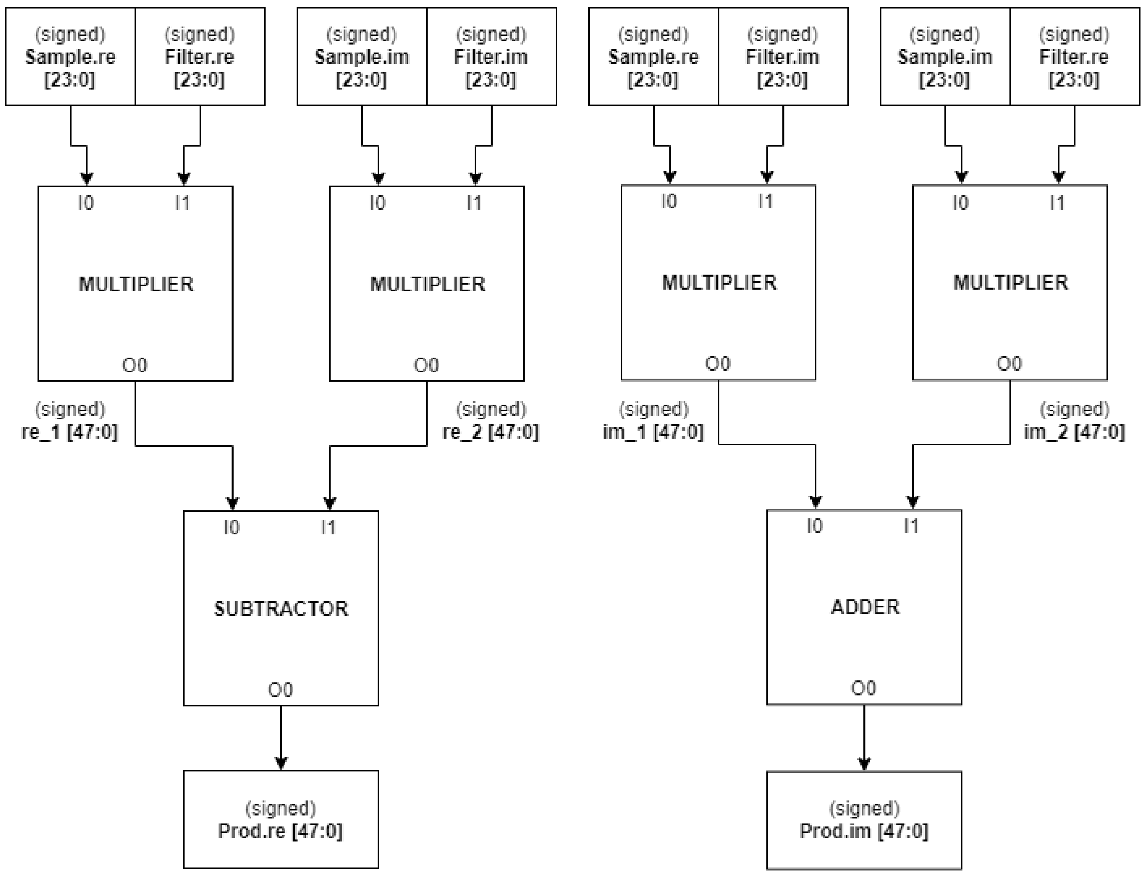


Figure 24 - Simplified Alternative Filter module block design.

4.2.4 Complex Multiplication Module

The MultC module is responsible for performing the complex multiplication between the samples and the matched filter.



(signed) Sample.re [23:0]
(signed) Filter.re [23:0]
(signed) Sample.im [23:0]
(signed) Filter.im [23:0]
(signed) Sample.re [23:0]
(signed) Filter.im [23:0]
(signed) Sample.im [23:0]
(signed) Filter.re [23:0]
I0
I1
MULTIPLIER
O0
(signed) re_1 [47:0]
(signed) re_2 [47:0]
(signed) im_1 [47:0]
(signed) im_2 [47:0]
SUBTRACTOR
ADDER
(signed) Prod.re [47:0]
(signed) Prod.im [47:0]

Figure 25 - Simplified MultC module block design.

It was implemented using only basic mathematical IP cores, performing the operations detailed in eq.52 and 53:

$$Prod_{Re} = (Sample_{Re} * Filter_{re}) - (Sample_{Im} * Filter_{Im}) \quad (52)$$

$$Prod_{Im} = (Sample_{Re} * Filter_{Im}) + (Sample_{im} * Filter_{Re}) \quad (53)$$

The module's composition is shown in table 24:

Table 24 - MultC module composition in terms of basic arithmetic IP cores

IP core	Signal + Operator size [inA,inB,Out]	Latency	Quantity
Multiplier	S + [24,24,24]	2	4
Subtractor	S + [24,24,24]	1	1
Adder	S + [24,24,24]	1	1

4.2.5 Accumulator Module

The Accumulator module is composed by the B3 memory and the circuit that performs the accumulation of values.

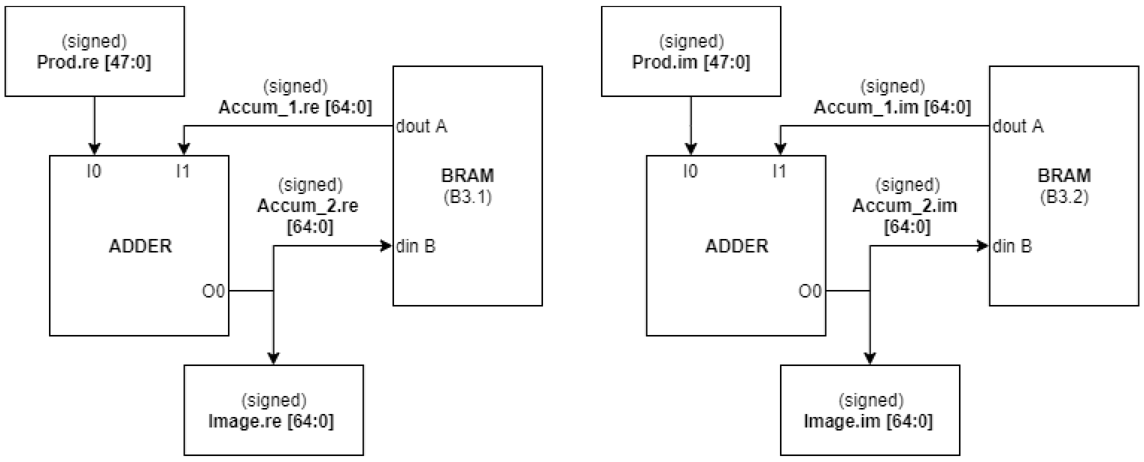


Figure 26 - Simplified Accumulator module block design.

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. This module is the last belonging to the part of the circuit responsible for computing the algorithm. The operations performed by this module are detailed in eq.54 and 55:

$$Accum_{Re_{i+1}} = Accum_{Re_i} + Prod_{Re_i};\ i\ \in\ [0; 511] \tag{54}$$

$$Accum_{Im_{i+1}} = Accum_{Im_i} + Prod_{Im_i}\ ;\ i\ \in\ [0; 511] \tag{55}$$

The module's composition is shown in table 25.

Table 25 - Accumulator module composition in terms of basic arithmetic IP cores

IP core	Signal + Operator size[inA,inB,Out]	Latency	Quantity
Adder	S + [48,64,64]	1*	2

* This adder has a latency of 1 cycle to achieve the timing constraint on the Datapath of that leads to the output of the module, not for the Accumulator's Loop Datapath.

4.3 System Control

To guarantee the correct execution of the Hardware Accelerator, a control mechanism was added. The need for this control system stems from the existence of an initialization phase to fill the BRAM modules. These BRAMs need to be populated with input values before the first iteration can begin, so the first condition of the control system is to wait until all the information needed for the first iteration is in the BRAMs. This means that all the positions of the satellite and all the samples from the first pulse were written in the BRAMs. The start flag changes its value to 1 when the first sample of the second pulse is being written.

The second condition of the control system is related to the output. Because the HWAccel output comes in bursts of values, the module that writes those values to the DDR, the AXI DMA, must start writing in the same cycle as the first value of a burst is calculated, otherwise that result is lost. This causes a problem since the AXI DMA doesn't give any control of its side of the channel to the programmer. To solve the late start and unsynchronized reading operations of the AXI DMA, a FIFO was introduced configured with 2 flags, the programmable empty and programmable full flags. With these flags the control system can now halt the execution before the FIFO is overloaded, leaving enough space for the results of the iterations that already begun, and resume execution before the FIFO becomes empty. This architecture allows minimizes time consumption with the type of control existent. In order to allow the software to correctly control the AXI DMA transfers a custom module was built, the FIFO Transfer Control, with the sole purpose of managing the TLAST flag of the AXI-Stream channel between the Data FIFO and the AXI DMA. The accelerator has an enable signal that runs through the whole circuit, 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.

5 Results

This chapter details the results obtained from the developed Hardware/Software system, beginning by presenting the image quality obtained, followed by the resources required by the design described in the previous chapter and the timing considerations, finishing with the predicted energy consumption breakdown.

5.1 SNR

This section presents the SNR values obtained from the execution of the system, calculated using the method described in section 2.1.4. 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.

Figures 27, 28, 29 and 30 show the image taken as the golden reference and the images formed by the original floating-point implementation, the fixed-point software implementation that achieved the highest dB score and the fixed-point Hardware/Software implementation, respectively.

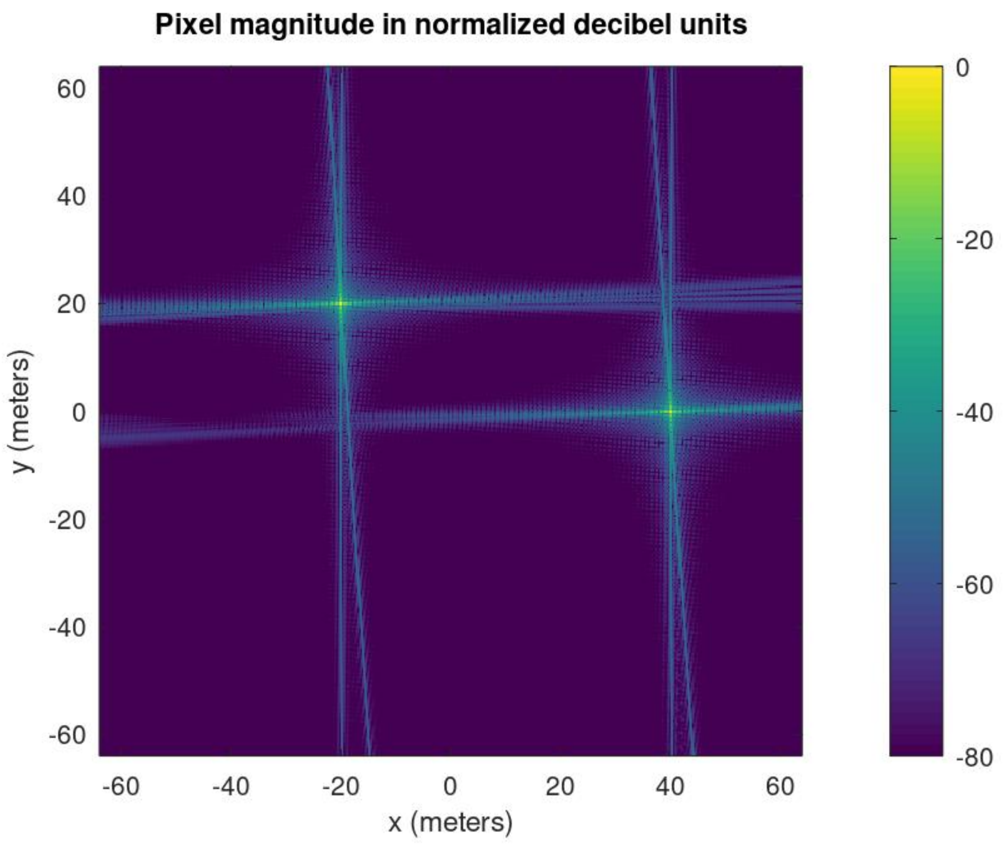


Figure 27 - Image correspondent to the golden reference

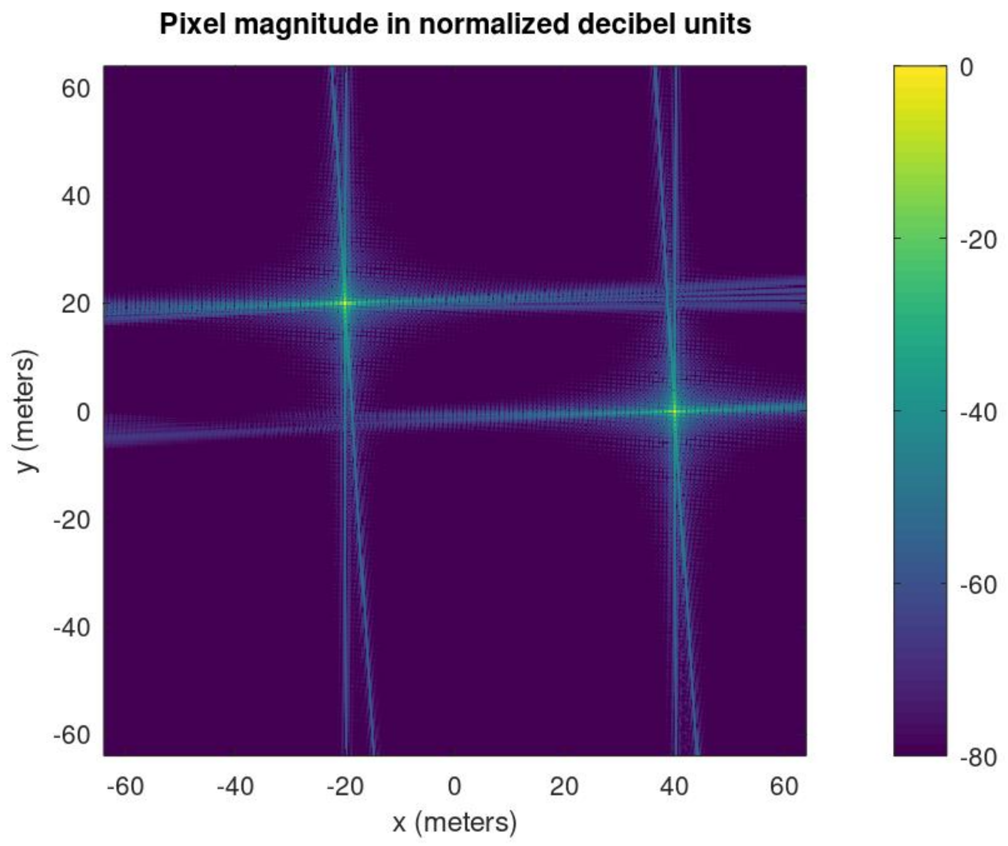


Figure 28 - Output of original algorithm (139 dB)

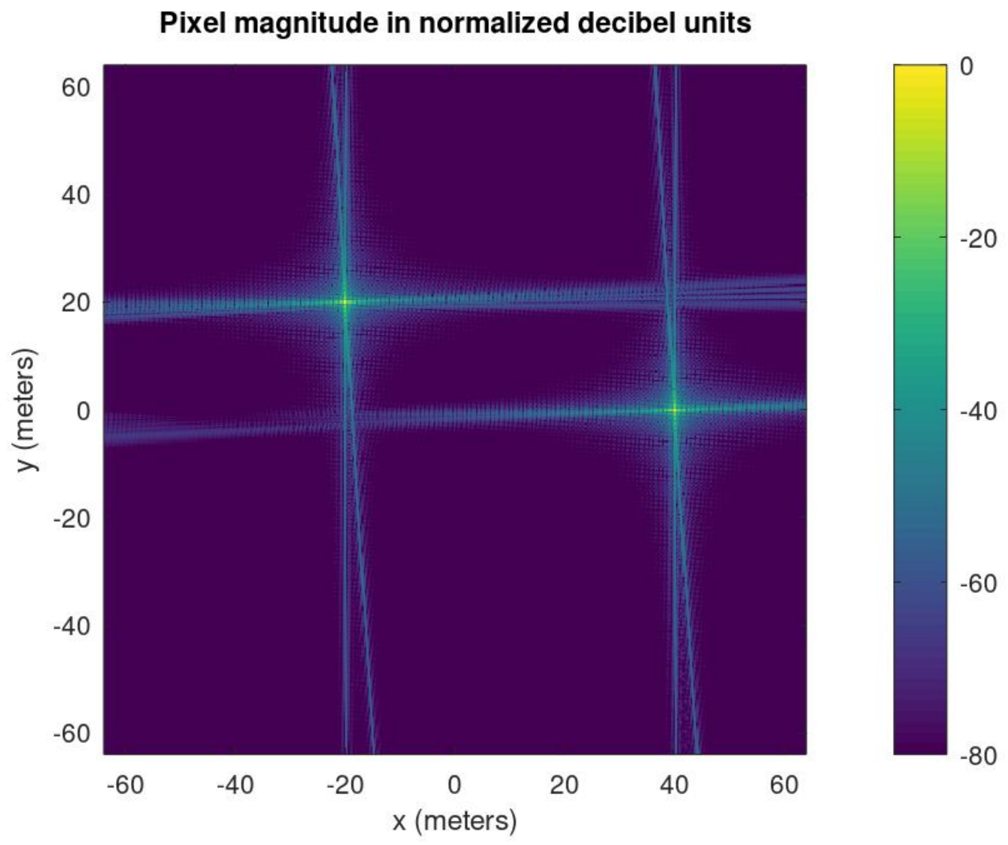


Figure 29 - Output of the fixed-point algorithm with the largest word-lengths (183 dB)

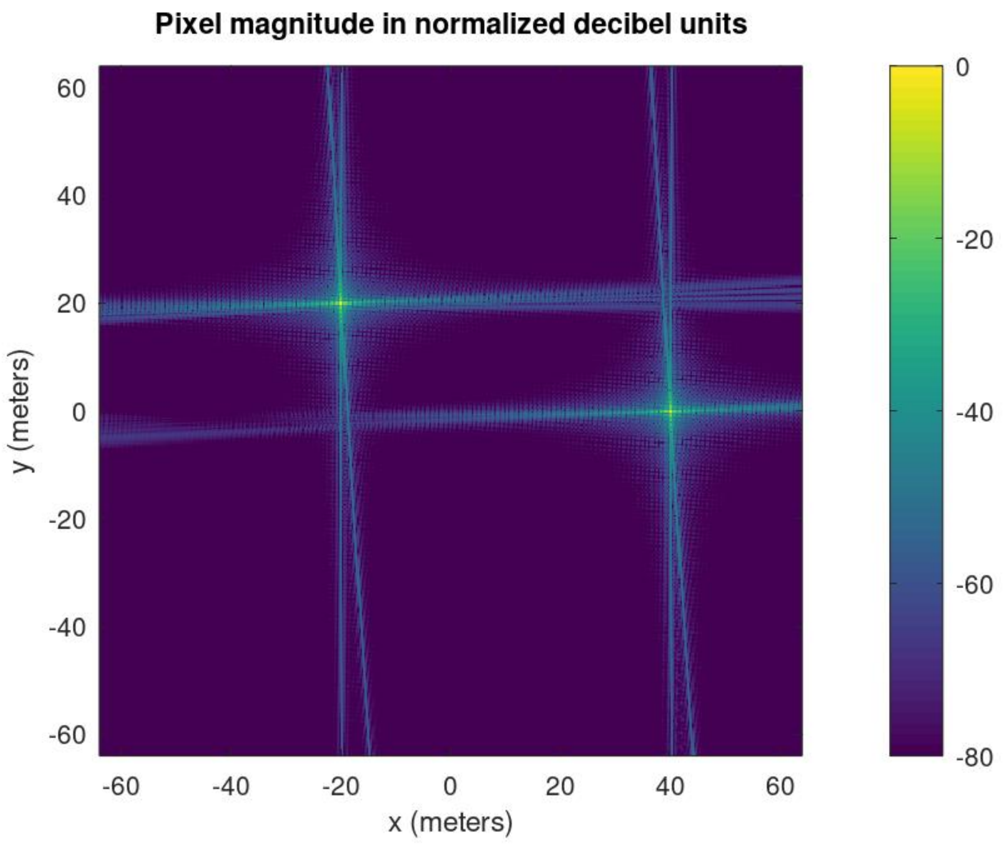


Figure 30 - Output of the final Hardware/Software implementation (99.21 dB)

The SNR value obtained is below the estimation. This difference can be explained by the precision offered by the CORDIC IP core used in the design. When comparing the results from this IP core with the software implementation, a divergence of up to 2 bits can be found between the output values. This difference wasn't 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.

5.2 Resources

This section presents the resources of the SoC device allocated by the Hw/Sw implementation. The PL resources are detailed first, followed by the CPU resources and the external memory used.

The resources allocated by the final design can be found in their entirety in the target device. Although the design fits the board, it does not leave room for much else, mainly due to the paths being used and not the resources being occupied. When comparing the resource allocation with estimations made in section 3.3, 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.

5.2.1 *Programmable Logic Resources*

Table 26 shows the amount of resources from the PL system allocated by the design after the implementation phase performed in Vivado. The values in table 26 correspond to the resources allocated by one unit of a module. The shift registers present between the top-level modules are omitted, with their contribution appearing only in the total resources allocated.

-

Table 26 - Resources used in PL by the whole system

Name	LUTs	Registers	DSPs	BRAMs	Slice
Distance	2292	2807	0	2	917
Sample	398	356	10	13	199
Filter	2198	2491	25	0	715
MultC	0	68	10	0	16
Accum	219	231	0	14.5	105
Init	2	2	0	0	1
Axi DMA	1636	2407	0	3	715
Axi-Interconnect (Axi-Stream)	543	650	0	0	234
Axi-Interconnect (Axi-Lite)	505	657	0	0	250
Axi Data FIFO	60	88	0	4	36
FIFO Transfer Control	8	20	0	0	10
Processor System Reset	16	33	0	0	11
TOTAL	10438 (59.3 %)	13304 (37.7 %)	55 (68.8 %)	58 (96.7 %)	3833 (87.1 %)

The rescheduling of the algorithm proved to be one of the biggest contributions to optimization in the design of the circuit, allowing to keep the architecture suggested after analysing the profiling study. This was achieved by lowering the BRAM resources needed by the implementation enough to fit the target device (as explained in section 3.5). This process reduced the BRAM needed by a factor of 77 times. It was through the rescheduling that the flexibility offered by this SoC system could be explored to this extent.

5.2.2 CPU and Memory

The final design only used one core of the Dual-core ARM Cortex-A9 CPU. Table 27 shows the contents of the Processing system's memories during execution (RAMs and DDR).

Table 27 - DDR and RAM memories occupancy through the execution (PS memories)

File	Address	Size	Memory Unit
Position.bin	0x01000000	8 KB	DDR
Samples.bin	0x02000000	16 384 KB	DDR
Output_real.bin	0x10000000	2 MB	DDR
Output_img.bin	0x20000000	2 MB	DDR
Code & Data	0x00000000	102.4 KB	Ram 0
Heap & Stack	0xFFFF0000	10 KB + 10 KB	Ram 1

5.3 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 28 shows the paths with the longest delays in the circuit (Paths within one module have the same From and To).

Table 28 - Slowest Datapaths between 2 endpoints, in the circuit

From	To	Total Delay (ns)	Logic Delay (ns)	Net Delay (ns)
AXI DMA 0	AXI DMA 0	7.02	3.80	3.23
AXI DMA 1	AXI DMA 1	7.012	3.86	3.15
*PP/Counter 0	PP/Counter 3	6.15	1.08	5.08

*PP is a reference to the Pixel Position module.

Table 29 shows the latency of the top-level modules of the system.

Table 29 - 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 execution time is very close to the expectations presented in section 3.2. The prediction is close to the result because the overwhelming factor in the execution time was known from the begging to be the number of iterations. The pipelined architecture and the re-scheduling of the algorithm allowed the overheads from accessing the external memory to be hidden without impacting the execution time. This last consideration is very important when considering that the system copies the full extent of the sample data input file once per region during execution. Although the frequency of the PL is 4.7 times lower than the CPU's, its capacity to initiate a new iteration in every clock cycle due to the pipeline architecture presents an advantage that the serial architecture of the CPU cannot compete with.

To achieve a valid design with a working frequency of 140 MHz some adjustments were made to the circuit, along with the employment of an implementation strategy focused on achieving the timing constraints. The adjustments consisted of adding extra pipeline stages to give Vivado the flexibility to separate physically two consecutive modules in the Datapath.

5.4 Energy Consumption

The energy consumption estimate provided by Vivado appears in table 30.

Table 30 - 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 table 30 it can be concluded that the CPU is responsible for most of the energy consumption of the system. When relating the power consumption profile presented in table 30 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.

These results also act as a proof of the superior energetic efficiency of the PL fabric.

6 Conclusion

This work explored the development of a method of optimization to be applied to a large range of mathematical algorithms, along with the design of an optimized system to compute the imaging forming algorithm for a SAR system.

The methodology's greatest contributions lie mainly in the world of embedded devices and microprocessors without a floating-point unit. The methodology was developed around 2 main ideas: the advantage offered by flexibility of an FPGA regarding parallel and pipeline architectures and the advantages offered by the fixed-point format. The use of an FPGA device allowed for the design of a pipeline architecture, the development of custom modules with custom-sized operators and custom storage solutions adapted to the memory access patterns of the algorithm. The fixed-point conversion allowed for several optimizations, including the replacement of multiplications by simple bitwise operations, the reduction of resources required and the use of binary arithmetic that can be easily implemented with digital logic. The influence of both these ideas is exposed throughout this work.

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 re-scheduling 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.

Although Vivado provides a good environment to design these types of projects, the time consumed in the development of the system depended heavily on the size of the algorithm that would be executed in programmable hardware. The ways to perform the debugging and testing in these systems are still very slow.

As a result of this work, a cheap, lightweight, small, power-efficient solution was achieved, capable of computing the Backprojection algorithm to form an image of 512*512 pixels in less than a second.

6.1 Future Work

The fact that the device used in this work is low-end should show the potential of the SoC FPGA family in optimization efforts. The cheapest investment that can be made to achieve a significant performance upgrade is the use of a larger device that can present parallelization opportunities that were not taken in this work due to the lack of resources. With a high-end device, the working frequency could be increased, and the circuit could be cloned to easily create a parallel architecture, requiring small modifications in the Pixel Position module.

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. Furthermore, none of the logic applied or the instructions depend on the higher frequency of the CPU to function properly. With these considerations in mind, the cheapest-fastest way to reduce energy consumption would be to replace the CPU with a custom module implemented in the FPGA fabric and choosing a more appropriate target device.

While maintaining the new schedule of the algorithm, a continuous working mode could be built. Because the circuit computes the output image strip by strip, it could be adapted to accept new samples and compute new areas as the SAR platform moves, creating a stream of images.

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, detailed in table 28 in section 5.3.

A combination of these improvements could result in a product with useful real-time applications in a device with good portability characteristics.

References

[1] 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.

[2] 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.

[3] 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.

[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] 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.

[6] R. P. Duarte, H. Cruz, and H. Neto, *Reconfigurable Accelerator for On-Board SAR Imaging Using the Backprojection Algorithm*, vol. 10824. 2018.

[7] 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.

[8] C. Colesanti and J. Wasowski, “Investigating landslides with space-borne Synthetic Aperture Radar (SAR) interferometry,” *Eng. Geol.*, vol. 88, no. 3–4, pp. 173–199, 2006, doi: 10.1016/j.enggeo.2006.09.013.

[9] A. Moreira, “Synthetic aperture radar (SAR): Principles and applications,” *4th Adv. Train. Course L. Remote Sens.*, pp. 1–5, 2013.

[10] T. R. Lauknes, “Rockslide Mapping in Norway by Means of Interferometric SAR Time Series Analysis,” *PhD Diss. Univ. Tromsø*, no. December, p. 124 pages., 2011.

[11] K.-S. Chen, *Principles of Synthetic Radar Imaging*. 2016.

[12] S. Ramakrishnan, V. Demarcus, J. Le Ny, N. Patwari, and J. Gussy, “Synthetic Aperture Radar Imaging Using Spectral Estimation Techniques,” *Univ. Michigan, EECS 559 - Adv. Signal Process.*, 2002.

[13] A. F. Yegulalp, “Fast backprojection algorithm for synthetic aperture radar,” *IEEE Natl. Radar Conf. - Proc.*, pp. 60–65, 1999, doi: 10.1109/nrc.1999.767270.

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

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

[16] H. Alexandra and S. Ruivo, “SAR Imaging Architecture,” no. November, 2018.

[17] Altera Corporation, “What is an SoC FPGA?,” *Altera Support Resour.*, pp. 1–4, 2014, [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ab/ab1_soc_fpga.pdf.

[18] W. Chapman *et al.*, “Parallel processing techniques for the processing of synthetic aperture radar data on GPUs,” *IEEE Int. Symp. Signal Process. Inf. Technol. ISSPIT 2011*, pp. 573–580, 2011, doi: 10.1109/ISSPIT.2011.6151626.

[19] 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.

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

[21] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, “A 7.663-TOPS 8.2-W Energy-efficient FPGA Accelerator for Binary Convolutional Neural Networks (Abstract Only),” 2017, pp. 290–291, doi: 10.1145/3020078.3021786.

[22] H. F. W. Sadrozinski and J. Wu, *Applications of Field-Programmable Gate Arrays in Scientific Research*, vol. 4, no. 1. 2016.

[23] M. Gocho, N. Oishi, and A. Ozaki, “Distributed Parallel Backprojection for Real-Time Stripmap SAR Imaging on GPU Clusters,” *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, vol. 2017-September, pp. 619–620, 2017, doi: 10.1109/CLUSTER.2017.64.

[24] X. Song and W. Yu, “Processing video-SAR data with the fast backprojection method,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 52, no. 6, pp. 2838–2848, 2016, doi: 10.1109/TAES.2016.150581.

[25] M. Wielage, F. Cholewa, C. Riggers, P. Pirsch, and H. Blume, “Parallelization strategies for fast factorized backprojection SAR on embedded multi-core architectures,” *2017 IEEE Int. Conf. Microwaves, Antennas, Commun. Electron. Syst. COMCAS 2017*, vol. 2017-November, pp. 1–6, 2017, doi: 10.1109/COMCAS.2017.8244770.

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

[27] 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.

[28] 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.

[29] Microprocessor Standards Committee, *IEEE Standard for Floating-Point Arithmetic - IEEE Xplore Document*. 2019.

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

[31] A. Mathematics and S. Review, “Historical Development of the Newton-Raphson Method,” *Society*, vol. 37, no. 4, pp. 531–551, 2012.

[32] 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.

[33] T. Sutikno, “an Optimized Square Root Algorithm,” pp. 1–9.

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

[35] D. Barton, “On Taylor Series and Stiff Equations,” *ACM Trans. Math. Softw.*, vol. 6, no. 3, pp. 280–294, 1980, doi: 10.1145/355900.355902.

[36] I. Xilinx, “Block Memory Generator v8.3,” *Data Sheet*, pp. 1–129, 2017.

[37] Xilinx, “AXI4-Stream Infrastructure IP Suite v3.0 LogiCORE IP Product Guide Vivado Design Suite,” 2018, [Online]. Available: www.xilinx.com.

[38] Xilinx, “AXI DMA Reference Guide v7.1,” *Pg021*, pp. 1–7, 2019.

[39] Xilinx, “Xilinx Adder/Subtracter,” *LogiCORE IP Prod. Guid.*, 2015.

[40] Xilinx Inc., “Multiplier v12.0,” 2015, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23917890.

[41] Xilinx.inc, “CORDIC v6.0,” no. December, 2017, [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf.

Appendix A – Word-length study results

Table 31 - Results from Word-length study of Constants

Constants	Value	No. Bits
ku	33.333	5
dxdy	0.25	0
dR_inv	32	5
Z0	0	0
R0	9936	14

Table 32 - Results from Word-length study of the Distance group variables

Variables	Max **	Nº Bits
ix	511	9
iy	511	9
Px	63.875	5
Py	63.875	5
PosX	7071.067	13
PosY	423.182	9
PosZ	7071.067	13
Xdiff	7007.192	12
Ydiff	487.057	9
Zdiff	7071.067	13
X1	50,907,409.773	26

Y1	237,224.914	19
Z1	50,000,000.837	26
XY	50,963,934.279	26
XYZ	100,963,935.117	28
R	10,048.081	14

Table 33 - Results from Word-length study of Sample group variables

Variables	**Max ****	**Nº Bits**
Rdiff	112.081	7
Bin	3586.59	11
Bin floor	3586	11
W	< 1	0
Wc	< 1	0
Data.Re	< 1	0
Data.Im	< 1	0
Sample.Re	< 1	0
Sample.Im	< 1	0

Table 34 - Results from Word-length study of Filter group variables

Variables	Max **	Nº Bits
Quadrant	3	2
Arg *	669,872.077	20
Filter.Re (cos)	1	1
Filter.Im (sin)	1	1

Table 35 - Results from Word-length study of Complex Multiplication group variables

Variables	Max **	Nº Bits
Prod.Re	1.000	1
Prod.Im	1.000	1
Accum.Re	295.677	9
Accum.Im	325.648	9

Appendix B - Area models for basic arithmetic operators with operands of different sizes

The following tables contain the resources needed to implement basic arithmetic operators in the PL fabric. The operators are studied for several operand sizes.

Table 36 - Resources used by a multiplier, latency = 0, implemented with DSP

Input Size	Output Size	DSP	Latency
64	128	16	0
56	112	10	0
48	96	9	0
40	80	5	0
32	64	4	0
24	48	2	0
16	32	1	0

Table 37 -Resources used by an adder, latency = 0, implemented with LUT

Input Size	Output Size	LUT	Latency
128	128	128	0
64	64	64	0
56	56	56	0
48	48	48	0
40	40	40	0
32	32	32	0
24	24	24	0
16	16	16	0

Table 38 - Resources used by a multiplier, latency = 1, implemented with DSP

Input Size	Output Size	Register	DSP	Latency (cycles)	Slack (ns)
64	128	102	16	1	-24.026
56	112	85	10	1	-13.693
48	96	68	9	1	-11.980
40	80	51	5	1	-5.128
32	64	34	4	1	-3.415
24	48	17	2	1	0.011
16	32	0	1	1	1.725

Table 39 - Resources used by an adder, latency = 1, implemented with LUT

Input Size	Output Size	LUT	Register	Latency (cycles)	Slack (ns)
128	128	128	128	1	2.254
64	64	64	64	1	3.225
56	56	56	56	1	3.453
48	48	48	48	1	3.681
40	40	40	40	1	3.909
32	32	32	32	1	4.137
24	24	24	24	1	4.365
16	16	16	16	1	4.593

Table 40 - Resources used by an adder, latency = 1, implemented with DSP

Input Size	Output Size	DSP	Latency (cycles)	Slack (ns)
48	48	1	1	3.746
40	40	1	1	3.746
32	32	1	1	3.746
24	24	1	1	3.746
16	16	1	1	3.746

Table 41 - Resources used by a multiplier, latency = 2, implemented with DSP

Input Size	Output Size	Register	DSP	Latency (cycles)	Slack (ns)
64	128	102	16	2	-22.765
56	112	85	10	2	-12.432
48	96	68	9	2	-10.719
40	80	51	5	2	-3.867
32	64	34	4	2	-2.154
24	48	17	2	2	1.272

Table 42 - Resources used by a multiplier, latency = 3, implemented with DSP

Input Size	Output Size	LUT	Register	DSP	Latency (cycles)	Slack (ns)
64	128	27	102	16	3	-3.867
56	112	9	85	10	3	-5.513
48	96	9	68	9	3	-3.800
40	80	0	51	5	3	-0.374
32	64	0	34	4	3	1.272

Table 43 - Resources used by a multiplier, latency = 4, implemented with DSP

Input Size	Output Size	LUT	Register	DSP	Latency (cycles)	Slack (ns)
64	128	27	132	16	4	-8.994
56	112	9	158	10	4	-5.513
48	96	9	99	9	4	-3.800
40	80	9	74	5	4	1.339

Table 44 - Resources used by a multiplier, latency = 5, implemented with DSP

Input Size	Output Size	LUT	Register	DSP	Latency (cycles)	Slack (ns)
64	128	52	210	16	5	-2.087
56	112	39	181	10	5	-0.374
48	96	34	128	9	5	1.339

Table 45 - Resources used by a multiplier, latency = 6, implemented with DSP

Input Size	Output Size	LUT	Register	DSP	Latency (cycles)	Slack (ns)
64	128	77	250	16	6	-0.496
56	112	57	182	10	6	1.272

Table 46 - Resources used by a multiplier, latency = 7, implemented with DSP

Input Size	Output Size	LUT	Register	DSP	Latency (cycles)	Slack (ns)
64	128	113	331	16	7	1.272

Appendix C – Fixed-point formats for circuit signals

Table 47- Fixed-Point formats of the signals in the figures in section 4.2

Signal	Format
DISTANCE:	
Platpos.X	Q14.25
Platpos.Y	Q14.25
Px	Q14.25
Py	Q14.25
XDiff	Q14.25
YDiff	Q14.25
X1	Q30.50
Y1	Q30.50
XY1	Q30.50
Z1	Q29.50
Radicant	Q28.50
R	Q14.25
FILTER:	
2ku	Q7.57
ARG	Q0.64
Quadrant	Q2.0
2PI	Q3.61
ARG_1	Q1.22
ARG_2	Q9.22
SEN	Q1.22
COS	Q1.22
SEN_B	Q1.22
COS_B	Q1.22
QUAD_0	Q48.0
QUAD_1	Q48.0
QUAD_2	Q48.0
QUAD_3	Q48.0
Quad	Q2.0
filter.re	Q1.22
filter.im	Q1.22
SAMPLE:	
WBIN:	
R0	Q14.25

bin_0	Q13.25
bin_dec	Q0.20
Bin	Q14.0
W1	Q1.25
W2	Q0.25
W1_	Q1.25
W2_	Q0.25
Bin_1	Q14.0
data_1	Q64.0
data_2	Q64.0
INTERPOLATION:	
data_1.re	Q1.22
data_1.im	Q1.22
data_2.re	Q1.22
data_2.im	Q1.22
s_re_1	Q1.22
s_re_2	Q1.22
s_im_1	Q1.22
s_im_2	Q1.22
sample.re	Q1.22
sample.im	Q1.22
MULT C:	
re_1	Q3.44
re_2	Q3.44
im_1	Q3.44
im_2	Q3.44
Prod.re	Q3.44
Prod.im	Q3.44
ACCUM:	
Accum_1.re	Q19.44
Accum_2.re	Q19.44
Accum_1.im	Q19.44
Accum_2.im	Q19.44