INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Electrónica e Telecomunicações e de Computadores

LITESAR – Sistema de Análise de Imagens SAR em Tempo-Real

David José Queirós Mesquita Mourato Mota
(Licenciado)

Trabalho de Projecto ou Dissertação de natureza científica para obtenção do grau de Mestre
em Engenharia de Eletrónica e Telecomunicações

Orientador:
DOUTOR Mário Pereira Véstias

Júri:
Presidente: DOUTOR Carlos Eduardo Meneses Ribeiro
Vogais:
DOUTOR José Manuel Peixoto do Nascimento
DOUTOR Mário Pereira Véstias

Dezembro de 2021

Agradecimentos

Quero começar por agradecer ao meu orientador de Mestrado, Mário Véstias pela sua disponibilidade e suporte na realização do projecto.

À minha família pelo suporte dado durante a realização desta dissertação.

Resumo

O SAR (*Synthetic Aperture Radar*) é uma forma de RADAR (*Radio Detection And Ranging*) que tem sido bastante utilizada e desenvolvida ao longo dos últimos anos. Esta técnica permite a criação de imagens com alta resolução para, por exemplo, deteção de objetos, independentemente das condições meteorológicas e de ser dia ou noite.

O processamento em tempo-real das imagens SAR permite a tomada de decisões junto dos dados, sem haver necessidade de os transmitir para uma plataforma terrestre para posterior processamento. Além disso, o processamento dos dados à medida que os vai recebendo, permite que estes não tenham de ser armazenados.

Neste trabalho, pretende-se o desenvolvimento de um sistema (LiteSAR) de análise de imagens SAR em FPGA (*Field Programmable Gate Array*) de baixa densidade, para ser utilizado em ambientes com recursos operacionais limitados, como aeronaves e satélites, com capacidade elevada de computação que permita um processamento em tempo real das imagens SAR.

A escolha da implementação deste sistema em FPGA deve-se ao facto de estes dispositivos apresentarem boas eficiências energéticas quando comparados com outros dispositivos de computação. Além disso, Mesmo os dispositivos de baixa custo, apropriados para este tipo de sistemas, conseguem bons desempenhos computacionais. A FPGA também tem a vantagem de poder ser adaptada a modificações dos algoritmos, ao contrário dos dispositivos de aplicação específica que, apesar de terem melhor desempenhos, estão dedicados a um determinado algoritmo.

Com o desenvolvimento do sistema LiteSAR consegue-se uma taxa de processamento de imagens de cerca de 1 imagem por segundo (FPS - *Frames per Second*). Esta taxa de processamento de imagens foi atingida com a implementação do sistema numa FPGA ZYNQ7020 a uma frequência de relógio de 127MHz.

Palavras-chave: FPGA,LiteSAR, Processamento de imagens, SAR, Tempo-real, ZYNQ7020.

Abstract

SAR (Synthetic Aperture Radar) is a form of RADAR (*Radio Detection And Ranging*) that has been widely used and developed over the last years. This technic buits images with high resolution that can be used, for example, for object detection, independent of the metereological conditions and the time of the day.

The processing of real-time SAR images allows the decision making on the imaging plataform without the need to transmit the image to a ground platform for processing. Moreover, the real-time image processing does not need to store the data, which allows to save hardware storage.

This project aims at the development of a system (LiteSAR) that analyses SAR images with low-density FPGA to be used in resource limited environments like satellites and drones with high computing capacity allowing for real-time SAR image processing.

In this project an FPGA has been adopted since it provides good power efficiency when compared with other computing devices. Besides, even low-cost devices, appropriate for this type of systems, achieve goog computing performances, The FPGA also has the advantage of being able to adapt to algorithm changes unlike the specific hardware devices that although better in performance are committed to an algorithm.

With the development of the LiteSAR system it is possible to achive a processing speed of 1 FPS (Frame per Second). This speed was achieved with the implementation of the system in a FPGA ZYNQ7020 with a frequency of 127MHz.

Keywords: FPGA, Image processing, LiteSAR, Real time, SAR, ZYNQ7020.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

ARM Advanced RISC Machine

AXI Advanced eXtensible Interface

BRAM Block Random-Access Memory

CORDIC Coordinate Rotation Digital Computer

CPU Central Processing Unit

CSA Chirp Scaling Algorithm

FFT Fast Fourier Transform

FIFO First In First Out

FPGA Field-Programmable Gate Array

FPS Frames Per Second

GBP Global Back-Projection

HDL Hardware Description Language

ICBM Intercontinental Ballistic Missile

IP Intellectual Property

IRE Institute Of Radio Enginners

KA Omega-K Algorithm

PLA Programmable Logic Array

PROM Programmable Read-Only Memory

RADAR Radio Detection And Ranging

RAM Random-Access Memory

RAR Real Aperture Radar

RDA Range Doppler Algorithm

RISC Reduced Instruction Set Computer

SAR Synthetic-Aperture Radar

SDK Software Development Kit

SLAR Side Looking Airborne Radar

SNR Signal-To-Noise Ratio

SoC System-On-Chip

UAV Unmanned Aerial Vehicle

VHDL VHSIC Hardware Description Language

VHSIC Very-High-Speed Integrated Circuit

1. Introdução

Em Junho de 1951, no Goodyear Aircraft Company Litchfield Park, Arizona, o matemático Carl A. Wiley inventou o Synthetic Aperture Radar (SAR)[12] enquanto trabalhava num sistema de orientação de correlação para o programa da Atlas ICBM (*Intercontinental Ballistic Missile*) [21].

Em abril de 1960, o SAR foi reconhecido publicamente através de um comunicado de imprensa sobre um sistema experimental do Exército dos Estados Unidos (*AN/UPD-1 system*), que consistia num elemento aéreo feito pela Texas Instruments, instalado num aeroplano (*Beech L-23D*) com uma estação para processamento de dados, sem revelar a forma do processamento dos dados.

Em fevereiro de 1961, um artigo técnico foi apresentado no jornal IRE (Institute of Radio Engineers) [5] descrevendo o principio do *AN/UPD-1 system* sem apresentar o processamento feito aos dados ou a resolução máxima.

Em junho de 1960, membros do *Michigan group* apresentaram o ”Optical Data Processing and Filtering Systems”[4] que apesar de não referir o uso destas técnicas para radar era possível perceber a existência duma ligação entre os dois artigos, que partilham alguns autores.

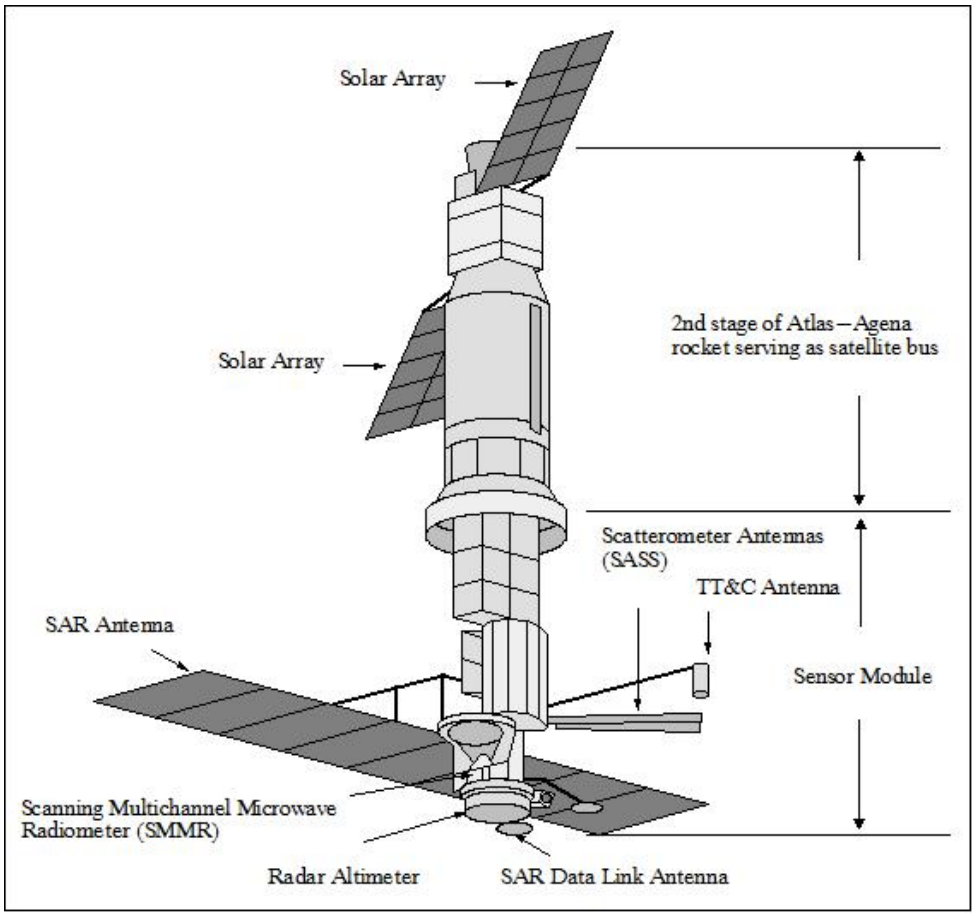

Figura 1: Primeiro satélite civil SAR. [14]

No desenvolvimento do SAR, foi entendido que o trajeto orbital das plataformas fora da atmosfera era ideal para a operação do SAR. Experiências com satélites também demonstraram que o efeito de Doppler através da atmosfera e da ionosfera eram estáveis o suficiente para permitir uma resolução fina a milhares de quilómetros[17].

O SAR também pode ser implementado como SAR inverso, observando um objeto em movimento durante um tempo com uma antena estacionária.

Atualmente estão a operar mais de 15 satélites com sensores SAR[14]. O primeiro satélite SAR civil foi lançado a 27 de Junho de 1978 (Figura 1).

O SAR tem aplicações em sensores remotos e no mapeamento das superfícies da Terra e de outros planetas. Outras aplicações são a monitorização do crescimento urbano, geologia, glaciologia, monitorização ambiental como derramamentos de óleo e inundações, monitorização de vulcões, mudanças globais, oceanografia, terremotos, topografia, agricultura e vigilância militar[19].

1.1 FPGA

A FPGA (Field Programmable Gate Array) é um dispositivo com recursos hardware reconfiguráveis capaz de ser reconfigurado depois de ser fabricado[1]. Este dispositivo é constituído por lógica e interligações programáveis que possibilitam o projeto e desenvolvimento de implementações otimizadas de sistemas hardware de complexidade diversa.

Algumas famílias das FPGAs atuais apresentam arquiteturas com sistemas completos de processamento com um ou mais microprocessadores embebidos para além da lógica reconfigurável. Este tipo de arquitetura é designado *System on chip* (SoC) e pretende ser uma solução de implementação com a capacidade de desenvolver numa única FPGA sistemas que utilizem hardware e/ou software, podendo assim combinar os dois para resultar num sistema que tira partido das vantagens de ambos.

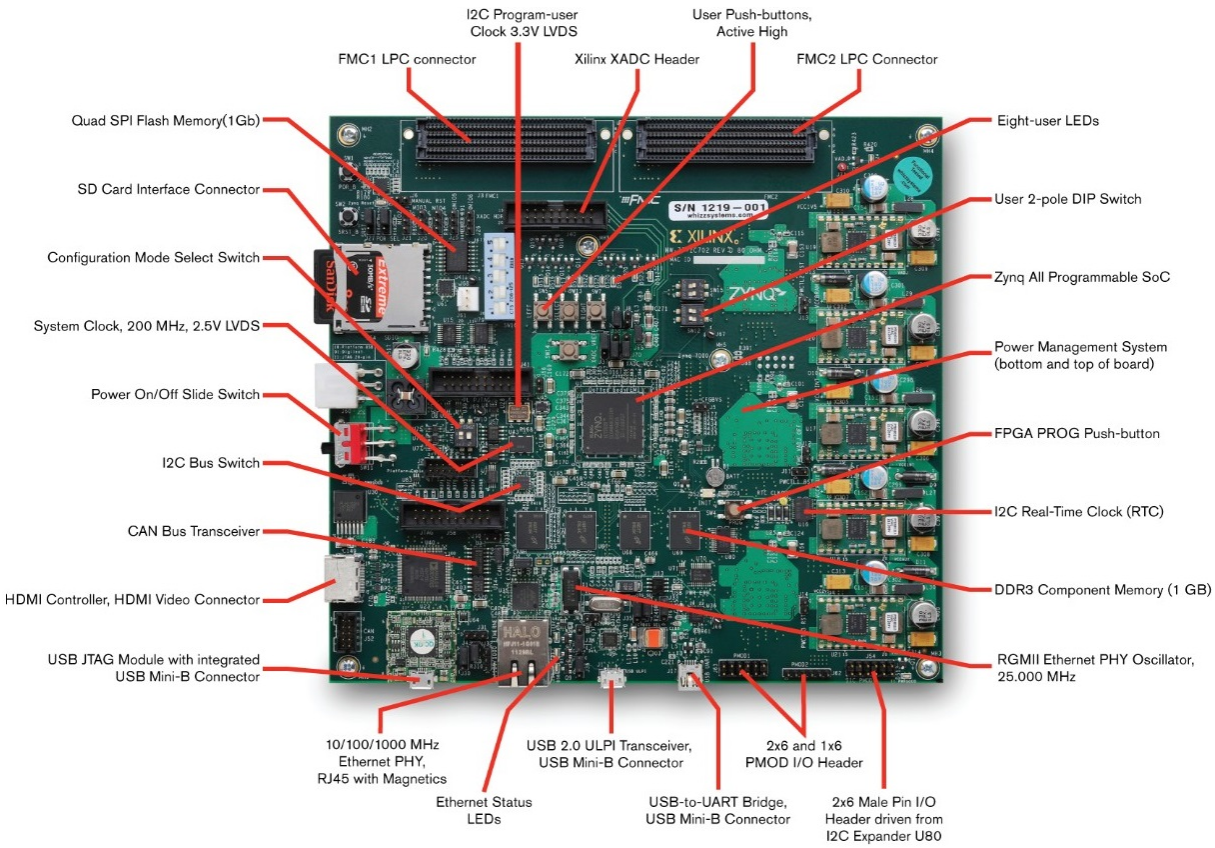

Figura 2: Xilinx ZYNQ-7000[1]

A família *Xilinx Zynq-7000* é um exemplo de uma SoC FPGA com um dual-core ARM Cortex-A9 integrado com lógica programável (figura 2) ou a Altera Arria V com um dual-core ARM Cortex-A9.

Os dispositivos FPGA são normalmente utilizados para acelerar a execução de algoritmos por hardware. Este conceito refere-se à utilização de hardware para realizar alguma das funções com mais eficiência energética e de desempenho do que seria possível realizar num processador genérico (*CPU - Central Processing Unit*). Alguns exemplos de aplicações são Inteligência artificial[9], processamento de som[15] e simulações de física[13].

A implementação de algoritmos e aplicações com software apresenta algumas vantagens comparativamente com o hardware. As vantagens típicas são o desenvolvimento rápido, a facilidade em alterar o código, bem como o baixo custo para tarefas genéricas, não recorrentes ou básicas, reduzindo a necessidade de adicionar novo hardware para além do CPU.

Por outro lado, a utilização de hardware dedicado na execução de algoritmos conduz, em geral, à implementação de sistemas com melhor desempenho e menor consumo energético. O elevado custo de projeto e a reduzida configuração pesam sobre a escolha de uma solução hardware dedicado em detrimento de uma solução unicamente software. Nestes aspetos, a configurabilidade da FPGA permite alterar o

hardware após fabrico e reduz os custos de projeto.

A possibilidade de reconfigurar o hardware apresenta uma grande flexibilidade, podendo executar várias tarefas para as quais inicialmente não foi projetado. Também possibilita o melhoramento ou ajustamento do hardware atual, para melhorar a capacidade do dispositivo na execução de determinadas tarefas.

As vantagens da FPGA no desenvolvimento de sistemas de processamento dedicados com elevado desempenho, comparativamente a um sistema de processamento genérico, levou a que se adotasse uma FPGA SoC para o desenvolvimento do LiteSAR. Esta solução permite a implementação de sistemas computacionais de elevado desempenho em ambientes com recursos operacionais limitados, como aeronaves e satélites.

1.2 Objetivos

Neste trabalho, pretende-se desenvolver um sistema (LiteSAR) de análise de imagens SAR em FPGA SoC de baixa densidade, para ser utilizado em ambientes com recursos computacionais limitados, tais como aeronaves e satélites.

O trabalho desenvolvido considerou a utilização de hardware dedicado e a sua integração com software, de forma a possibilitar o processamento das imagens SAR do radar em tempo-real e em sistemas embebidos de baixo custo.

O processamento de dados de SAR em tempo real permite intervalos mais pequenos de observação, possibilitando a monitorização de atividade dinâmicas.

1.3 Estrutura da Dissertação

A dissertação é constituída por sete capítulos e apresenta a seguinte estrutura:

Capítulo 1 - Este capítulo descreve os objetivos, as aplicações e uma breve introdução ao tópico do radar SAR e FPGAs.

Capítulo 2 - Este capítulo apresenta o estudo feito do estado atual da tecnologia em relação ao tópico da tecnologia SAR, *drones* de dimensões reduzidas e algoritmos.

Capítulo 3 - Este capítulo resume e explica os conceitos teóricos relacionados com a tecnologia SAR.

Capítulo 4 - Este capítulo apresenta o estudo do algoritmo e o planeamento realizado para o desenvolvimento do projecto.

Capítulo 5 - Este capítulo descreve a implementação do sistema LiteSAR, nomeadamente as técnicas e métodos utilizados no desenvolvimento da arquitetura.

Capítulo 6 - Este capítulo apresenta os resultados de desempenho e de ocupação de recursos do sistema desenvolvido.

Capítulo 7 - Este capítulo apresenta as conclusões da dissertação assim como as propostas para trabalhos futuros.

2. Estado da arte

A tecnologia SAR é atualmente utilizada em diferentes aplicações militares e civis, sendo a mais comum a criação de imagens da superfície terrestre.

Tem existido um amplo interesse recentemente no uso de pequenos *drones* para fotografia aérea. Estes possibilitam informações complementares e condições operacionais ampliadas em comparação com imagens óticas. O uso de drones portáteis de baixo custo para imagens de radar abre outras potenciais aplicações na monitorização científica, agrícola e ambiental.

As melhorias da tecnologia SAR e dos sistemas computacionais embebidos, juntamente com uma redução significativa do seu custo, resultou numa introdução forte de sistemas baseados em UAV (*Unmanned Aerial Vehicles*) para aplicações de deteção remota na última década [6].

UAV pequenos com menos de 3 kg estão a ser introduzidos na observação de terreno com SAR, evitando a necessidade de aeronaves de grande porte (especialmente para monitorizar áreas de tamanho reduzido).

Os requisitos para o sistema SAR dependem fortemente do tipo de plataforma na qual ele será instalado. No caso de um sistema de SAR montado em UAV, os requisitos mais importantes são o tamanho, o peso e o consumo de energia.

As plataformas UAV mais pequenas são mais suscetíveis a variações de trajetória devido a rajadas de vento ou turbulência, que podem ter um impacto na qualidade das imagens SAR criadas. Além disso, no processo de seleção do sistema SAR mais adequado, também deve ser tido em conta o seu preço.

Na aplicação do processamento digital de SAR, o processamento de sinais representa um desafio significativo devido ao seus requisitos de armazenamento de dados e computacionais. Um sistema SAR de alta resolução captura os ecos e produz uma quantidade enorme de dados brutos.

Algoritmos para processamento de sinais de radar, como o SAR, são computacionalmente intensivos e requerem tempos de execução consideráveis num processador.

Para superar estes problemas, tem-se verificado recentemente um grande desenvolvimento e otimização de algoritmos para geração de imagens SAR, bem como de plataformas computacionais.

Em [10] é proposta uma metodologia para aceleração de algoritmos. O trabalho utiliza o SAR como um caso de estudo para ilustrar o potencial de aceleração de um algoritmo SAR com dispositivos FPGA. Inicialmente, criam um perfil do algoritmo SAR e implementam um filtro homomórfico usando uma implementação de hardware do logaritmo natural. Os resultados experimentais mostram uma aceleração média de 188 vezes ao usar o acelerador de hardware baseado em FPGA ao invés de usar uma implementação de software executado num processador de uso geral.

Em [11] é realizado um estudo que compara o algoritmo Range Doppler Algorithm (RDA), o Chirp Scaling Algorithm (CSA) e o Omega-K Algorithm (Ω KA) para o processamento de SAR. O estudo foca-se na Transformada Rápida de Fourier (*Fast Fourier Transform* - FFT) no contexto do SAR, pois desempenha um papel importante nas operações de convolução, como a filtragem correspondente. A operação da FFT é accelarada com recurso à sua implementação em hardware juntamente com o restante sistema de processamento do SAR. O documento destaca o projeto e o desenvolvimento de uma unidade de pré-processamento baseada em FGPA para imagens de SAR em tempo real. Em particular um co-processador FFT eficiente. A arquitetura do co-processador proposta reduz significativamente os recursos de hardware, e ao mesmo tempo atinge uma velocidade de processamento elevada.

É feita uma revisão detalhada da literatura da implementação de FFT baseada em FPGA, o desempenho desses co-processadores é comparado em termos de 7 métricas. A partir da avaliação, os co-processadores propostos mostraram resultados promissores e satisfatórios na transformação de tempo-frequência .

Em [18] é apresentada uma arquitetura de alto desempenho que permite a execução eficiente e escalável do algoritmo de GBP (*Global Back-Projection*) numa FPGA.

O Algoritmo de GBP tem sido utilizado na formação de SAR ao longo dos últimos anos. Mas, apesar de muitas vantagens em comparação com outros algoritmos, o uso do GBP é limitado pois tem uma complexidade computacional significativa, com o consequente consumo indesejável de energia. A arquitetura do artigo é baseada num *systolic array* que realiza a *back-projection* num *pipeline* usando paralelismo sobre os pixeis da imagem e os pulsos do radar. Os núcleos utilizam um cálculo em vírgula flutuante na FPGA para obter um alcance preciso e altamente dinâmico na integração do GBP. Os resultados apresentados mostram o desempenho obtido da formação de imagem baseada em GBP utilizando uma FPGA detalhando o consumo de energia da implementação. As comparações são feitas com outras plataformas computacionais para determinar as vantagens para aplicações SAR em tempo real e com restrições de energia.

No trabalho proposto em [8] é implementado um sistema de múltiplos núcleos baseado em FPGA para o processamento de dados SAR com o algoritmo *Range-Doppler*.

O sistema descrito consiste em 4 dispositivos FPGA *Xilinx Virtex-6-550T* e vários dispositivos de memória fora da FPGA. É projetado um processador de 16 núcleos de processamento escalável que consiste em 16 elementos de processamento e uma rede 2D das FPGA. O sistema calcula dados brutos de SAR em tempo real com *pipeline* e pode adquirir imagens SAR de 256 níveis cinza contendo 2048x4096 pixeis em 12,03 segundos a uma frequência de 130MHz. O sistema consome cerca de 85 watts.

3. Radar de Abertura Sintética

O radar é montando tipicamente em plataformas móveis como satélites ou aeronaves, devido à necessidade de existir movimento entre o objeto e o radar de modo a ser possível criar a abertura sintética da antena. De forma geral, quanto maior a abertura, maior a resolução. Assim, podem criar-se imagens de alta resolução com antenas físicas pequenas[20].

3.1 Teoria Eletromagnética

O SAR é um radar ativo ao contrário dos radares mais comuns. Radares ativos emitem radiação para poderem receber o sinal refletido, enquanto os radares passivos não emitem radiação e apenas recebem radiação emitida por outros objetos.

O radar SAR emite radiação micro-ondas. A região do espectro eletromagnético com micro-ondas está contida entre 1 e 1000 GHz (figura 3), mas o SAR utiliza apenas um subconjunto destas frequências. Alguns exemplos destas bandas estão na tabela 1. As bandas apresentam vantagens diferentes para o uso em diversas aplicações.

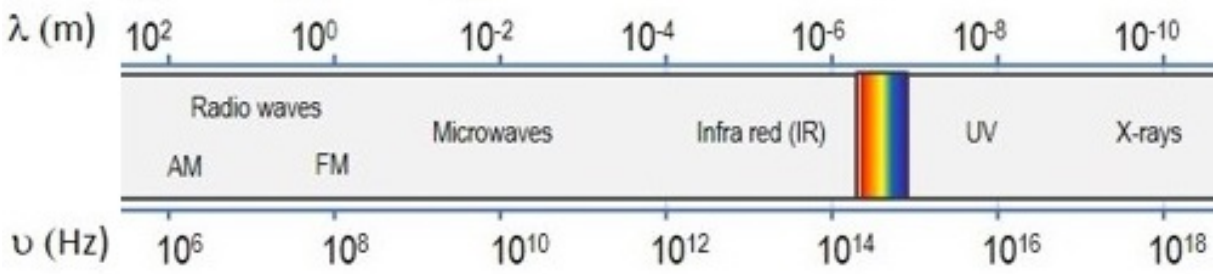

Figura 3: Espetro eletromagnético.

Banda	Frequência [Ghz]	Comprimento de onda [cm]
Ka	40–25	0.75–1.2
Ku	17.6–12	1.7–2.5
X	12–7.5	2.5–4
c	7.5–3.75	4–8
S	3.75–2	8–15
I	2–1	15–30
P	0.5–0.25	60–120

Tabela 1: Bandas comuns dos sistemas SAR

Alguns exemplos são a penetração da vegetação, imagens sub-superficial e estimativa de biomassa na banda P e L; Monitorização de agricultura, oceanos ou gelo na

banda L, C, S e X; Monitorização de neve na banda X e Ku; Imagens de alta resolução na banda X e Ka. As bandas mais frequentes de utilização são as bandas L, C e X. [14]

Os dois principais tipos de imagem de radar são o PPI (plan position indicator) circular e o *side-looking*. O SLAR (side-looking airborne radar) ou SLR (side-looking radar), apresenta *real aperture* e utiliza uma antena longa montada numa plataforma móvel (avião ou satélite). A antena emite pulsos eletromagnéticos perpendiculares à direção do trajeto da plataforma. Estes pulsos são direcionados ao solo, que depois são refletidos e dispersos em várias direções, incluindo a direção da antena. Estes ecos ou sinais refletidos, chegam à antena em tempos diferentes e com intensidades mais ou menos atenuadas, o que indica a refletividade dos objetos.

Os canais em que o SAR opera foram escolhidos devido à interação das ondas eletromagnéticas (micro-ondas) com a atmosfera e o solo. Estas escolhas devem-se às diferentes capacidades de penetração no solo e ao tamanho dos alvos.

3.2 Efeito Doppler (SAR)

O efeito Doppler é um fenómeno físico observado nas ondas quando são emitidas ou refletidas por um objeto que está em movimento em relação ao observador.

Quando existe movimento entre o recetor e o emissor, as ondas que neste caso são eletromagnéticas sofrem um desvio de frequência.

Na observação com um radar, dois objetos no solo separados na direção perpendicular ao trajeto da plataforma, também conhecido por direção azimute, apresentam diferentes ângulos em relação à antena. Esta diferença cria uma alteração na frequência em relação ao sinal enviado. Através do efeito de *Doppler* é possível calcular a distância do objeto.

Esta técnica permite o aumento da resolução azimute para o Radar de Abertura Sintética.

3.3 Resolução azimute

A resolução azimute é a capacidade do radar de separar dois refletores em alcances semelhantes, mas posições diferentes de um ponto de referência, ou seja, a largura da área do solo iluminada por cada pulso de radiação eletromagnética.

Para perceber o radar de abertura sintética é preciso primeiro perceber o radar com abertura real(RAR). A resolução do RAR é derivado da relação entre o tamanho físico da antena (a abertura real) e o comprimento de onda usado.

RAR

L_a = tamanho da antena
λ = largura do feixe de antena
R = distância à antena
δ_a = Resolução

$$\delta_a = \frac{\lambda}{L_a} R \tag{3.3.1}$$

É possível perceber que as antenas SLAR com abertura real não são possíveis de implementar devido ao tamanho necessário para atingir a resolução de azimute desejada. O SAR utiliza abertura sintética para superar este problema.

SAR

$L_a =$ tamanho da antena
λ= largura de feixe de antena
$v_{rel} =$ velocidade
R= distância à antena
δf_d = Resolução do desvio de Doppler.
δ_x = azimuth resolution

$$\delta_x = \frac{\lambda R}{2 v_{rel}} \delta f_d \tag{3.3.2}$$

$$\delta f_d = 1/t_{span} \tag{3.3.3}$$

$$t_{span} = \frac{R\lambda}{L_a v_{rel}} \tag{3.3.4}$$

$$\delta_x = \frac{L_a}{2} \tag{3.3.5}$$

Ao contrário da abertura real(equação 3.3.1), a abertura sintética obtém melhor resolução quanto menor for a antena(equação 3.3.5). A resolução azimute não depende da distância em relação à antena, o que torna este radar ideal para sistemas em satélites.

3.4 Compressão Azimute (SAR)

A distância do objeto ao recetor (*slant range*) varia aquando da passagem da plataforma com o radar SAR. Com a aproximação do alvo a distância diminui. Quando a plataforma está paralela ao alvo tem o mínimo de distância e à medida que se afasta do alvo a distância aumenta. A distância do radar ao alvo em função do tempo apre-

senta um hipérbole. Todos os pontos da curva da hipérbole representam um pixel na imagem final.

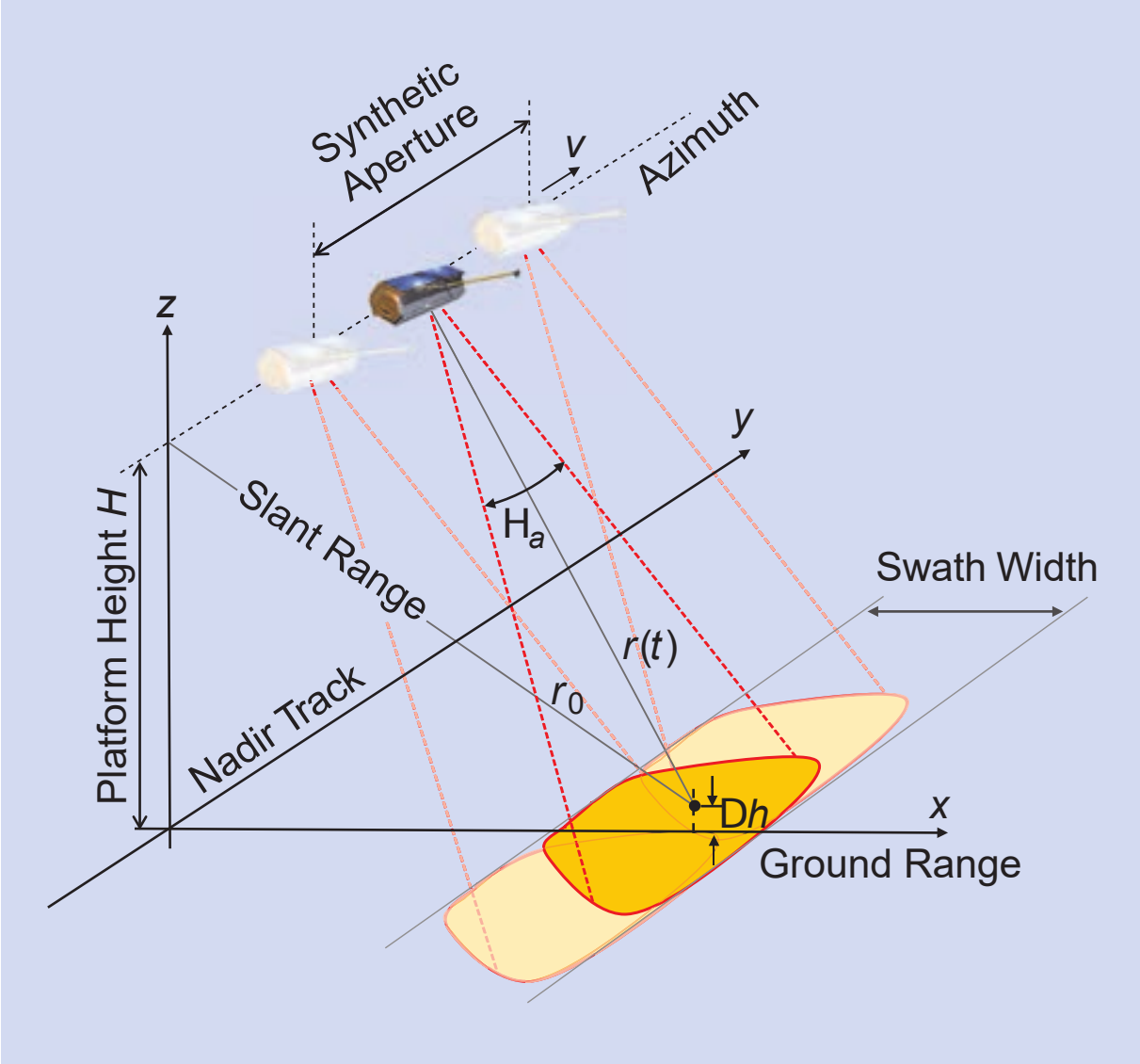

Figura 4: Geometria do SAR. [14]

3.5 Processamento SAR

Ao contrário dos sensores óticos, os dados sem processamento do SAR não oferecem nenhuma informação útil. O processamento de imagem envolve essencialmente duas convoluções no domínio do tempo com os dados do RADAR. A primeira é a convolução dos dados com uma função da distancia de referência e a segunda com uma função de referência azimute.

As convoluções do domínio do tempo são realizadas com multiplicações ponto a ponto no domínio da frequência para diminuir o peso computacional do processamento, isto porque a convolução no tempo requer uma operação complexa, enquanto que no domínio da frequência requer apenas multiplicações. Este processo é ilustrado

na figura 5[14].

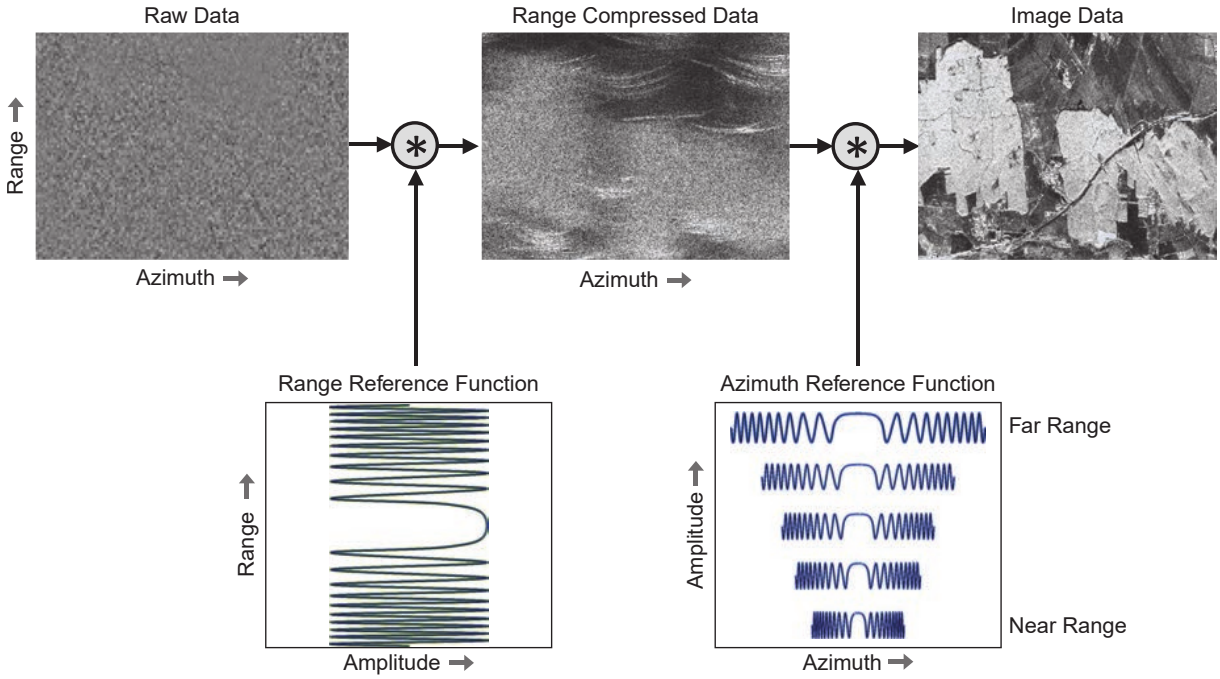

Figura 5: Ilustração do processamento SAR. [14]

As imagens SAR são representadas em termos de intensidade, de forma a cada pixel indicar a refletividade do ponto correspondente no solo. Isto envolve dois passos adicionais: a calibração e o *geocoding*.

A calibração garante que os valores representados tem intensidades correspondentes. Este processo de calibração não é simples, sendo necessário calibrar o equipamento interno e externo, através de objetos com índices de reflexão conhecidos[7].

O *geocoding* garante que a localização de todos os pixeis na imagem SAR, é diretamente associada com a posição no solo.

3.6 Modos de Imagem

Existem vários modos para captação de imagem através do controlo do padrão da radiação da antena, como está ilustrado na figura 6.

O modo fundamental é o *Stripmap* onde o padrão fixa o radar, criando a imagem, de uma faixa única contínua.

Para uma faixa mais larga, pode ser utilizado o modo *ScanSAR*. A elevação da antena é alterada para criar várias faixas. Este modo cria uma faixa mais larga, mas reduz a resolução azimute em comparação com o *Stripmap*.

Uma resolução melhor pode ser obtida utilizando o modo *Spotlight*. A antena fixa um ponto, para criar a imagem de uma região. Este modo reduz a faixa para uma região, mas aumenta a resolução.

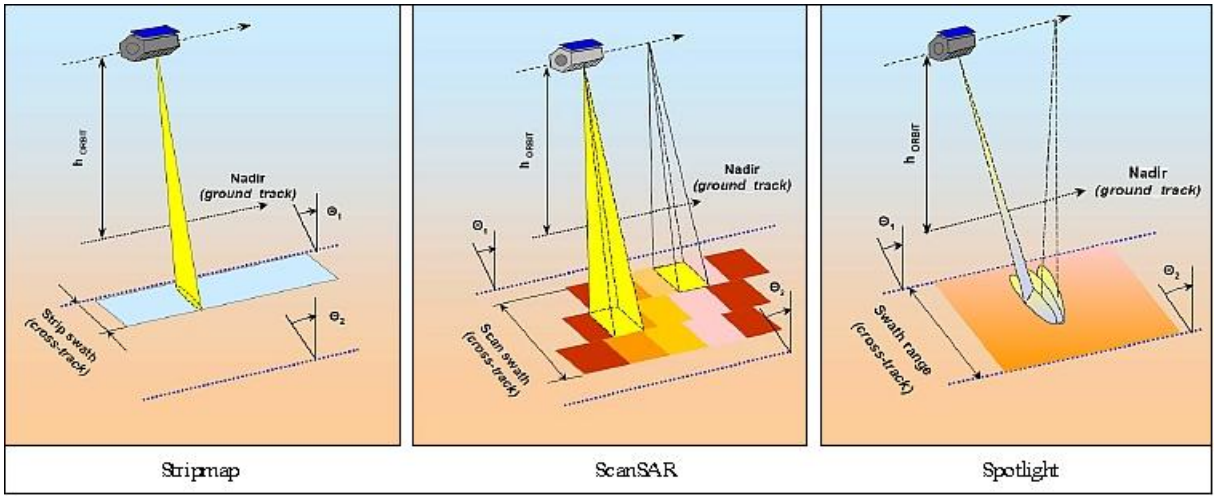

Figura 6: Geometria do SAR. [14]

4. Estudo do Algoritmo SAR - *Backprojection*

Os algoritmos para processamento deste tipo de sinais de radar são computacionalmente intensivos e requerem tempos de execução consideráveis numa unidade central de processamento (CPU) de uso geral.

Um sistema SAR de alta resolução captura uma quantidade enorme de dados brutos que precisam de ser processados para ter imagens percetíveis. Para superar este problema, existe o desenvolvimento de algoritmos para geração de imagens SAR, aplicações e otimizações.

4.1 Algoritmo *Backprojection*

Este método de formação de imagem integra diretamente a contribuição de cada pulso em cada pixel. Para cada pixel e cada pulso o algoritmo executa os passos seguintes [2]:

1. Calcula a distância entre a plataforma e o pixel.
2. Converte a distância para uma posição no conjunto de dados (ecos recebidos).
3. Calcula amostras usando interpolação linear utilizando a Equação (4.1.1)

$$g_{x,y}(r_k) = g(n) + \frac{g(n+1) - g(n)}{r(n+1) - r(n)} \times (r_k - r(n)) \tag{4.1.1}$$

   - $g(n)$ Amostra de onda na posição anterior.
   - $g(n+1)$ Amostra de onda na posição seguinte.
   - $r(n)$ Posição correspondente na posição anterior.
   - $r(n+1)$ Posição correspondente na posição seguinte.
   - r_k Distancia do pixel ao ponto de abertura(*aperture point*).

4. Dimensiona o valor amostrado por um filtro correspondente para criar a contribuição do pixel. Este valor é calculado usando a Equação (4.1.2). d_r é calculado usando a Equação (4.1.3).

$$e^{i\omega 2|\vec{r_k}|} = cos(2 \times \omega \times dr) + isin(2 \times \omega \times dr) \tag{4.1.2}$$

$$dr = \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} - r_c \tag{4.1.3}$$

   - dr distância da plataforma a cada pixel em relação ao centro.
   - x_k, y_k, z_k Localização da plataforma de radar em coordenadas cartesianas.

- x, y, z Localização do pixel em coordenadas cartesianas.
- r_c distância da plataforma de radar ao centro.

5. Acumula a contribuição no pixel. O valor final de cada pixel é dado pela equação (4.1.4).

$$f(x,y) = \sum_k g_{x,y}(r_k, \theta_k) \times e^{i\omega 2|\vec{r_k}|} \tag{4.1.4}$$

- $f(x,y)$ Valor de cada pixel (x,y)
- θ_k Ponto de abertura.
- r_k Distância do pixel $f(x,y)$ do ponto de abertura θ_k.
- ω Velocidade angular mínima da onda.
- $g_{x,y}(r_k, \theta_k)$ Onda refletida (calculado usando a interpolação linear)

4.2 Análise do Algoritmo de *Backprojection*

O algoritmo de *Backprojection* foi analisado com o objetivo de determinar qual a melhor forma de otimizar a sua execução sem comprometer a sua precisão. A análise foi feita sobre uma implementação do algoritmo *backprojection* em código C.

Através deste código é possível perceber que os cálculos dos pixeis não têm dependências entre si e por isso podem ser paralelizados.

4.2.1 Estudo da Precisão dos Dados

O estudo da variação das variáveis do algoritmo permite planear os formatos necessários para as representar e, consequentemente, o tipo de operadores aritméticos. Com este estudo é possível perceber a relação existente entre a precisão das variáreis, a qualidade de imagem, a ocupação de memória, entre outros.

De modo a reduzir a complexidade computacional, colocou-se como objetivo uma representação de vírgula fixa para todas as variáveis do algoritmo. A escolha do formato de vírgula fixa para o projeto é devido às características seguintes:

- A implementação dos circuitos aritméticos com vírgula fixa ocupa menos área do que os circuitos com vírgula flutuante, resultando numa ocupação de recursos mais reduzida e menor consumo de energia;
- Os cálculos em vírgula fixa requerem menos memória e menos tempo do processador para serem executados.

Existem limitações para este tipo de formato de dados que afeta o resultado final. Um número de virgula fixa não tem o mesmo intervalo de variação de números que é possível com vírgula flutuante para o mesmo número de bits da representação. A escolha da posição da virgula fixa limita a variação máxima da variável ou a precisão sendo por isso importante a escolha correta da posição da virgula de forma a permitir um equilíbrio entre a variação máxima, a precisão e o custo computacional.

Para minimizar a perda de informação de uma variável é preciso começar por garantir que a parte inteira do valor original vai ser representada no seu total, isto quer dizer que é necessário saber os limites dos valores originais. A perda de precisão da variável acontece na parte fracionária onde a redução do número de bits afeta a precisão da variável.

Com o estudo dos dados das simulações feitas em software foi possível determinar quais as variáveis que têm um maior impacto no resultado final, quantificado por uma métrica de relação sinal/ruído. A análise considera como ponto de referência mínimo uma relação sinal ruído (*SNR - Signal to Noise Ratio*) de 100dB para de obterem imagens razoáveis[3].

Devido à grande quantidade de variáveis do algoritmo, não é possível fazer um estudo exaustivo no qual se tenham todas as combinações possíveis de representação das variáveis. O algoritmo tem treze variáveis distintas que não só tem interferência no resultado final como também tem interferência entre si.

A precisão de cada variável foi inicialmente alterada individualmente, mantendo todas as outras em vírgula flutuante, de modo a determinar o valor que apresenta a melhora qualidade e o que conduz ao SNR mínimo de referência (100 dB). Com este método, é possível perceber o efeito de cada uma das variáveis no SNR da imagem final.

No estudo da precisão, consideraram-se as seguintes variáveis

platposX - Posição da plataforma no eixo X, x_k

platposY - Posição da plataforma no eixo Y, y_k

platposZ - Posição da plataforma no eixo Z, z_k

xdiff - Distancia entre a plataforma e o pixel no eixo X, $x - x_k$

ydiff - Distancia entre a plataforma e o pixel no eixo Y, $y - y_k$

zdiff - Distancia entre a plataforma e o pixel no eixo Z, $z - z_k$

bin - Distancia da plataforma a cada pixel, dr

w - Peso de interpolação, Int(bin)

matched.re - Parte real da amostra, $Re(g_{x,y}(r_k, \theta_k))$

matched.im - Parte imaginária da amostra, $Im(g_{x,y}(r_k, \theta_k))$

prod.re - Parte real do filtro correspondente, $Re(e^{i\omega 2|\vec{r_k}|}$

prod.im - Parte imaginária do filtro correspondente, $Im(e^{i\omega 2|\vec{r_k}|}$

pir - Ângulo utilizado na amostra, reduzido para o intervalo $[0, 2\pi[$

O maior valor de qualidade possível na imagem foi de SNR igual a 136,69dB. Os valores obtidos nas simulações são apresentados nas tabelas 2 e 3.

Bits Frac.	platposX	platposY	platposZ	xdiff	ydiff	zdiff
16	136.69	99.11	136.15	136.15	99.15	136.15
17	136.69	110.69	136.15	136.15	110.67	136.15
18	136.69	121.19	136.15	136.15	121.25	136.15
19	136.69	129.31	136.15	136.15	129.62	136.15
20	136.69	134.37	136.15	136.15	134.81	136.15

Tabela 2: Estudo SNR(Parte 1)

Bits Frac.	R / bin	w	matched.re	matched.im	prod.re	prod.im	pir
14		102.15					
15		107.98					
16	102.15	113.7	101.6	101.48			101.39
17	107.98	119.23	107.58	107.35			107.38
18	113.7	124.95	113.37	113.16			113
19	119.23	129.5	118.87	118.83			118.77
20	124.95	133.12	124.16	124.23	101.86	101.85	124.65
21	129.5	135.52	129.32	128.91	107.87	107.86	128.89
22	133.12	135.89	133.04	132.56	113.87	113.86	132.81
23	135.52		135.45	135.35	119.79	119.78	134.38
24	135.89				125.5	125.47	
25					130.42	130.4	
26					133.72	133.7	
27					135.2	135.2	

Tabela 3: Estudo SNR(Parte 2)

Os dados das tabelas 2 e 3 pode ser representados graficamente de forma a facilitar a sua compreensão.

Através do declive das curvas pode verificar-se o impacto das variáveis sobre a precisão do algoritmo. Quanto maior o declive do gráfico da variável, maior é seu impacto sobre a precisão.

Com o gráfico 7 é possível ver que algumas variáveis não têm grande impacto na qualidade da imagem final (p.ex., platposX, platposY e platporsZ). Também se verifica que, como era de esperar, o intervalo de variação do número de bits fracionários também varia. Com isto se conclui que para uma implementação mais eficiente se deve considerar uma representação dinâmica de vírgula fixa.

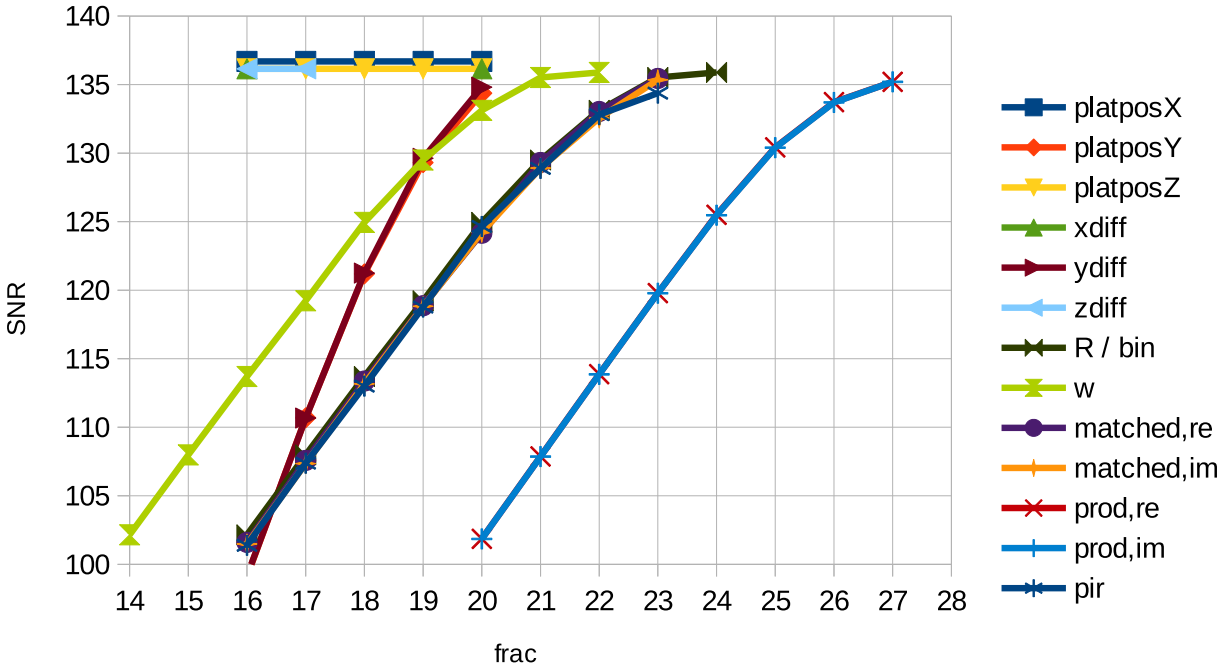

Figura 7: Valores de SNR relativos ao número de bits fracionários das diferentes variáveis

Conhecendo o intervalo de variação de cada uma das variáveis, converteram-se todas as variáveis para uma representação em vírgula fixa. O número de bits fracionários de cada uma determina a precisão do algoritmo. Existe uma grande correlação entre as variáveis, pelo que a variação da precisão de uma determina a variação de outras. Assim, é necessário explorar o espaço de representação das variáveis em conjunto. Face ao número de variáveis e ao intervalo de variação do número de bits fracionários de cada uma, propôs-se uma heurística de exploração de espaço de representações das variáveis.

O método baseia-se na atribuição de um peso para cada uma das variáveis que determina a influência do número de bits fracionários sobre a precisão do algoritmo. Este factor depende da operação a realizar sobre os dados e da localização da variável no fluxo de processamento do algoritmo. As variáveis iniciais têm um maior impacto sobre o erro, pois existe um caminho maior de propagação do erro (factor de propagação do erro - FPE). Adicionalmente, operações como adição geram um erro menor no resultado final do que em multiplicações. Além disso, o método também tem em conta o impacto que a redução da precisão tem sobre a implementação dos operadores. Uma redução do número de bits dos operadores de uma multiplicação leva a uma maior

redução dos recursos harwdare comparativamente, por exemplo, a uma soma.

A heurística proposta consiste em alterar as variáveis iterativamente, começando pelas que têm menor impacto no erro e maior impacto na implementação (ver figura 8). A heurística é executada até que a qualidade do resultado final atinja o ponto de referência de SNR igual a 100dB.

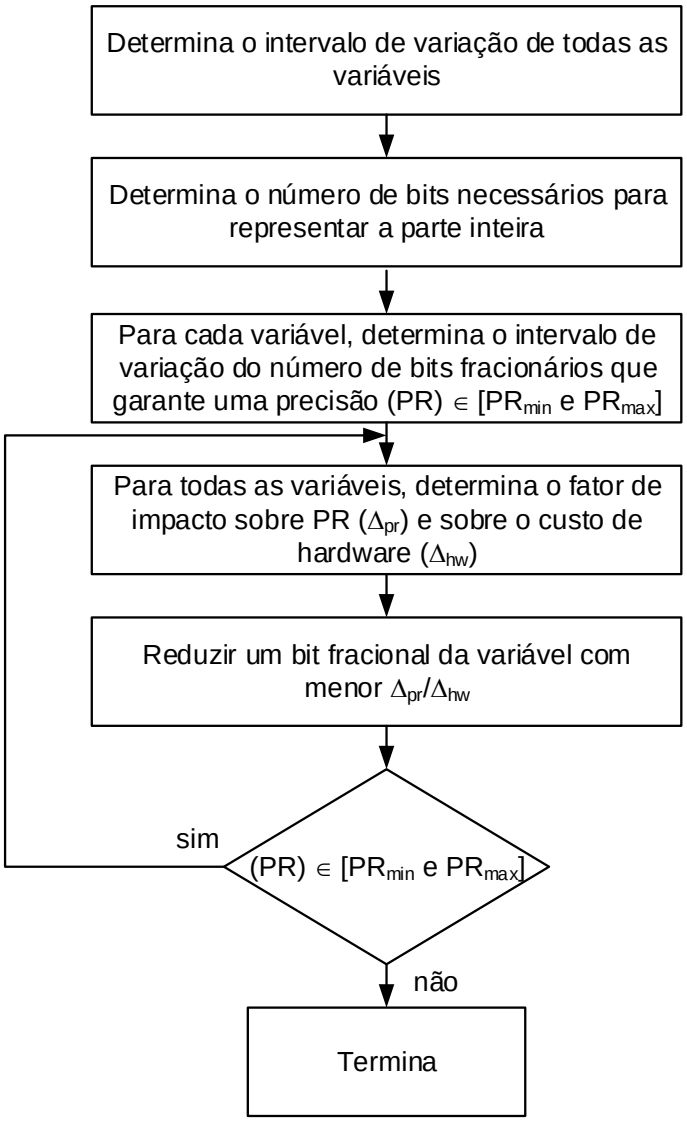

Figura 8: Heurística para determinação do formato vírgula fixa de cada uma das variáveis do algoritmo

A heurística proposta começa por determinar o intervalo de variação de cada uma das variáveis de forma independente. Com isto, determina o número de bits necessários para representar a parte inteira e a parte fracionária. De seguida, inicia um processo iterativo de variação do número de bits fracionários das variáveis. Em cada iteração e para cada variável determina a relação entre impacto sobre a precisão e custo de hardware. Reduz-se um bit à variável que tem o maior factor. O processo repete-se enquanto a precisão do algoritmo se mantiver dentro do intervalo permitido.

Os fatores utilizados para cada uma das variáveis são apresentados nas tabelas 4 e 3.

Operação	Factor
ADD	0.1
MUL	0.8
SQRT	2.0
TRIG	2.0

Tabela 4: Factor Operações

A tabela 4 descreve os factores de implementação considerados na heurística. Quanto maior o peso de implementação das operações, maior o factor. Procurou-se ainda estabelecer uma proporcionalidade dos factores que espelhe a complexidade relativa das operações.

Variável	FPE	Int	Frac(min)	Frac(max)
PTXe	0.455	13	16	21
PTYe	0.455	9	16	21
PTZe	0.455	13	16	21
XDIFFe	0.46	13	16	21
YDIFFe	0.46	10	16	21
ZDIFFe	0.46	10	16	21
XDSe	0.372	26	16	20
YDSe	0.372	18	16	20
ZDSe	0.372	26	16	20
Rvare	0.658	14	19	28
Wvare	0.752	0	14	22
SINvare	0.768	2	16	24
COSvare	0.768	2	16	24
MREe	0.776	2	16	24
MIMe	0.776	2	16	24
PREe	0.784	2	20	28
PIMe	0.784	2	20	28

Tabela 5: Factor Variáveis

A tabela 5 descreve os factores de propagação de erros (FPE) (quanto menor o valor, menor a variação das variáveis, logo menor o peso sobre o error final) e o intervalo de variação do número de bits fracionários da representação em vírgula fixa. Com os resultados do estudo da precisão das variáreis do algoritmo é possível converter o formato das variáveis no código original para o formato vírgula fixa a utilizar no projeto FPGA (vírgula fixa) de forma a otimizar o processamento das imagens do radar sem prejudicar a precisão do algoritmo.

4.2.2 Estudo do Fluxo de Dados do Algoritmo

O objetivo deste estudo é o de determinar a estrutura de ciclos do código que reduz os acessos à memória externa, a quantidade de memória interna e obtém o melhor tempo de processamento que não esteja limitado pelo acesso aos dados.

No estudo do fluxo de dados é necessário perceber as dependências entre variáveis assim como o tipo e quantidade de memória utilizada. O algoritmo de Backprojection é constituído por três ciclos principais, com algumas variáveis dependentes de outras. Os ciclos permitem percorrer todos os pulsos (P) constituídos por imagens de X por Y pixeis. Originalmente o código tem em memória uma quantidade de informação que não é possível ter na memória interna da FPGA. Devido a esta limitação é necessário reestruturar o código para reduzir a memória interna a utilizar na FPGA.

O algoritmo em software permite o processamento de imagens de entrada que podem apresentar dimensões diferentes. Neste trabalho, foram considerados três formatos de imagens de saída:

SMALL - Tamanho mais pequeno com imagens de saída de 512×512;

MEDIUM - Tamanho intermédio com imagens de saída de 1024×1024;

LARGE - Tamanho maior com imagens de saída de 2048×2048.

Na implementação em hardware é utilizado apenas o tamanho mais pequeno(SMALL).

Existem três grupos de dados que são necessários para a obtenção se uma imagem SAR. O de maiores dimensões são os dados obtidos dos pulsos a serem processados - *data*. Esta informação é aumentada (*upsample*) para melhorar a qualidade da interpolação. O tamanho do *data* em bytes é dado pela Equação (4.2.1).

$$data\_size = sizeof(complex) \times num\_data\_elements \tag{4.2.1}$$

O número de elementos (*num_data_elements*) é dado pela Equação (4.2.2).

$$num\_data\_elements = N\_PULSES \times N\_RANGE \times UPSAMPLE \tag{4.2.2}$$

em que *N_PULSES* representa o número de pulsos, *N_RANGE* é a dimensão da imagem de saída e *UPSAMPLE* é o factor de aumento da imagem.

O segundo conjunto de dados, designado *platpos* diz respeito aos dados da posição da plataforma de imagem para cada pulso. O tamanho destes dados em bytes é dado pela Equação (4.2.3).

$$platpos\_size = sizeof(position) \times N\_PULSES \tag{4.2.3}$$

em que *position* é um conjunto de três coordenadas em formato vírgula flutuante de dupla precisão dado pela Equação (4.2.4).

$$sizeof(position) = 3 \times sizeof(double) \tag{4.2.4}$$

O terceiro conjunto é o das variáveis relacionadas com os parâmetros da plataforma de imagem. O tamanho destes dados em bytes é dado pela Equação (4.2.5).

$$Const\_size = 5 \times sizeof(float) \tag{4.2.5}$$

Na tabela 6 são apresentadas as dimensões dos parâmetros para cada uma das dimensões de imagem de saída.

Parâmetro	Variável	SMALL	MEDIUM	LARGE
Pulsos	P	512	1024	2048
Tamanho da imagem	$Nx \times Ny$	512 x 512	1024 x 1024	2048 x 2048
Tamanho da imagem (bytes)		256K	1M	4M
Factor de aumento		8	8	8
Número de pulsos após aumento	NBP	4096	8192	16384
Dimensão do *data* (bytes)		2M	8M	32M

Tabela 6: Parâmetros de *INPUT*

Para otimizar a estrutura de memória, é necessário alterar a ordem pelo qual são percorridos os pulsos de forma a alterar o acesso aos dados. Dependendo da ordem dos ciclos no algoritmo é possível ter diferentes características. Considerando os ciclos em X, Y e P, os casos possíveis para organização dos ciclos são:

XYP

Os pixeis são percorridos por linhas e por cada pixel é acumulado a contribuição de todos os pixeis. Esta ordem precisa de percorrer todos os dados por cada pixel que processa. Para percorrer os dados é necessário ter o *array* em memória ou então transferir um elemento do *data* sempre que se passa ao próximo pulso, ou seja, transfere $N\_RANGE \times UPSAMPLE$ elementos por cada pulso incrementado. Esta solução limita o desempenho da arquitetura pois o tempo necessário para a transferência é superior ao tempo necessário para processar. Como exemplo, temos a imagem SMALL que teria de ler 4096 elementos para poder processar um pulso. Os pixeis de saída poderiam ser transmitidos 1 a 1 visto que os pixeis são calculados de forma independente.

YXP

Os pixeis são percorridos por colunas e por cada pixel é acumulado a contribuição de todos os pixeis. Esta solução não apresenta nenhuma diferença da solução anterior apenas a ordem alterada de linhas para colunas.

PXY

Os pulsos são percorridos por colunas de forma a calcular cada contribuição do pulso para cada pixel. Esta ordem permite processar a informação à medida que é obtida possibilitando, dependendo de outros fatores, guardar a informação de apenas um pulso. Esta solução necessita de guardar a contribuição de cada pulso em todos os pixeis o que resulta na memória da acumulação ter o tamanho da imagem final.

PYX

Os pulsos são percorridos por linha de forma a calcular cada contribuição do pulso para cada pixel. Esta solução não apresenta nenhuma diferença da solução anterior apenas a ordem alterada de colunas para linhas.

YPX

Cada linha é processada de forma a produzir os pixeis, ou seja, a imagem final é produzida linha a linha. Esta ordem permite uma memória de acumulação reduzida pois cria a imagem linha a linha, o que permite a transmissão dos pixeis e a libertação de memória.

O acesso aos dados precisa de transferir um elemento por cada X elementos processados. Esta é a ordem utilizada no projeto da arquitetura hardware.

XPY

Cada coluna é processada de forma a produzir os pixeis, ou seja, a imagem final é produzida coluna a coluna. Esta solução não apresenta nenhuma diferença da solução anterior apenas a ordem alterada de linhas para colunas.

4.2.3 Otimização da Estrutura de Ciclos

A ordem de execução dos ciclos é assim alterada de forma a sobrepor a transferência de dados com o processamento, ou seja, os dados relativos ao próximo ciclo de processamento são carregados, enquanto o processamento dos dados atuais é executado.

Uma vez que os dados de entrada foram aumentados de um factor de 8, para cada pulso teremos de transferir 8 vezes mais dados do que os que são processados.

Com esta limitação é possível obstruir o acesso à memória.

Para equilibrar os tempos de processamento e de comunicação é necessário subdividir o ciclo dos Y (*tiling*). O tempo mínimo necessário para carregar os dados relativos a cada pulso é superior ao processamento do ciclo X para um determinado pulso. Para balancear os tempos, considerou-se a execução de vários Y com os mesmos dados relativos a um determinado pulso.

Considerando, por exemplo, uma imagem de 512×512 e factor de aumento de 8, a ordem pelo qual são percorridos os ciclos é alterada de 512Y-512P-512X para 64Y 512P 8Y 512X. O Y é dividido em grupos de forma a não desperdiçar a capacidade de

processamento.

Supondo que por cada ciclo de relógio é processado um pixel e que por cada ciclo é lido um elemento da data é possível obter a equação do número de ciclos de transferência de dados por *tilling, t,* $Comm_{cy}[t]$, e o número de ciclos de processamento por *tilling*, $Exec_{cy}[t]$ (ver Equações (4.2.6) e (4.2.7))

$$Comm_{cy}[t] = \left(\frac{X \times UPSAMPLE \times sizeof(dataelement)}{BW} \right) \tag{4.2.6}$$

$$Exec_{cy} = X \tag{4.2.7}$$

em que BW representa a largura de banda de acesso à memória externa onde estão alojados os dados, em bytes/ciclo.

Para otimizar a sobreposição do tempo de computação e de transferência de dados, a relação entre os dois tempos deve ser unitária. O *tilling* ótimo é assim determinado pela Equação (4.2.8).

$$\textit{Tiling} = \frac{X \times UPSAMPLE \times sizeof(dataelement)}{BW \times X} \tag{4.2.8}$$

Considerando, como exemplo, que X = 512, $UPSAMPLE$ = 8, BW = 8 e sizeof(data element) = 8, teríamos que o *Tiling* ótimo é igual a 8.

Neste casos, os ciclos em Y seriam agrupados de 8 em 8. Este valor não depende do tamanho da imagem, apenas do valor de aumento da imagem $RANGE\_UPSAMPLE$, da largura de banda de acesso à memória externa, BW, e do tamanho de cada elemento de dados. Assume-se que um ciclo de transferência ocorre à mesma frequência que um ciclo de processamento.

4.2.4 Paralelismo

A estrutura de ciclos e a independência no cálculo dos pixeis de saída, permite a paralelização do algoritmo em hardware. Para criar um processamento mais rápido é possível ter múltiplos fluxos de dados iguais a trabalhar em diferentes Grupos de Y. O acesso à memória com o *data[p]* é igual para todas os fluxos de dados assim como a memória do *platpos*.

Tendo em conta o factor de paralelização do algoritmo, é possível estimar o tempo de execução, e o aumento na memória necessária. Para todos os cálculos é desprezado o atraso inicial de enchimento do pipeline do caminho de dados que é desprezável face ao tempo total de processamento assim como o tempo necessário para a transferência dos dados processados.

Considerando um fator de paralelismo, NP, e uma estrutura Y P Yg X, ou seja, com a ordem YPX e um *tiling* de 8, resulta num tempo de processamento, $Tproc_N P$

dado por:

$$Tproc\_NP = \frac{Tproc\_1}{NP} \tag{4.2.9}$$

em que $Tproc\_1$ é o tempo de processamento com apenas um caminho de dados, dado por:

$$Tproc\_1 \approx X \times Y \times N\_PULSES \times Freq \tag{4.2.10}$$

O tempo de processamento diminui de forma linear à custa de utilizar mais hardware, mas este paralelismo tem um limite, MAX_NP dado por:

$$MAX\_NP = \frac{Y}{tiling} \tag{4.2.11}$$

Com estas equações e a tabela 6 é possível obter a tabela 7 que apresenta o máximo paralelismo possível.

Parâmetro	Variável	SMALL	MEDIUM	LARGE
Pulsos	P	512	1024	2048
Tamanho da imagem	X x Y	512 x 512	1024 x 1024	2048 x 2048
Max_NP, tiling = 8	MAX_NP	64	128	256
Tproc_NP		1/64	1/128	1/256
Tproc_NP com atraso		1/64+delay	1/128+delay	1/256+delay

Tabela 7: Valor máximo de paralelismo para um *tiling* de 8 e para diferentes tamanhos de imagem

A memória interna do dispositivo alvo também determina o factor de paralelismo da arquitetura. A memória utilizada para guardar o platpos pode ser partilhada por todos os caminhos de dados em paralelo. No entanto, a memória dos dados por pulso, apesar de ser a mesma para todos os caminhos de dados em paralelo, pode ter de ser acedida em posições diferentes por todos os circuitos em paralelo. Em consequência, são necessários $2 \times NP$ portos de acesso à memória. Na FPGA, uma memória apenas suporta até dois portos em paralelo. Como tal, é necessário replicar a informação para ter mais portos de acesso.

Também existe a possibilidade de utilizar uma memória com velocidade de acesso superior aos outros blocos no circuito, possibilitando um ritmo de acesso à memória superior ao de processamento, logo capaz de alimentar mais circuitos de processamento em paralelo. Este método não foi considerado neste trabalho.

A memória mínima em número de elementos, necessária para armazenar os dados de entrada do algoritmo de Backprojection, $Mdata$, o *platpos*, $Mplatpos$, e o acumula-

dor, $Macc$, é calculada por:

$$Mdata = N\_PULSES \times RANGE\_UPSAMPLED \times 2 \tag{4.2.12}$$

$$Mplatpos = N\_PULSES \tag{4.2.13}$$

$$Macc = X \times tiling \times 2 \tag{4.2.14}$$

O valor total da memória em bytes depende da dimensão de cada elemento dos dados guardados.

$$Ny/GroupY\_optimal * transf\_delay \tag{4.2.15}$$

Os recursos necessários para paralelizar o bloco de processamento de imagens *SAR* são apresentado na tabela 8.

Parâmetro	Variável	SMALL	MEDIUM	LARGE
Pulsos	P	512	1024	2048
Tamanho da imagem	Nx x Ny	512 x 512	1024 x 1024	2048 x 2048
Max_NP, Tiling = 8	MAX_N	64	128	256
Mdata		4096 x N	8192 x N	16384 x N
Mplatpos		512	1024	2048
Macc, NP = 1		4096 x N	8192 x N	16384 x N
Macc, NP = 2		8192 x N	16384 x N	32768 x N

Tabela 8: Recursos de memória com *tiling* = 8 e várias dimensões da imagem

A escolha do fator de *tiling* e de paralelismo depende dos recursos disponíveis da FPGA alvo e da largura de banda de acesso à memória externa.

5. Arquitetura Hardware para Execução do *Backprojection*

O presente capítulo descreve a arquitetura desenvolvida para a execução do algoritmo de Backprojection em FPGA. A implementação tem em conta os resultados da análise de precisão elaboradas no capítulo anterior. Começa-se por descrever o caminho de dados, seguido do controlo e dos parâmetros de configuração da arquitetura.

5.1 Caminho de Dados do Acelerador

Este bloco armazena e processa a informação da memória interna. O objetivo no desenvolvimento do bloco é a redução de tempo de processamento através de técnicas como a segmentação de instruções (pipeline) e através da exploração do paralelismo.

Todos os blocos do circuito são configuráveis, tornando a arquitetura configurável de modo a permitir escolher o número de bits de cada uma das variáveis do algoritmo.

Na implementação de blocos aritméticos mais complexos, como os blocos de cálculo das funções trignométricas e de raiz quadrada, são utilizados os geradores automáticos de módulos aritméticos existentes na ferramenta de síntese.

O fluxograma do caminho de dados do circuito desenvolvido é descrito no grafo da figura 9.

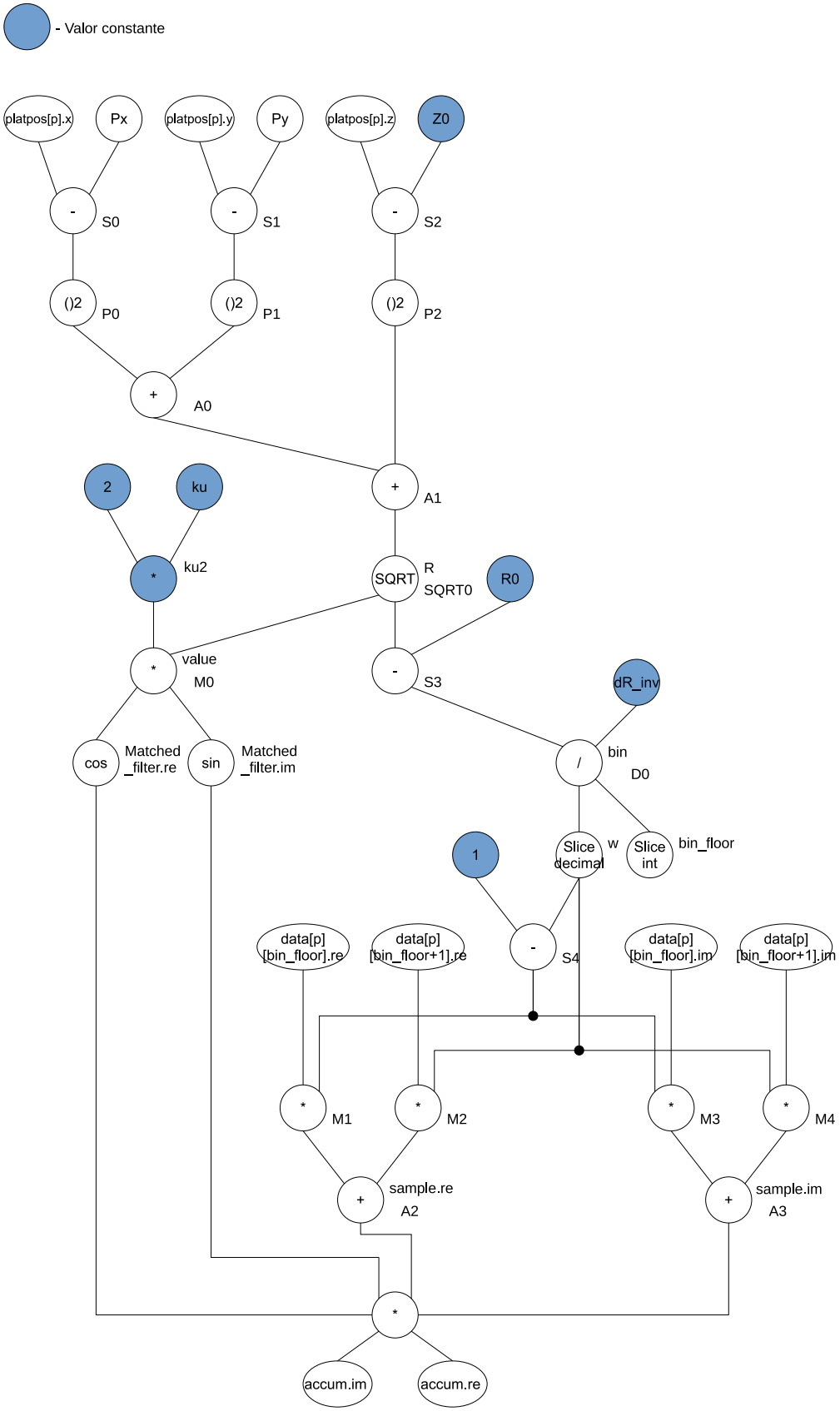

Figura 9: Grafo fluxo de dados do caminho de dados do acelerador LiteSAR

A implementação do fluxo de dados foi dividida em blocos funcionais, sendo cada

um deles implementado com um bloco lógico (ver figura 10) que se descrevem de seguida:

Calcular distância O primeiro bloco calcula a distância entre a plataforma e o pixel;

Conversão da distância O segundo bloco converte a distância entre a plataforma e o pixel para uma posição no conjunto de dados recebidos;

Interpolação linear O terceiro bloco calcula as amostras usando interpolação linear;

Filtro O quarto bloco calcula o filtro;

Cálculo do Pixel Este bloco determina a contribuição dos dados para os pixeis através da multiplicação entre a interpolação pelo filtro;

Acumulação dos Pixeis O bloco final acumula a contribuição dos dados de todos os pulsos.

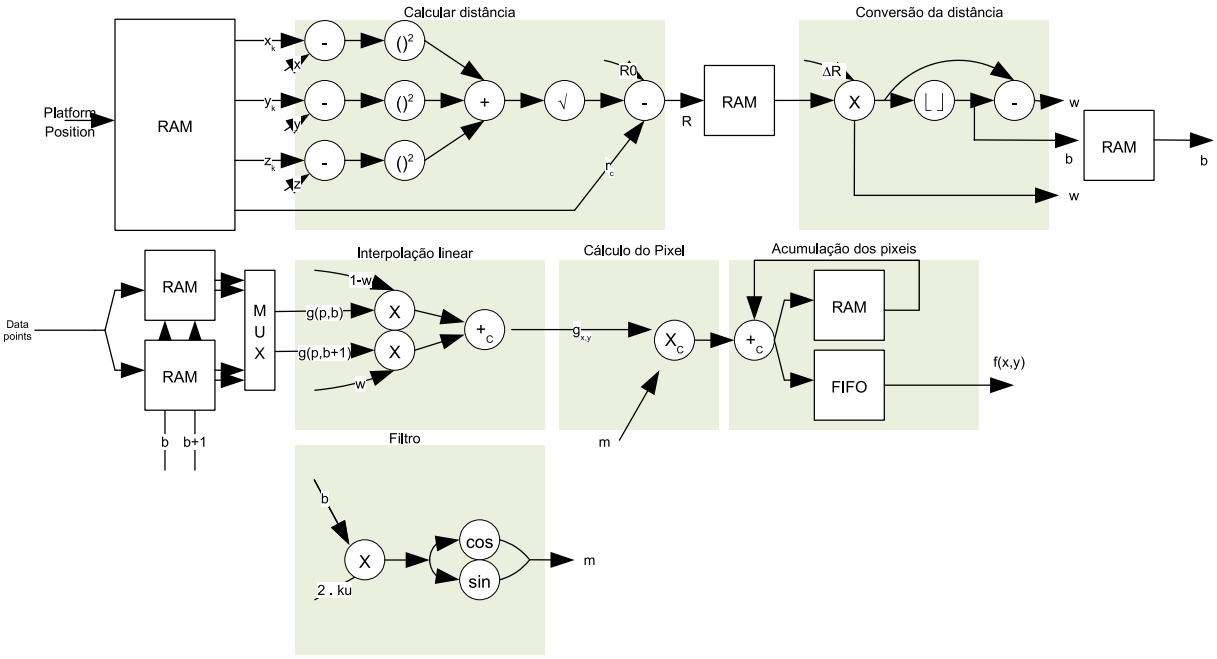

Figura 10: *Core SAR*

5.1.1 Calcular distância

Este bloco calcula a distância entre a plataforma e o pixel com recurso a somadores, multiplicadores e o IP do Vivado, CORDIC.

Nesta parte do *backprojection*, é calculada a distância entre a plataforma e o pixel. Os pontos são representados em coordenadas cartesianas(X,Y,Z) e a distancia entre os dois pontos, plataforma e o pixel, é calculado através da equação 5.1.1.

$$dr = \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} \tag{5.1.1}$$

O grafo da figura 11 representa a estrutura implementada neste bloco.

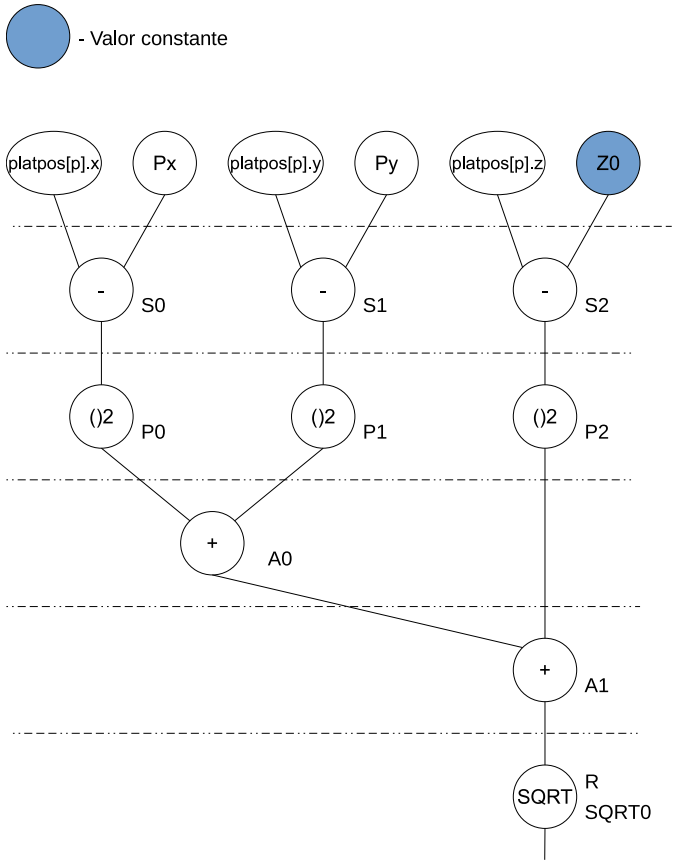

Figura 11: Calcular distância

O bloco utilizado para a função raiz quadrada é o bloco disponibilizado pelo Vivado. O valor de entrada não tem sinal e o valor de saída é truncado, ambos os valores são limitados de 8 a 48 bits. Para ser possível utilizar este IP com valores de virgula fixa é necessário à saída fazer um *shift* na posição da virgula dado pela equação 5.1.2.

$$Y = 2^{(-N-1)/2} \times \sqrt{X_{alt}} \tag{5.1.2}$$

Onde Y é o valor desejado, X_{alt} é o valor de saída do bloco e N é o número de bits da parte fracionária.

5.1.2 Conversão da distância

Este bloco converte a distância entre a plataforma e o pixel para uma posição no conjunto de dados. O módulo utiliza multiplicadores e subtratores.Neste parte do algoritmo é realizada a função aritmética dada pela equação 5.1.3.

$$bin = (R - R_0)/dr \tag{5.1.3}$$

O grafo da figura 12 representa a estrutura implementada neste bloco.

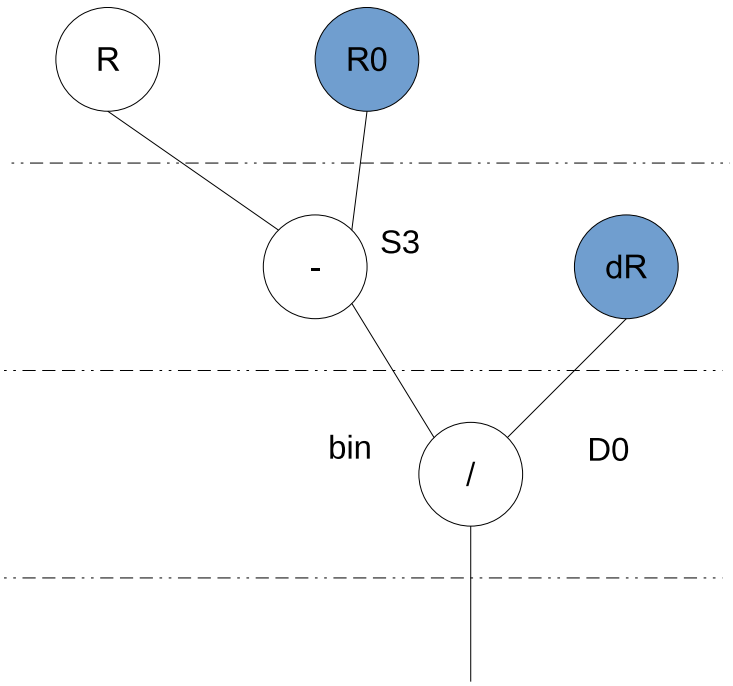

Figura 12: Conversão da distância

A implementação da operação de divisão requer um circuito lógico mais complexo e, por este motivo, a menos que seja estritamente necessário são utilizadas outras alternativas. Para evitar a divisão, uma vez que dr é constante, foi substituída por uma multiplicação. A constante dr é alterada para dr_{inv} em que o valor é dado por $dr = 1/dr_{inv}$, calculado estaticamente.

5.1.3 Interpolação linear

Este bloco calcula as amostras usando interpolação linear. Este é constituído por somadores, subtratores, multiplicadores e dois blocos de memória.

O bloco calcula a interpolação linear como representado nas equações 5.1.4 e 5.1.5. Como entrada tem os dados carregados na memória interna. As amostras que resultam desta operação contêm parte imaginária e parte real.

$$sample.re = (1 - w) \times data[p][bin_{floor}].re + w \times data[p][bin_{floor} + 1].re \quad (5.1.4)$$

$$sample.im = (1 - w) \times data[p][bin_{floor}].im + w \times data[p][bin_{floor} + 1].im \quad (5.1.5)$$

O grafo da figura 13 representa a estrutura implementada neste bloco.

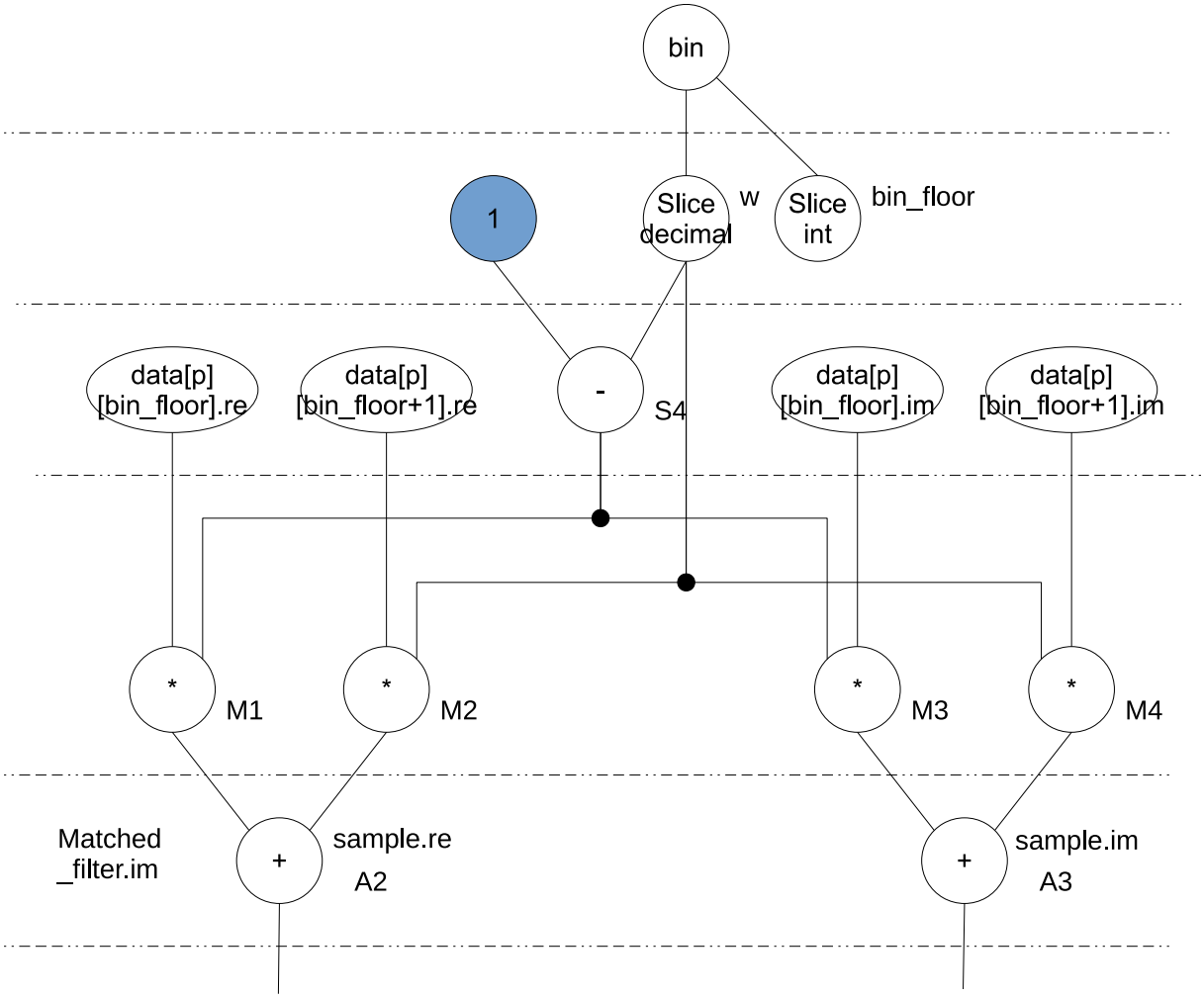

Figura 13: Interpolação linear

O acesso aos dados do pulso(data[p]) requer a leitura de quatro variáveis: os endereços $bin_{floor} + 1$ e bin_{floor} e as partes reais e imaginárias. Para evitar duas leituras para a parte real e imaginária, os dados são concatenados, em que os bits de maior peso representam a parte real e os de menor peso a parte imaginária. Com uma leitura é lido o número complexo.

O uso de duas memórias permite um acesso a duas posições diferentes ao mesmo tempo aumentado assim a quantidade de leituras em simultâneo. Estas memórias são carregadas com a mesma informação em simultâneo.

Com a aplicação destas duas técnicas é possível reduzir o número de leituras para duas leituras em paralelo.Para impedir diminuir o tempo de espera do carregamento da informação na memória é utilizado *double buffering* dentro das memórias, esta técnica consiste em ter o dobro de espaço na memória utilizando metade para a leitura e a outra metade para a escrita de forma alternada.

5.1.4 Filtro

Este bloco calcula o filtro com recuso a multiplicadores,*multiplexer* , blocos de arredondamento, somadores e o núcleo do Vivado CORDIC.

O bloco disponibilizado pelo Vivado funciona como uma função seno e cosseno. A entrada deste bloco é em radianos sem sinal e a saída é truncada. O intervalo de entrada está contido entre $[-\frac{\pi}{4}, \frac{\pi}{4}]$.

O grafo da figura 14 representa a estrutura implementada neste bloco.

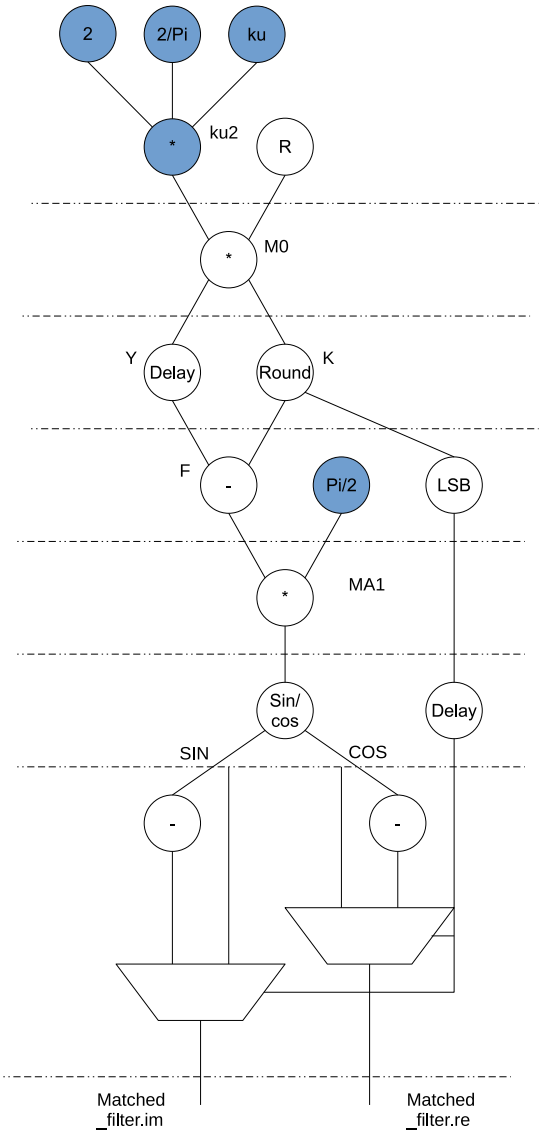

Figura 14: Filtro

Os valores de entrada precisam de ser convertidos para o intervalo de entrada através da redução argumento. A redução para o intervalo de $[-\pi/4, \pi/4]$ é dada pela equação 5.1.6.

$$x = k(\pi/2) + r \tag{5.1.6}$$

O k é um valor inteiro e $|r| \leq \pi/4$. O valor n é dado pelos dois bits de menor peso da variável k. O valor de saída é dado pela tabela 9.

n	sin(x)	cos(x)	sin(x)/cos(x)
0	sin(r)	cos(r)	sin(r)/cos(r)
1	cos(r)	-sin(r)	-cos(r)/sin(r)
2	-sin(r)	-cos(r)	sin(r)/cos(r)
3	-cos(r)	sin(r)	-cos(r)/sin(r)

Tabela 9: Redução do argumento

Multiplicando $2/\pi$ pela equação 5.1.6 obtemos:

$$x(2/\pi) = k + r(2/\pi) \tag{5.1.7}$$

O $|r| \leq \pi/4$ dando $r \times (2/\pi) \leq 0.5$ resultando na equação:

$$y = x + (2/\pi) \tag{5.1.8}$$

O valor k é o valor y arredondado para um valor inteiro.

$$k = |y| \tag{5.1.9}$$

$$f = y - k \tag{5.1.10}$$

$$r = f \times (\pi/2) \tag{5.1.11}$$

O artigo [16] que descreve estas equações apresenta uma nota na qual indica que estas equações não podem ser utilizadas com vírgula flutuante, especialmente se o exponente do argumento for grande. Isto é devido ao erro de arredondamento do $2/\pi$ ser aumentado pela subtração 5.1.10.

Apesar de este aviso ser dado para aritmética em vírgula flutuante também tem importância para a aritmética com vírgula fixa. O erro dado pela redução da precisão do $2/\pi$ tem um efeito significativo no resulta final. Para evitar erros maiores, é utilizado uma precisão superior na constante que contem o $2/\pi$ resultando no possível aumento de tempo de processamento no bloco.

5.1.5 Cálculo do Pixel

Este bloco determina a contribuição dos dados para os pixeis com recuso a multiplicadores e somadores.

Para calcular a contribuição dos dados para os pixeis é necessário multiplicar a interpolação pelo filtro. O bloco realiza uma multiplicação complexa representada na equação 5.1.12.

$$(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i \tag{5.1.12}$$

O grafo da figura 15 representa a estrutura implementada neste bloco.

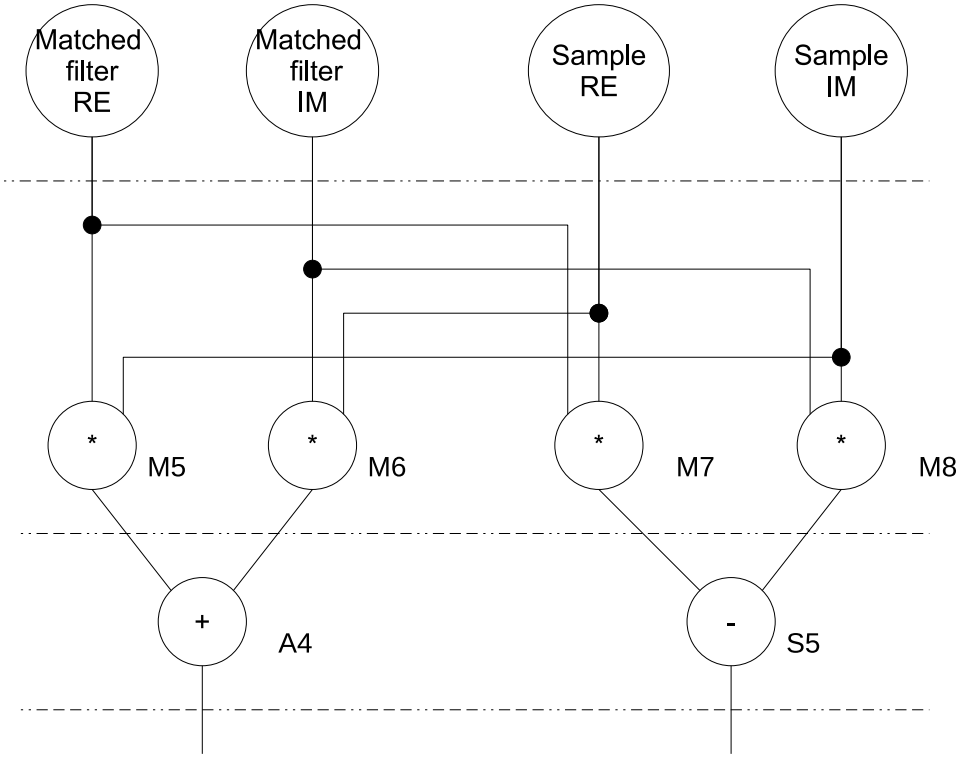

Figura 15: Cálculo do pixel

5.1.6 Acumulação dos pixeis

O bloco de acumulação é constituído por somadores e blocos de memória. Para a acumulação o bloco precisa de dois somadores para a parte real e imaginária dos dados, assim como um bloco de memória RAM para armazenar as acumulações. Na primeira iteração, o acumulador é carregado com zero e nas seguintes faz a acumulação com o novo valor.

Para ser possível não ter de esperar pela transferência do acumulador para a memória externa é possível utilizar duas memórias. Este processo utiliza o dobro da memória mas reduz o tempo de espera. Apenas uma das memórias precisa de

endereços visto que a outra pode servir apenas para guardar os dados para a transferência.

O *trade-off* desta situação permite escolher entre o dobro da memória no acumulador em troca de uma diminuição no tempo necessário para processar os dados de

5.2 Controlo do Acelerador

Este bloco tem como objetivo controlar o circuito de processamento, assim como as memórias e as comunicações com a memória externa. O controlo é constituído por blocos memórias e máquinas de estado. A escolha de memórias para este bloco depende principalmente no tipo de acesso necessário. No caso do acesso ser sequencial, os blocos de memoria são FIFOs (*first in, first out*), onde o primeiro elemento a entrar na memoria é o primeiro elemento a ser lido. No caminho de dados existem duas FIFO para armazenar as variavéis BIN, R e existe duas FIFO para armazenar os dados processados a serem enviados para memória externa e outra para o controlo dos dados válidos. O controlo esta dividido em 3 blocos.

PL_loader O primeiro bloco controla a comunicação com a memória externa para a transferência do *Platpos* e escreve os dados nas memórias internas;

U1 O segundo bloco controla a primeira parte do caminho de dados, a comunicação de saída de dados, e a transferência dos dados de entrada;

U2 O terceiro bloco controla a segunda parte do caminho de dados.

O bloco **PL_loader** controla a transferência do *Platpos* para as memórias internas. Este bloco contem três contadores, o contador1 que controla a entrada do contador2 de forma ao conjunto contar de até três N_PULSES vezes para guardar as três componentes XYZ do *platpos* que contém N_PULSES pulsos e saber quando os dados das constantes estão no bus de dados. O ultimo contador, contador3, serve para saber quando as constantes foram transferidas.

O bloco **U1** controla a primeira parte do caminho de dados até à conversão da distância, coordena o processamento em ping-pong com o terceiro bloco e controla a comunicação dos dados com a memória externa.

O bloco **U2** controla a segunda parte do caminho de dados até à acumulação dos pixeis.

5.3 Configuração do acelerador

Para ser possível a utilização deste bloco com outras imagens ou parâmetros de entrada diferentes, bem como com outros formatos de vírgula fixa, foram utilizados parâmetros configuráveis, enumerados nas tabelas 10 e 11.

Nome Descrição	
N Range Upsampled	$N\_RANGE\_UPSAMPLED$
Cst P	Tamanho em bits das variáveis Platpos , Px, Py e
Cst P Frac	Parte fracionária de Cst P
Cst Diff	Tamanho em bits das variáveis xdiff, ydiff, zdiff
Cst Diff Frac	Parte fracionária de Cst Diff
Cst Diff2	Tamanho em bits das variáveis xdiff2, ydiff2, zdiff2
Cst Diff2 Frac	Parte fracionária de Cst Diff2
Cst W	Tamanho em bits da variável W
Cst W Frac	Parte fracionária de Cst W
Cst Bin	Tamanho em bits da variável Bin
Cst Bin Frac	Parte fracionária de Cst Bin
Cst Data	Tamanho em bits da Data
Cst Data Frac	Parte fracionária de Cst Data
Cst Mw	Tamanho em bits da variável de saida de M1
Cst Mw Frac	Parte fracionária de Cst Mw
Cst M	Tamanho em bits das variaveis de saida de Matched e Sample
Cst M Frac	Parte fracionária de Cst M
Cst Mm	Tamanho em bits da variável de saida de Matched e Sample
Cst Mm Frac	Parte fracionária de Cst Mm
Cst Complex	Tamanho em bits da variável Prod
Cst Complex Frac	Parte fracionária de Cst Complex
Cst R	Tamanho em bits da variável R
Cst A1 O	Tamanho em bits da variável de saida de A1. A parte inteira de A1 tem de ser impar
Cst A1 O Frac	Parte fracionária de Cst A1 O
Cst Ma1 O	Tamanho em bits da variável de saída de Ma1
Cst Ma1 O Frac	Parte fracionária de Cst Ma1 O
Cst M0	Tamanho em bits da variável de M0
Cst M0 Frac	Parte fracionária de Cst M0
Cst Cos Sin	Tamanho em bits da variável do bloco SINCOS
Cst Xcountervalue	Valor Nx
Cst Xcountersize	Número de bits necessários para o contador X
Cst Ycountervalue	Valor Ny
Cst Ycountersize	Número de bits necessários para o contador Y
Cst Ycounter Group	Valor N, tamanho do grupo de y
Cst Ycounter Group Size	Número de bits necessários para o contador Ygroup
Cst Pcountervalue	Valor Pulses
Cst Pcountersize	Número de bits necessários para o contador P

Tabela 10: Parâmetros do acelerador (parte 1)

Nome Descrição	
Cst Bram Platpos Addra Size	Numero de bits de endereços da memória Platpos
Cst R Size	Tamanho da fifo R
Cst Bin Size	Tamanho da fifo Bin
Cst Accum Size	Tamanho da accumulação da linha (Nx)
Cst Accum Addrsize	Numero de bits de endereços do accumulador
Initvaluex	Valor $(-BP_N PIX_X/2.0 + 0.5 * dxdy)$
Initvaluey	Valor $(-BP_N PIY_X/2.0 + 0.5 * dxdy)$
Cst Bram Data Latency	Numero de clock necessário para ler da memória Data
Cst Bram Platpos Latency	Numero de clock necessário para ler da memória Platpis
Cst Bram Data Addra Size	Numero de bits de endereços da memória Data
Cst Data Size	Numero de endereços da memória Data
Cst Data Countersize	Cst Bram Data Addra Size-1
Cst Const	Numero de bits da constante com maor tamanho
Cst 4kupi Frac	Parte fracionaria da constante 4*ku*pi em bits

Tabela 11: Parâmetros do acelerador (parte 2)

5.4 Sistema LiteSAR

O sistema LiteSAR é constituído por seis blocos representados na figura 16.

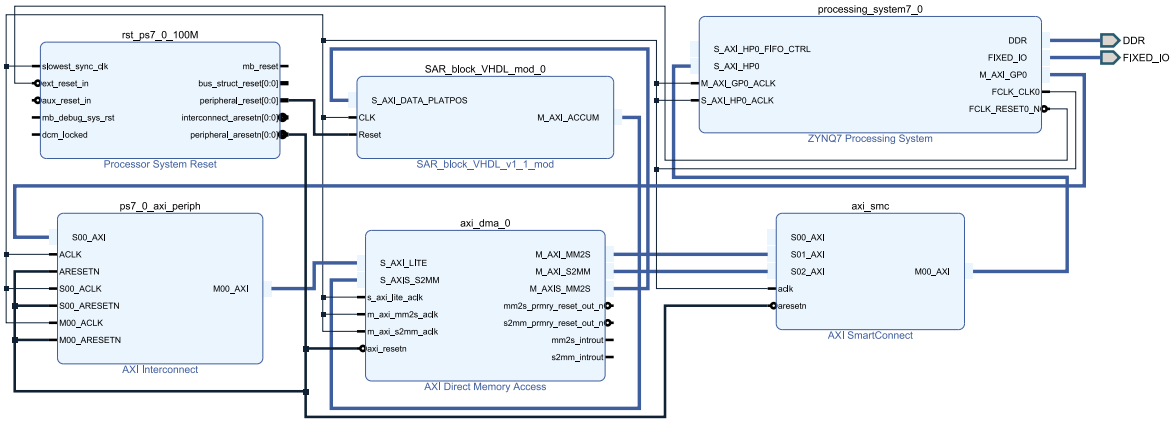

Figura 16: *Arquitetura do SoC de execução do algoritmo SAR*

ZYNQ7 Processing System Processador ARM que controla o acelerador e configura a arquitetura;

Processor Reset System Sistema que controla os sinais de *resets*.

AXI Direct Memory Access O circuito de acesso direto à memória (DMA - *Direct Memory Access* que permite acesso a memória do sistema para transferências diretas entre a memória externa e o acelerador;

AXI SmartConnect e AXI Interconnect Núcleos que interligam um ou mais módulos da arquitetura com uma interface AXI;

SAR block VHDL Bloco dedicado que processa os dados e devolve a imagem em grupos de linha.

5.4.1 Advanced eXtensible Interface - AXI4

A interface AXI faz parte do ARM Advanced Microcontroller Bus Architecture(AMBA) é uma interface disponível gratuitamente para a conexão e gestão de blocos num SoC. Este tipo de interface apresenta as seguintes características:

- Endereço / controle separado e fases de dados;
- Suporte para acessos de dados desalinhados;
- Transferências baseadas em rajadas, com uma única transmissão do endereço inicial;
- Canais de leitura e escrita separados e independentes;
- Suporte para transmissões pendentes;
- Suporte para conclusão de transmissões fora de ordem para transações com IDs de encadeamento diferentes na mesma porta mestre;
- Suporte para operações atómicas.

O *Advanced eXtensible Interface* define um mecanismo de *handshake* composto por dois sinais, o *Valid* e o *Ready*.

O sinal $VALID$ é operado pelo bloco que envia para informar a entidade de destino que os dados são válidos e podem ser lidos. O sinal $READY$ é operado pelo bloco recetor para notificar que está preparado para receber dados. Quando os dois sinais de *handshake* estiverem ativos no mesmo ciclo de relógio dá-se a transferência de dados. Este mecanismo permite controlar o fluxo de dados, acelerando ou desacelerando a velocidade de transmissão.

6. Resultado

Este capitulo apresenta os resultados de recursos hardware e o desempenho da implementação do LiteSAR na SoC FPGA ZYNQ7020. O acelerador foi desenvolvido em VHDL (*"VHSIC Hardware Description Language"*) e o projeto foi feito no *Vivado Design Suite* da Xilinx. O software executado no ARM foi desenvolvido SDK (*Software Development Kit*) integrado na ferramenta Vivado. O Vivado dispõem de uma biblioteca de geradores de módulos hardware que foi utilizada na geração de alguns dos módulos da arquitetura, como as memórias, as operações trigonométricas e a raiz quadrada.

A tabela 12 apresenta os dados de utilização de recursos FPGA obtidos após o mapeamento e encaminhamento do circuito na FPGA. A tabela indica os recursos utilizados em cada bloco, incluindo o acelerador SAR, e o total de recursos do sistema completo integrado com o ARM.

Os valores da tabela 12 são retirados da implementação do sistema LiteSAR com os parâmetros configurados para a maior precisão.

Name	LUT	FF	BRAMs	DSP	LUTRAM
total	10822 (20.34 %)	14605 (13.73 %)	76.00 (54.29 %)	73 (33.18 %)	924 (5.31 %)
system axi smc	2696	3905	0	0	815
SAR block	6560	8556	67.00	73	331
system rst	19	40	0	0	1
processing system	112	0	0	0	0
axi DMA	2006	2674	9.00	0	216
system auto pc	428	564	0	0	65

Tabela 12: Utilização de recursos da FPGA

A aplicação de um factor de paralelismo não foi possível pois a quantidade de memória necessária é superior aos recursos de memória disponíveis na FPGA.

Com a execução do sistema SAR no acelerador desenvolvido foi possível atingir um tempo de processamento de cerca de 1 FPS utilizando uma frequência de 127MHz.

O valor do tempo de processamento necessário foi tirado da execução do algoritmo na FPGA tendo em conta o processo de transmissão dos dados tanto de entrada como de saída. O valor exato obtido neste processo foi de $1.19s$.

Esta implementação tem como resultado uma imagem com um valor de SNR de 119.72dB. Apesar de ser um valor elevado este valor é inferior ao previsto na simulação de software onde a imagem final tem 134dB. Este valor tem um erro de

cerca de 14dB. O erro é propagado devido à precisão dos blocos e a forma como estes são implementados. O erro do caminho percorrido pelos dados é amplificado devido à acumulação dos pixeis, ou seja, qualquer erro é multiplicado no final pelo número de pulsos.

7. Conclusões e Trabalho Futuro

Este capitulo apresenta as conclusões ao trabalho desenvolvido, assim como as propostas para trabalhos futuros.

7.1 Conclusões

Este trabalho de projeto tem como base o desenvolvimento de um sistema de processamento de imagens SAR obtidas por uma plataforma radar numa plataforma computacional com recursos operacionais limitados.

Este sistema processa as imagens em tempo-real de forma a que as decisões tomadas com base nas imagens possam ser feitas junto ao sistema de recolha da imagem.

Um sistema SAR de alta resolução captura uma quantidade enorme de dados brutos que precisam de ser processados.

O estudo da variação das variáveis do algoritmo permitiu planear os formatos necessários para as representar e, consequentemente, o tipo de operadores aritméticos. O estudo permite perceber a relação existente entre a precisão das variáreis, a qualidade de imagem, a ocupação de memória, entre outros.

A ordem pelo qual são percorridos os pulsos pode ser alterada de forma a modificar o acesso aos dados. Dependendo da ordem, é possível ter diferentes características. Na implementação utilizada, cada linha é processada de forma a produzir os pixeis, esta ordem permite uma memória de acumulação reduzida. Esta ordem pode ser subdividida de forma a não desperdiçar a capacidade de processamento.

A utilização de paralelização do algoritmo em hardware, que não foi implementada no projecto, permite criar um processamento mais rápido com múltiplos fluxos de dados iguais a trabalhar em diferentes grupos de linhas. Esta técnica permite a diminuição do tempo de processamento, mas aumenta os recursos necessários do hardware.

O aspeto principal deste projecto é a utilização de técnicas de aceleramento do processamento dos dados e os compromissos para obter uma implementação que processa as imagens em tempo real. Este compromissos resultam na escolha de características mais úteis para o hardware final.

Como foi dito nos resultados, as diferentes implementações dos blocos aritméticos podem causar diferenças nos resultados em comparação com a execução em software.

O sistema LiteSAR teve uma taxa de processamento de imagens de cerca de 1 FPS. Esta velocidade de processamento de imagens foi possível ser atingida com a implementação do sistema numa FPGA ZYNQ7020 a uma frequência de relógio de

127MHz.

7.2 Trabalho Futuro

Com a elaboração do projeto foi possível identificar algumas formas de melhorar o trabalho.

Para simplificar o bloco controlo e eliminar a memória intermédia entre os blocos U1 e U2 é possível considerar um fluxo de dados único.

O resultado final apresenta uma diferença no SNR que provém das aproximações do caminho de dados. Esta diferença pode ser estudada de forma a produzir um resultado mais semelhante ao resultado do algoritmo software.

Referências

[1] Xilinx zynq-7000 soc zc702 evaluation kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html. Accessed: 2021-09-30.

[2] Helena Cruz. On-board multi-core fault-tolerant sar imaging architecture. Technical report, Instituto Superior Técnico, 2018.

[3] Helena Cruz, Rui Policarpo Duarte, and Horácio Neto. Fault-tolerant architecture for on-board dual-core synthetic-aperture radar imaging. In *International Symposium on Applied Reconfigurable Computing*, pages 3–16. Springer, 2019. DOI: 10.1007/978-3-030-17227-5_1.

[4] L. Cutrona, E. Leith, C. Palermo, and L. Porcello. Optical data processing and filtering systems. *IRE Transactions on Information Theory*, 6(3):386–400, 1960. DOI:10.1109/TIT.1960.1057566.

[5] Lc J Cutrona, WE Vivian, EN Leith, and GO Hall. A high-resolution radar combat-surveillance system. *IRE Transactions on Military Electronics*, 1051(2):127–131, 1961. DOI:10.1109/IRET-MIL.1961.5008330.

[6] Jurgen Everaerts et al. The use of unmanned aerial vehicles (uavs) for remote sensing and mapping. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(2008):1187–1192, 2008.

[7] Anthony Freeman. Sar calibration: An overview. *IEEE Transactions on Geoscience and Remote Sensing*, 30(6):1107–1121, 1992. DOI:10.1109/36.193786.

[8] Ning Hou, Duoli Zhang, Gaoming Du, and Yukun Song. An fpga-based multi-core system for synthetic aperture radar data processing. In *Anti-counterfeiting, Security, and Identification (ASID), 2014 International Conference on*, pages 1–4. IEEE, 2014. DOI:10.1109/ICASID.2014.7064956.

[9] Joo-Young Kim. Fpga based neural network accelerators. In *Advances in Computers*, volume 122, pages 135–165. Elsevier, 2021. DOI:10.1016/bs.adcom.2020.11.002.

[10] Youngsoo Kim, William Harding, Clay S Gloster Jr, and Winser Alexander. Acceleration of synthetic aperture radar (sar) algorithms using field programmable gate arrays (fpgas). In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 271–271. ACM, 2015. DOI:10.1145/2684746.2689125.

[11] YC Lee, VC Koo, and YK Chan. Design and development of fpga-based fft co-processor for synthetic aperture radar (sar). In *Progress in Electromagnetics Research Symposium-Fall (PIERS-FALL), 2017*, pages 1760–1766. IEEE, 2017. DOI:10.1109/PIERS-FALL.2017.8293422.

[12] A Love. In memory of carl a. wiley. *IEEE Antennas and Propagation Society Newsletter*, 27(3):17–18, 1985. DOI:10.1109/MAP.1985.27810.

[13] Anne K Madsen, M Scott Trimboli, and Darshika G Perera. An optimized fpga-based hardware accelerator for physics-based ekf for battery cell management. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020. DOI:10.1109/ISCAS45731.2020.9181073.

[14] Alberto Moreira, Pau Prats-Iraola, Marwan Younis, Gerhard Krieger, Irena Hajnsek, and Konstantinos P Papathanassiou. A tutorial on synthetic aperture radar. *IEEE Geoscience and remote sensing magazine*, 1(1):6–43, 2013. DOI=10.1109/MGRS.2013.2248301.

[15] Erdem Motuk, Roger Woods, Stefan Bilbao, and John McAllister. Design methodology for real-time fpga-based sound synthesis. volume 55, pages 5833–5845, 2007. DOI:10.1109/TSP.2007.898785.

[16] Kwok C Ng. Argument reduction for huge arguments: Good to the last bit. *Unpublished draft, available from the author (kwok. ng@ eng. sun. com)*, 1992.

[17] Leonard J Porcello. An experimental study of rapid phase fluctuations induced along a satellite-to earth propagation path. Technical report, Michigan Univ Ann Arbor Inst Of Science And Technology, 1964.

[18] Dan Pritsker. Efficient global back-projection on an fpga. In *Radar Conference (RadarCon), 2015 IEEE*, pages 0204–0209. IEEE, 2015. DOI:10.1109/RADAR.2015.7130996.

[19] Shivakumar Ramakrishnan, Vincent Demarcus, Jerome Le Ny, Neal Patwari, and Joel Gussy. Synthetic aperture radar imaging using spectral estimation techniques. *Advanced Signal Processing*, page 65, 2002.

[20] George W Stimson. Introduction to airborne radar, ed, 2014.

[21] Carl A Wiley. Synthetic aperture radars. *IEEE Transactions on Aerospace and Electronic Systems*, AES-21(3):440–443, 1985. DOI:10.1109/TAES.1985.310578.