UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ISADORA FERNANDA ZAPPE SCHMIDT

IMPLEMENTAÇÃO DO ALGORITMO DE AUTOCORRELAÇÃO EM VHDL PARA MEDIÇÃO DE VELOCIDADE

TOLEDO

2022

ISADORA FERNANDA ZAPPE SCHMIDT

IMPLEMENTAÇÃO DO ALGORITMO DE AUTOCORRELAÇÃO EM VHDL PARA MEDIÇÃO DE VELOCIDADE

Implementation of autocorrelation algorithm in VHDL for speed measurement

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Fábio Rizental Coutinho.

TOLEDO

2022

ISADORA FERNANDA ZAPPE SCHMIDT

IMPLEMENTAÇÃO DO ALGORITMO DE AUTOCORRELAÇÃO EM VHDL PARA MEDIÇÃO DE VELOCIDADE

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 12/dezembro/2022

_____

Fábio Rizental Coutinho
Doutor em Engenharia Elétrica e Informática Industrial
Universidade Tecnológica Federal do Paraná

_____

Felipe Walter Dafico Pfrimer
Doutor em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

_____

Tiago Piovesan Vendruscolo
Doutor em Engenharia Elétrica e Informática Industrial
Universidade Tecnológica Federal do Paraná

TOLEDO

2022

RESUMO

Determinar as distribuições de velocidade ao longo da seção transversal de um canal fluvial é de vital importância para detalhar as possibilidades do seu potencial energético. Visto isso, o presente trabalho implementou, em VHDL, um algoritmo de estimação de velocidade baseado no método de autocorrelação, o qual é embasado na estimativa de fase para pulsos sequenciais de um sinal complexo demodulado. Para isso, utilizou-se de um algoritmo previamente validado no MATLAB® em uma FPGA, empregando um *kit* DE10-Lite e o ponto flutuante no padrão IEEE 754, com precisão simples. Por sua vez, os testes foram realizados com dados reais adquiridos anteriormente a este trabalho. Ao final, o algoritmo de autocorrelação implementado em VHDL apresentou erro relativo de velocidade estimada abaixo de 0,00002%, porém, devido ao baixo erro, uma grande quantidade de recursos é consumida, inviabilizando a utilização em FPGAs com menor capacidade.

Palavras-chave: *ultrassom*; *efeito Doppler*; *medição de vazão*; *lógica reconfigurável*.

ABSTRACT

Determining the velocity distributions along the cross section of a river channel is of vital importance to detail the possibilities of its energy potential. Considering this, the present work implemented in VHDL a velocity estimation algorithm based on the autocorrelation method, which is established on the phase estimation for sequential pulses of a complex demodulated signal. An algorithm previously validated in MATLAB® was applied in a FPGA using a DE10-Lite kit and the IEEE 754 standard floating point with single precision. The tests were performed with real data acquired prior to this work. The autocorrelation algorithm implemented in VHDL presented a relative estimated velocity error below 0.00002%, however, due to the low error, a vast amount of resources is consumed thus making it unfeasible to be used in FPGAs with lower capacity.

Keywords: *ultrasound*; *Doppler effect*; *flow measurement*; *reconfigurable logic*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

f_0	Frequência original
f_r	Frequência refletida
f_{dop}	Frequência de desvio Doppler
v	Velocidade
c	Velocidade do som
θ	Ângulo entre o feixe de ultrassom e a direção do observador em movimento
t	Tempo
n	Emissões
r	Sinal complexo em fase e quadratura
$R(\,)$	Função de autocorrelação
N	Pares de emissões
r^*	Complexo conjugado
$\hat{R}(\,)$	Função de autocorrelação média
$frac$	Fração
s	Bit de sinal
e	Expoente não-polarizado
$1.frac$	Mantissa

SUMÁRIO

1 INTRODUÇÃO

A medição da velocidade com que a água flui em um rio permite obter a vazão ou descarga do canal fluvial. Ademais, realizar a caracterização da vazão espacial (na direção da profundidade) de forma periódica viabiliza a aplicação de técnicas para prever seu comportamento com o intuito de elaborar e expandir estudos do seu potencial energético e também prever o transporte de sedimentos e contaminantes em seu fluxo (PEKTAŞ, 2015).

Por volta de 1992, um grande interesse em instrumentos acústicos e ópticos não-intrusivos que aplicam o efeito Doppler para medição de velocidade em águas rasas foi despertado (MUSTE et al., 2004). Dentre eles, se destacaram os instrumentos ultrassônicos do tipo *Ultrasound Velocity Profiler* (UVP), também denominados *Acoustic Doppler Current Profiler* (ADCP), os quais emitem pulsos de ultrassom na água a fim de observar os efeitos Doppler nos ecos das partículas microscópicas suspensas nela, que servem de rastreadoras do movimento. Isso possibilita medir o perfil espacial da vazão ou corrente de um canal fluvial.

Em 2020, o Prof. Dr. Fábio Rizental Coutinho obteve fomento para o desenvolvimento de um medidor do tipo UVP para ser aplicado no monitoramento de rios. Intitulado “Mini UVP para monitoramento ambiental de rios por VANTs anfíbios”, o projeto visa desenvolver o *hardware* e o *software* de um medidor UVP miniaturizado de forma que possa ser embarcado em um Veículo Aéreo Não-Tripulado (VANT) anfíbio, também conhecido como drone anfíbio. No *software* desse projeto, o algoritmo principal é o que analisa os dados de ultrassom coletados e retorna a velocidade do líquido por meio do algoritmo de autocorrelação (KASAI et al., 1985) ou deslocamento de fase. Dentro deste contexto, este trabalho propôs um estudo e implementação em *Very High Speed Integrated Circuit* (VHSIC) *Hardware Description Language* (VHDL) do método de autocorrelação para a caracterização da velocidade de escoamento de fluidos.

1.1 Justificativa

Determinar as distribuições de velocidade ao longo da seção transversal de um canal fluvial é de vital importância para detalhar as possibilidades do seu potencial energético (PEKTAŞ, 2015). A Agência Nacional de Águas (ANA, 2017) fornece uma série de dados hidrológicos, tais como vazão, nível de água, concentração de

sedimentos, entre outros. Todavia, inexiste nos dados fornecidos qualquer informação acerca da velocidade do rio. As razões variam desde os altos custos envolvidos nas medições até a dificuldade de acesso à área de estudo (CRUZ et al., 2019). Uma alternativa para contornar esses empecilhos é a utilização de embarcações ou VANTs controlados por tecnologias de rádio frequência, *Global System for Mobile communication* (GSM) ou *Global Positioning System* (GPS).

A Universidade Tecnológica Federal do Paraná (UTFPR) – Câmpus Toledo tem desenvolvido um VANT no qual já é embarcado tanto um sistema de medição de profundidade de rios (JÚNIOR, 2017) quanto um sistema de detecção de obstáculos (GONÇALVES, 2018). O sistema, quando completo, estará apto a realizar um levantamento do perfil de profundidade de um rio de forma remota e movimentar-se de forma autônoma na superfície de seu leito, dado que, na água, o drone não precisa despender energia para se manter flutuando.

Visto isso, o método de autocorrelação para a medição de velocidade de escoamento de fluidos que foi implementado em VHDL neste trabalho, no qual o cômputo das expressões e as atribuições de valores ocorrem em paralelo, resultando em um código declarativo (HEXSEL, 2021), facilitará a realização de medições periódicas em diversos trechos do rio ou canal fluvial, dado que o medidor do tipo UVP dispõe de flexibilidade e expansibilidade para medir a velocidade de mais de um sensor de ultrassom ao mesmo tempo.

Na década de 2020 ADCPs utilizam de três a quatro transdutores de ultrassom, que permitem, entre outras coisas, uma visualização em três dimensões (3D) do vetor da velocidade da água. E, para isso ocorrer, é necessário ter os dados computados ao mesmo tempo – uma das características da linguagem VHDL. Dessa forma, a solução para com a insuficiência de dados hidrográficos e o custo para obtê-las estará tangível.

1.2 Objetivos

Este trabalho teve como objetivo principal implementar, em VHDL, o método de autocorrelação para a medição de velocidade de escoamento de fluidos, assegurando as seguintes especificidades:

• Implementar em VHDL o algoritmo de autocorrelação previamente validado no software MATLAB®;

• Testar e validar o algoritmo em VHDL com dados experimentais previamente coletados.

2 REFERENCIAL TEÓRICO

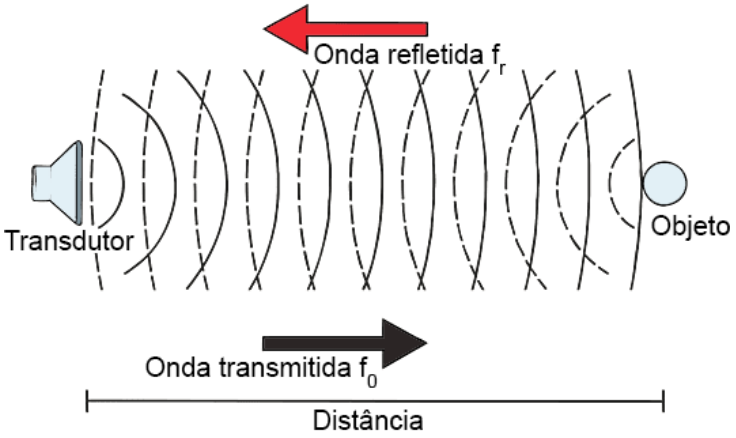
Neste capítulo serão discutidos os conceitos imprescindíveis para a realização deste trabalho. Em um primeiro momento será apresentada uma breve discussão acerca das ondas ultrassônicas. Na sequência, uma introdução ao efeito Doppler. Por fim, será abordado o método da autocorrelação, empregado na obtenção da velocidade de escoamento de fluidos.

2.1 Ondas ultrassônicas

Som e ultrassom são termos usados para descrever a propagação de uma perturbação mecânica em diferentes frequências. O ultrassom corresponde a uma onda mecânica que se propaga em frequências acima do alcance da audição humana (acima de 20 kHz). Ademais, ela se propaga tanto em fluidos (gases e líquidos) como sólidos. Inobstante, um transdutor é qualquer dispositivo que converte uma forma de energia em outra. Tratando-se do ultrassom, o transdutor converte energia elétrica em energia mecânica e vice-versa (LAUGIER e HAIAT, 2010).

Neste trabalho, o transdutor teve duas funções: (1) converter a energia elétrica fornecida pelo transmissor em pulsos acústicos direcionados ao rio ou canal fluvial; (2) servir como receptor de ecos, convertendo mudanças de pressão fracas em sinais elétricos para processamento, como pode ser visualizado na Figura 1.

Figura 1 - Ondas ultrassônicas



Fonte: Adaptado de Twinkl, s.d.

2.2 Efeito Doppler

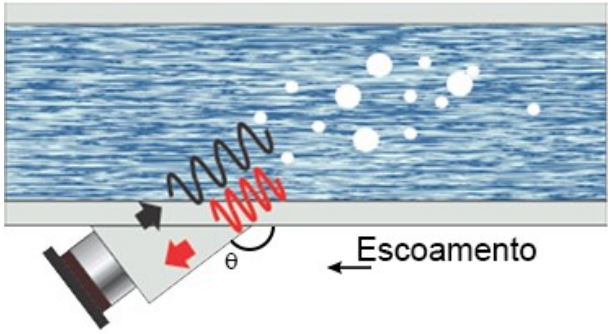
O efeito Doppler é caracterizado pela alteração na frequência de uma onda quando há movimento relativo existente entre fonte e observador (ELMORE e HEALD, 1985). De acordo com Merritt (2007), um objeto estacionário refletirá o sinal original na mesma frequência em que foi enviado, de modo que $f_0 = f_r$ (Figura 1), onde f_0 é a frequência do sinal original (proveniente do transdutor) e f_r é a frequência do sinal refletido. Por outro lado, um objeto em movimento – em relação a fonte – refletirá um sinal com uma frequência diferente da original.

Se o objeto refletor estiver se movendo em direção ao transdutor, f_r aumentará e, se o objeto estiver se afastando do transdutor, f_r diminuirá. Essa mudança na frequência é diretamente proporcional à velocidade do observador em relação ao transdutor, sendo resultado do efeito Doppler. A relação de f_r para com a velocidade do observador pode ser descrita como:

$$f_{dop} = f_r - f_0 = 2f_0 \frac{v}{c}, \qquad \textbf{(1)}$$

onde f_{dop} é a frequência de desvio Doppler, v a velocidade do observador em relação ao transdutor e c a velocidade do som no meio (MERRITT, 2007). Um exemplo de aplicação do efeito Doppler semelhante à deste trabalho é mostrado na Figura 2, na qual as bolhas de ar presentes no escoamento de água atuam como refletores do ultrassom.

Figura 2 - Seção transversal de uma tubulação exibindo uma aplicação da técnica ultrassônica utilizando o efeito Doppler



Fonte: Adaptado de Coutinho, 2014

Entretanto, geralmente o feixe de ultrassom se aproxima do observador em movimento em um determinado ângulo (Figura 2). Neste caso, f_{dop} é reduzido proporcionalmente ao cosseno deste ângulo:

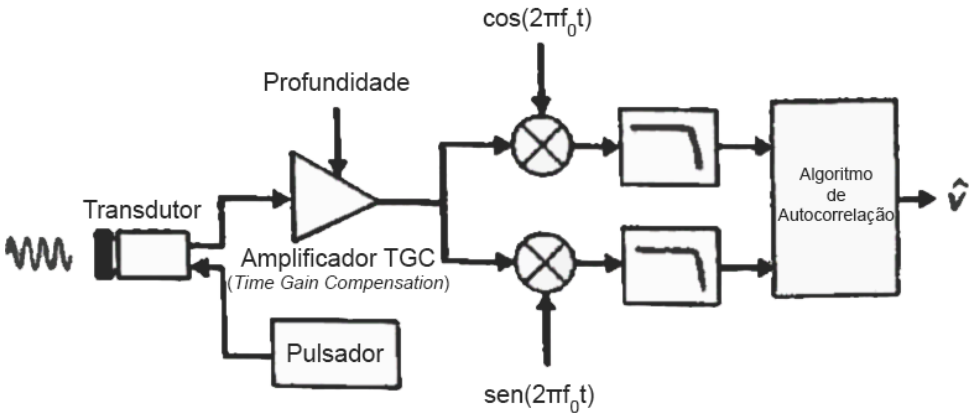
$$f_{dop} = f_r - f_0 = 2f_0 \cos\theta \frac{v}{c}, \quad \textbf{(2)}$$

onde θ é o ângulo entre o feixe de ultrassom e a direção do observador em movimento.

2.3 Método de autocorrelação

O método de autocorrelação é baseado na estimativa de fase para pulsos sequenciais de um sinal complexo demodulado. A visão geral do estimador de velocidade deste método é exibida na Figura 3. Nela, o transdutor emite e recebe sinais de ultrassom e, como os ecos são recebidos com diferentes tensões devido à profundidade, os sinais são amplificados. Posteriormente, os ecos recebidos são submetidos a demodulação complexa e, dessa forma, passam a ter componentes em fase e quadratura. Por fim, ao passarem pelo algoritmo de autocorrelação, o qual foi o foco deste trabalho, a velocidade do escoamento pode ser obtida.

Figura 3 - Visão geral do estimador de velocidade do método de autocorrelação

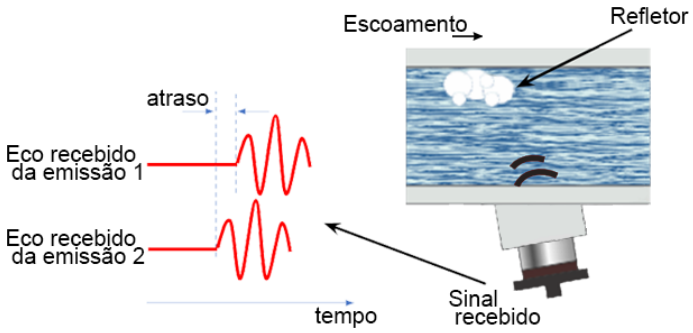


Fonte: Adaptado de Jensen, 1999

Dada a visão geral, as Figuras 4, 5 e 6 analisam mais detalhadamente o método de autocorrelação. Na Figura 4, um transdutor emite pulsos de ultrassom de curta duração que, ao serem refletidos por um objeto em movimento, geram ecos com a sua frequência alterada pelo efeito Doppler e com um atraso de tempo. É importante destacar que, apesar da Figura 4 apresentar bolhas como refletores, o objetivo deste

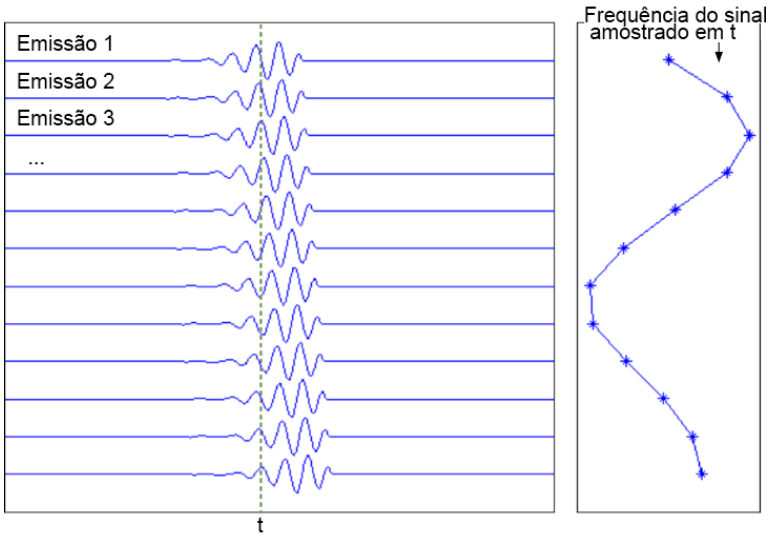
trabalho, futuramente, é a utilização de sedimentos dos rios ou canais fluviais como refletores. Amostrando-se em um tempo fixo t cada uma das n emissões, obtém-se um sinal que irá possuir frequência equivalente ao desvio Doppler (Figura 5). Como esse sinal não é contínuo dado que é analisado uma quantidade pequena de emissões, o sinal amostrado apresentará um espectro com uma banda larga (Figura 6). Essa banda dificulta a estimação da frequência de desvio e por isso é necessário a utilização de algoritmos mais complexos como o de autocorrelação para obter a frequência central do sinal.

Figura 4 - Sinais senoidais de curta duração com um desvio Doppler na frequência e atraso no tempo (em relação à emissão anterior)



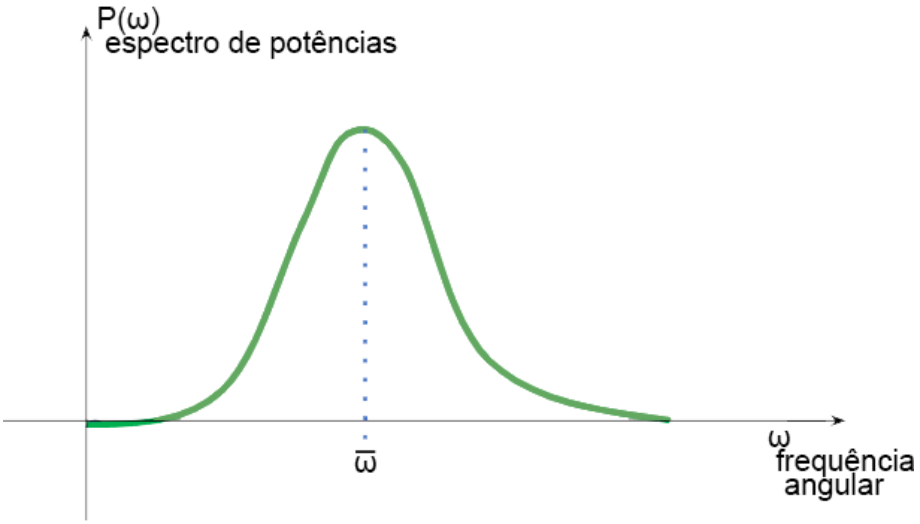
Fonte: Adaptado de Coutinho, 2014

Figura 5 - Sinal obtido amostrando-se cada emissão em t



Fonte: Adaptado de Coutinho, 2014

Figura 6 - Espectro de frequência do sinal amostrado na Figura 5



Fonte: Adaptado de Coutinho, 2014

Um sinal complexo em fase e quadratura $r = x(n) + iy(n)$, onde n é a n-ésima emissão, pode ser ter sua fase expressa como $\tan^{-1}\frac{y}{x}$, de forma que a diferença de fase temporal é aproximada para forma discreta como:

$$\tan^{-1}\frac{y(n)x(n-1) - y(n-1)x(n)}{x(n)x(n-1) + y(n)y(n-1)}, \quad \textbf{(3)}$$

onde o numerador representa a parte imaginária e o denominador a parte real da função de autocorrelação $R(\,)$ para $N-1$ pares de linhas ou emissões descritas como:

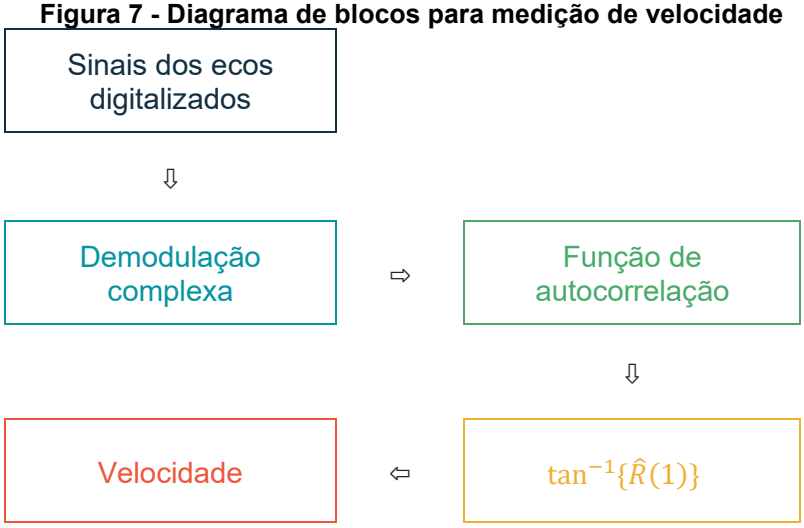
$$R(1) = \frac{1}{N-1}\sum_{n=0}^{N-2} r^*(n)r(n+1)\,, \quad \textbf{(4)}$$

onde r^* é o complexo conjugado. Segundo Kasai et al. (1985), a fase média, ou a frequência angular no sinal, pode ser então estimada com a $\tan^{-1}\{\hat{R}(1)\}$, onde $\hat{R}(1)$ é a função de autocorrelação média.

3 MATERIAIS E MÉTODOS

O sistema de medição de velocidade por efeito Doppler proposto neste trabalho se baseou no método de autocorrelação, apresentado na subseção 2.3. Dessa forma, ele pode ser representado por um diagrama de blocos, conforme a Figura 7.

Figura 7 - Diagrama de blocos para medição de velocidade



Fonte: Autoria própria, 2022

Vale ressaltar que a medição de velocidade foi efetuada com dados reais, adquiridos conforme será apresentado na subseção 3.1. Tais dados são sinais de ecos que foram, anterior a este trabalho, tratados digitalmente. Isto é, sinais que foram submetidos a um processo de demodulação complexa ou demodulação em quadratura para a redução de largura de banda. Então demodulados, neste trabalho, os sinais foram processados em VHDL para obter a função de autocorrelação. Dessa forma, a partir da função de autocorrelação obtida, foi possível adquirir a frequência do desvio Doppler e, em seguida, estimar a velocidade do fluido analisado. Por fim, foi possível discorrer sobre a acurácia do sistema desenvolvido ao compará-lo com os valores esperados.

3.1 Aquisição dos dados reais

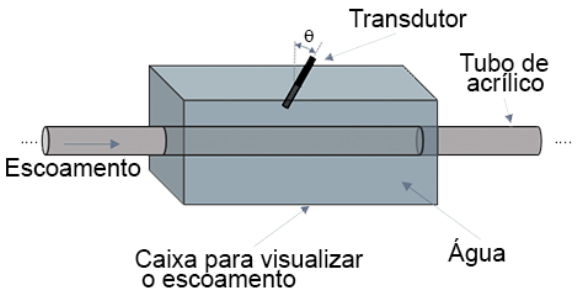
Os dados reais, utilizados para a medição de velocidade, foram obtidos através de um experimento empregando um instrumento ultrassônico do tipo UVP, como expressa a Figura 8.

Figura 8 - Diagrama industrial do experimento

Fonte: Autoria própria, 2022

No experimento, a bomba centrífuga – acionada através de um inversor de frequência variável – circula o líquido em um tubo de acrílico cujo diâmetro interno é de 25,9 mm. Ademais, como refletor, um pó marcador com densidade de 1,07 g/cm^3 foi adicionado ao reservatório, a uma concentração de 4 g/L. Como a densidade do pó é próxima da densidade da água (1 g/cm^3), a velocidade com que o pó flui é próxima a velocidade da água. Outrossim, a vazão de água é mensurada por um medidor de vazão do tipo Coriolis. Por fim, um transdutor de ultrassom de 4 MHz e 5 mm de diâmetro da Met-Flow foi posicionado em A, o qual foi imerso em água e angulado em 18º, conforme a Figura 9. É importante destacar que, apesar deste experimento utilizar um pó marcador como refletor, o objetivo futuro deste trabalho é a utilização de sedimentos dos rios ou canais fluviais como refletores.

Figura 9 - Posicionamento do transdutor de ultrassom



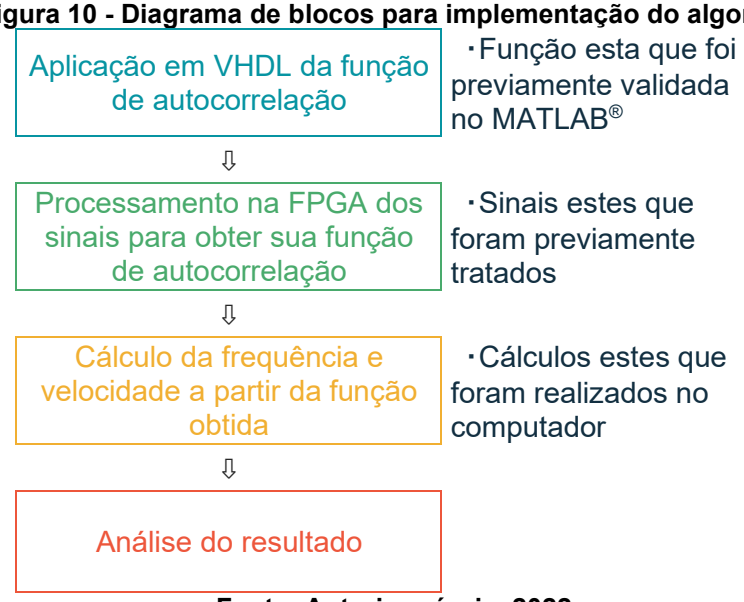
Fonte: Autoria própria, 2022

A medição de velocidade foi realizada com o UVP-DUO e com um PXI System da National Instruments, modelo NI5752R. A transmissão dos sinais de ecos de radiofrequência do UVP foi utilizada como entrada para a placa PXI de aquisição. Ainda, foi combinada a frequência de repetição de pulso em ambos os instrumentos e foram usadas 128 emissões cuja frequência de repetição de pulsos foi de 6 kHz. A velocidade de cada escoamento experimentado foi estimada usando 9,81 s de dados adquiridos, e os ecos recebidos foram amostrados a 50 MHz no PXI. Por fim, a velocidade média de escoamento de 1,0 m/s foi definida como base para testes, nos quais os dados adquiridos pelo PXI foram, anterior a este trabalho, processados no MATLAB® e, por conseguinte, validados.

3.2 Implementação em VHDL do método de autocorrelação

Em relação a implementação em VHDL do método de autocorrelação, seguiu-se uma série de etapas descritas na Figura 10.

Figura 10 - Diagrama de blocos para implementação do algoritmo



Fonte: Autoria própria, 2022

A primeira etapa foi realizada com base em um algoritmo no MATLAB® previamente validado (Figura 11), e será aprofundada na seção 4. Na segunda etapa, os diferentes sinais digitalmente tratados foram enviados para uma *Field-Programable Gate Array* (FPGA). A FPGA, rodando o algoritmo aferido portado para VHDL, processou a autocorrelação e enviou os dados de volta ao computador para, então na

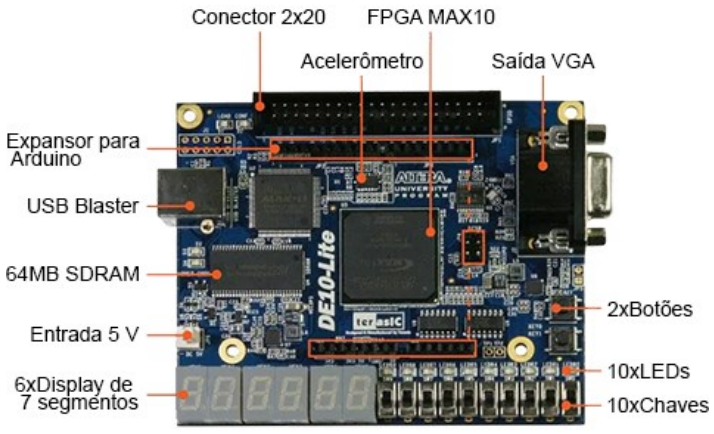
terceira etapa, obter a velocidade e, por fim, analisá-la. Para tal feito, foi utilizado um *kit* DE10-Lite, da Terasic (Figura 12).

Figura 11 - Algoritmo validado no MATLAB®

```
>> clear all
        % reference frequency
                fc = 219;
        % sampling frequency
                fs = fc*8;
        % time array t=0 to 1 sec
                t = linspace(0,1,1*fs);
        % sinusoidal signal with frequency fc
                y = sin(2*pi*fc*t);

                plot(t,y)

%% complex demodulation
        % applies Hilbert transform
                iq = hilbert(y);
                t = (0:1/fs:size(iq,1)*(1/fs)-(1/fs))';
                datad = iq.*exp(-1i*2*pi*fc*t)/sqrt(2);

%% apply autocorrelation lag 1
                R = datad(2:end)*datad(1:end-1)';
                w = atan2(imag(R),real(R));
        % shows the estimated frequency
                fc_est=fs*w/(2*pi)
```

Fonte: Autoria própria, 2022

Figura 12 - *kit* DE10-Lite



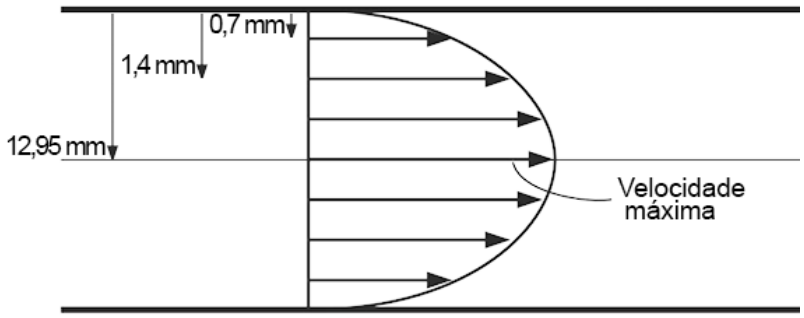
Fonte: Adaptado de Terasic, s.d.

Para a comunicação direta do *kit* com o computador, foi utilizado, dentro da FPGA, o Altera Nios II, o qual é um processador *softcore* RISC de 32 bits que possibilita a fácil conexão com outros periféricos. O Nios II, neste trabalho, é implementado na linguagem de programação C e é responsável somente por encaminhar os pares complexos e exibir o resultado da autocorrelação em sua tela; isto é, tanto a parte das operações aritméticas quando a parte de leitura e escrita na memória são processadas em VHDL.

3.3 Testes

Os testes para a validação do sistema proposto foram produto da autocorrelação realizada com três vetores de dados digitalizados obtidos no experimento da subseção 3.1, no qual a velocidade horizontal média do escoamento real é de 1 m/s. A localização dos diferentes vetores dentro do tubo de acrílico cujo diâmetro interno é de 25,9 mm é exibida na Figura 13. Ademais, para a análise de erros, foram relacionados os resultados da autocorrelação média e as velocidades estimadas no MATLAB® – validadas pelo PXI – e na FPGA.

Figura 13 - Localização das amostras utilizadas para os testes

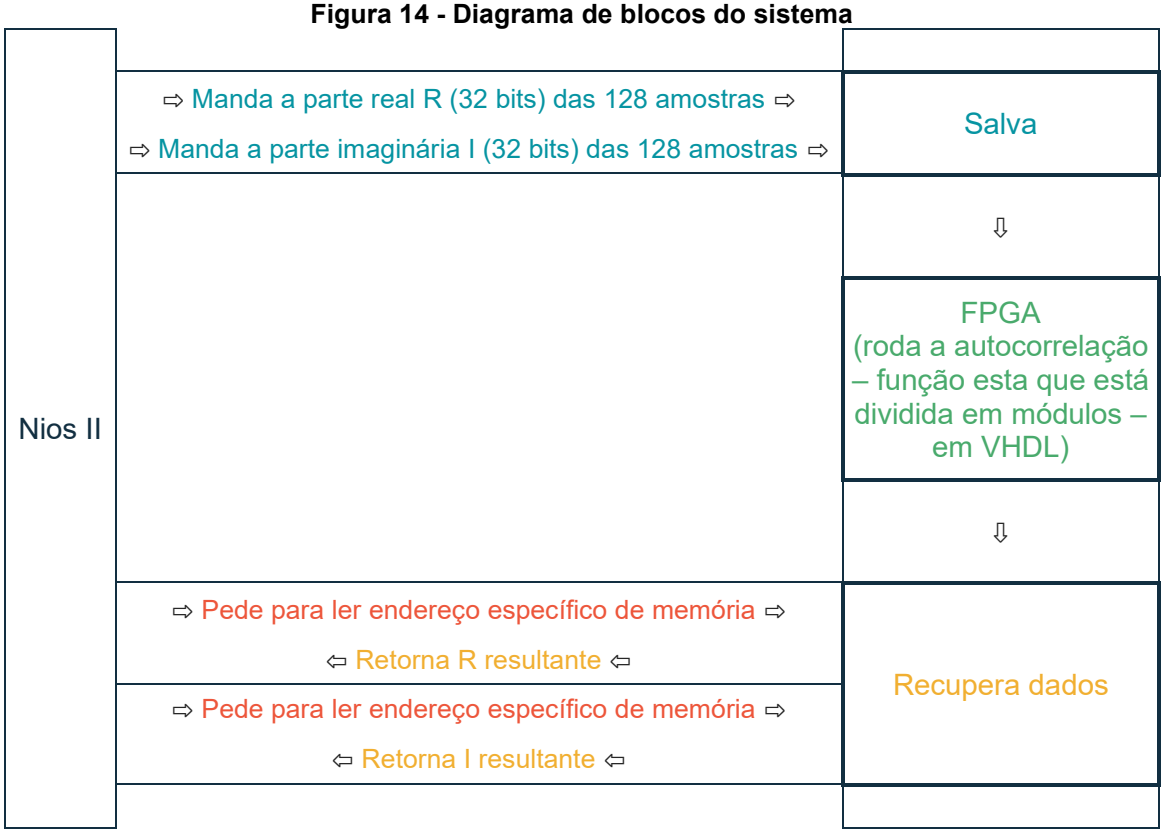


Fonte: Adaptado de Tecconcursos, s.d.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Neste capítulo serão discutidos os comportamentos dos códigos implementados na FPGA (SCHMIDT, 2022). Em um primeiro momento será apresentado o algoritmo de autocorrelação executado em VHDL. Na sequência, o algoritmo para a interface paralela de comunicação e, por fim, a acurácia dos resultados. O sistema, como um todo, é apresentado na Figura 14.

Figura 14 - Diagrama de blocos do sistema



Fonte: Autoria própria, 2022

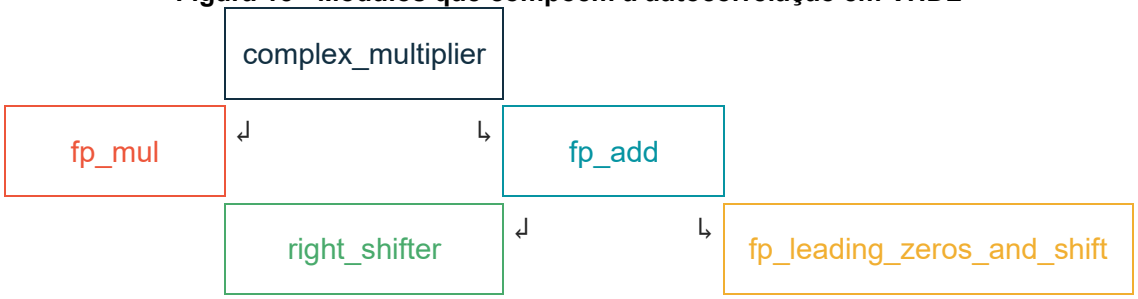
O Nios II é responsável por mandar o vetor de 128 pares complexos, os quais são inseridos manualmente no algoritmo e posteriormente salvos na memória da FPGA. Com os dados recebidos, a FPGA executa o algoritmo de autocorrelação em VHDL e computa o resultado. Por fim, o Nios II recupera os endereços de memória no qual o par complexo resultante está contido e o exibe em sua tela.

4.1 Autocorrelação implementada em VHDL

A execução da autocorrelação em VHDL considerou o padrão IEEE 754, o qual é uma forma de representação de números reais em *hardware*, e baseou-se em Deschamps et al. (2012). Ela foi dividida em cinco módulos: (1) complex_multiplier;

(2) fp_mul; (3) fp_add; (4) right_shifter; (5) fp_leading_zeros_and_shift, conforme a Figura 15.

Figura 15 - Módulos que compõem a autocorrelação em VHDL



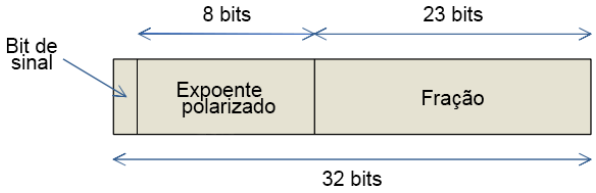
Fonte: Autoria própria, 2022

O módulo complex_multiplier é responsável por realizar a multiplicação do par complexo por intermédio dos demais módulos: fp_mul, que multiplica dois pontos flutuantes de 32 bits; fp_add, que soma dois pontos flutuantes de 32 bits; right_shifter, que é encarregado de igualar os expoentes dos pontos presentes no fp_add; e fp_leading_zeros_and_shift, que, ainda dentro do fp_add, caso haja zeros à esquerda no resultado decorrente de uma subtração, desloca-o.

4.1.1 Padrão IEEE 754

Em um número de ponto flutuante, o ponto é deslizado dinamicamente para uma posição conveniente. Tal feito possibilita a expressão em poucos dígitos de números muito grandes e muito pequenos. O padrão IEEE 754 fornece dois formatos para representar números de ponto flutuante: precisão simples e precisão dupla. O formato simples, utilizado neste trabalho, consiste em 32 bits, sendo 1 bit para o sinal, 8 bits para o expoente e 23 bits destinados à fração, como é mostrado na Figura 16.

Figura 16 - Formato precisão simples no padrão IEEE 754



Fonte: Adaptado de Figueiró, 2011

Tal número pode ser representado, de forma analítica, pela seguinte equação:

$$(-1)^s 2^e 1.frac\,, \quad \textbf{(5)}$$

onde $frac$ é a fração, s é o bit de sinal (0 para positivo, 1 para negativo), e é o expoente polarizado (e = expoente não-polarizado + 127 (*bias*)) e $1.frac$ é chamado de mantissa, sendo composto pela fração $frac$ e um bit implícito. O expoente polarizado é uma maneira de representar números não positivos adicionando-se um número de desvio (PATTERSON et al., 2003). Neste caso, o desvio ou *bias* é 127, ou seja, a variação do expoente está entre 1 e 254. Além disso, como a base 2 é a mesma para todos os números dentro deste padrão, ela não é armazenada.

A fim de evitar representações múltiplas para o mesmo número, neste formato é requerido que ele seja normalizado, isto é, o ponto é deslocado até que o ponto esteja à direita do dígito diferente de zero mais à esquerda. Como exemplo de normalização, será utilizado o número 1101.01 * (2^0):

1101.01 * (2^0)
= 110.101 * (2^1)
= 11.0101 * (2^2)
= 1.10101 * (2^3),

que, depois de normalizado, tem seu primeiro bit implícito no armazenamento em 32 bits. Como exemplo de conversão para os 32 bits do padrão IEEE 754, será utilizado o número 12,3125:

Primeiro, converte-se para a base 2:

1100.0101,

depois, converte-se para a forma $2^e 1.frac$:

(2^3)1.1000101,

em seguida, adiciona-se 127 (*bias*) ao expoente e o converte para binário:

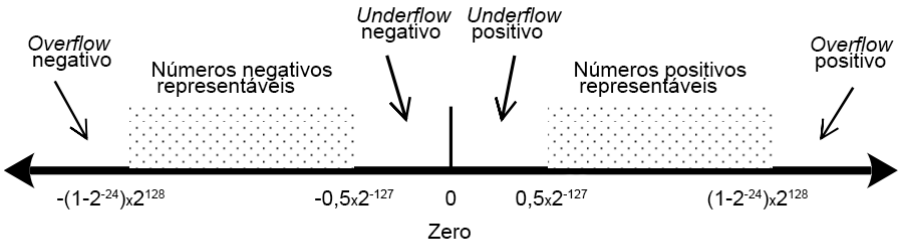
3 + 127 = 130 = 10000010,

e, por último, determina-se o bit de sinal e junta-se os números (sem o bit implícito):

0 10000010 10001010000000000000000.

Adicionalmente, a Figura 17 contém o intervalo de números em ponto flutuante representáveis em uma palavra de 32 bits, na qual notam-se regiões de *overflow* e *underflow*. Um *overflow* ocorre quando o expoente do resultado é superior ao expoente máximo. Por sua vez, um *underflow* ocorre quando o expoente do resultado é muito pequeno.

Figura 17 - Números de ponto flutuante representáveis em formato 32 bits



Fonte: Adaptado de Figueiró, 2011

Destaca-se que, em números inteiros, isto equivale ao intervalo de -2^{31} a (2^{31} - 1). Em outras palavras, a quantidade de números representáveis é na ordem de 2^{32} números distintos.

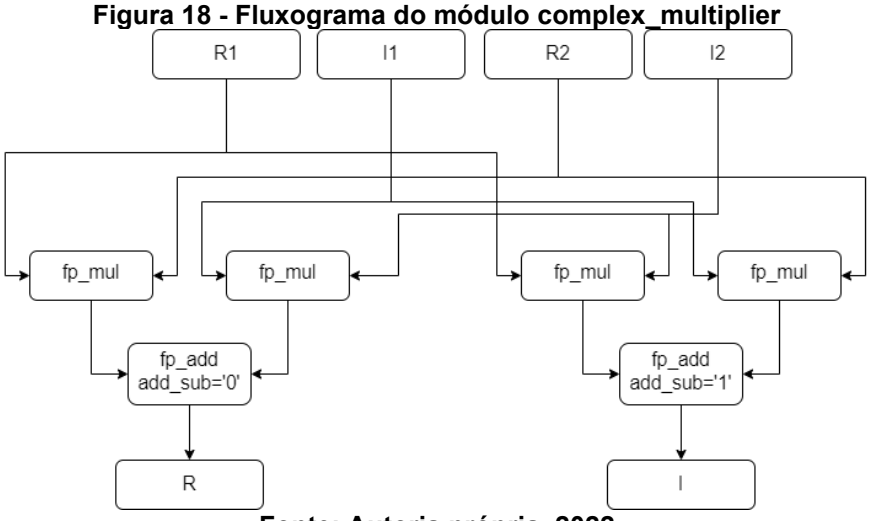
4.1.2 Módulo complex_multiplier

Neste módulo é feita a multiplicação de dois pares complexos de pontos flutuantes armazenados em 32 bits. É fato que, ao multiplicar dois pares complexos, obtém-se:

$$(R_1 + I_1 i)(R_2 + I_2 i) = R_1 R_2 - I_1 I_2 + (R_1 I_2 + I_1 R_2) i \,, \qquad \textbf{(6)}$$

e, como VHDL não suporta a multiplicação direta de pares complexos, utiliza-se, adicionalmente, de dois submódulos: (1) fp_mul, para a multiplicação; (2) fp_add, para adição, de acordo com o fluxograma da Figura 18.

Figura 18 - Fluxograma do módulo complex_multiplier



Fonte: Autoria própria, 2022

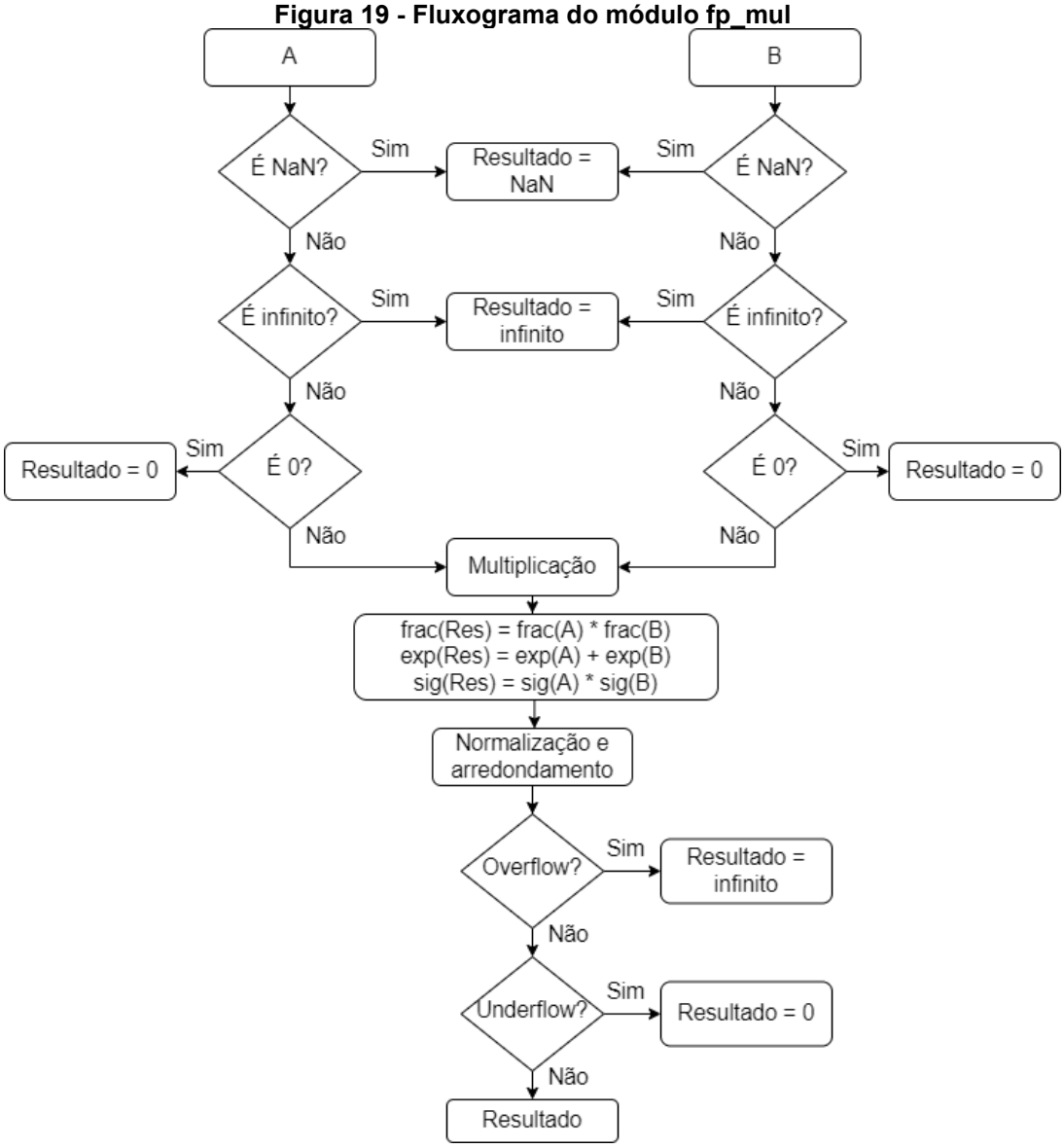
Dessa forma, para a aquisição da parte real R, R1 e R2 são multiplicados e seu resultado é armazenado em R1R2. O mesmo acontece para I1 e I2, no qual o resultado é armazenado em I1I2. Visto que I1I2 será subtraído de R1R2 – decorrência da multiplicação de números complexos – add_sub é definido como 0; isto significa que I1I2 receberá o oposto do seu sinal original. Por sua vez, a aquisição da parte imaginária I envolve a multiplicação de R1 e I2 – armazenada em R1I2 – e a multiplicação de I1 e R2 – armazenada em I1R2. Dado que os resultados são somados, add_sub é definido como 1. É importante destacar que, apesar das operações aritméticas, os resultados decorrentes deste módulo mantêm-se no formato de 32 bits.

4.1.3 Módulo fp_mul

Neste módulo (Figura 19) é feita a multiplicação de dois pontos flutuantes armazenados em 32 bits. Para tal, primeiro é verificado se algum deles é *Not a Number* (NaN), infinito ou zero, seguido do tratamento adequado. Um NaN ocorre quando um número tem seu expoente composto somente por uns (11111111) e sua fração é diferente de zero, podendo apresentar tanto sinal positivo como negativo. Ele é interpretado de duas formas: silenciosa, na qual sua propagação em operações aritméticas não é detectada como exceção; sinalizador, reconhecido facilmente visto que gera uma operação inválida. Um número infinito, por sua vez, ocorre quando um número tem seu expoente composto somente por uns e sua fração, diferente de um NaN, é igual a zero, podendo apresentar tanto sinal positivo como negativo. Por fim, um expoente composto somente por zeros junto com sua fração igual a zero representa um zero, também aceitando tanto bit de sinal positivo quanto negativo.

Verificadas as exceções, a multiplicação é calculada – as mantissas são multiplicadas, os expoentes somados e os sinais multiplicados. O resultado da multiplicação da mantissa terá o dobro do comprimento dos operandos, estendendo sua precisão de forma a evitar erros de arredondamento. Consecutivamente, o ponto flutuante resultante é normalizado e arredondado. Por fim, *overflow* – superação da capacidade suportada de bits – e *underflow* – decorrência de um expoente muito pequeno, ocasionando no arredondamento para zero – são verificados e, se não estiverem presentes, o resultado computado anteriormente é salvo.

Figura 19 - Fluxograma do módulo fp_mul



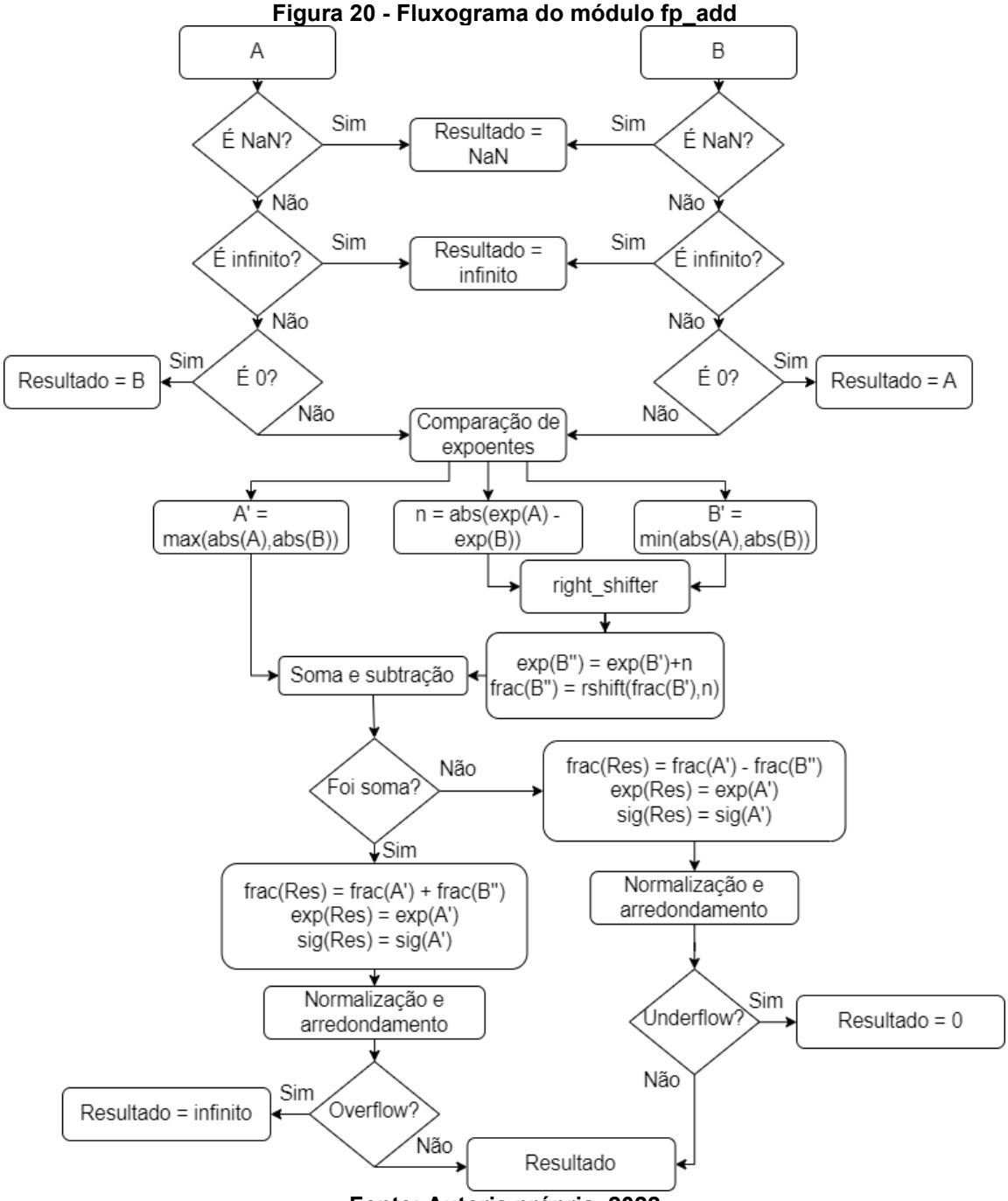
Fonte: Autoria própria, 2022

4.1.4 Módulo fp_add

Neste módulo (Figura 20) é feita a adição e subtração de dois pontos flutuantes armazenados em 32 bits. Para tal, primeiro é verificado se algum deles é NaN, infinito ou zero, seguido do tratamento adequado. Posteriormente, os expoentes são comparados para, caso haja diferença entre eles, seu alinhamento possa ser realizado. Se não houver diferença, a soma e subtração são calculadas. Senão, A' recebe o módulo do maior valor entre os pontos, B' recebe o módulo do menor valor entre os pontos e n recebe o módulo da diferença entre os expoentes; isso é necessário para que a mantissa de menor valor seja deslocada para apresentar o mesmo expoente da mantissa de maior valor. Assim, torna-se possível a soma e

subtração das mantissas que apresentavam expoentes diferentes. Por fim, B'' recebe o valor de B' deslocado – tanto da mantissa quanto do expoente.

Figura 20 - Fluxograma do módulo fp_add



Fonte: Autoria própria, 2022

Após a aquisição de B'', soma e subtração são calculadas; porém, o resultado utilizado depende do sinal de A' e B''. Se os sinais forem positivos, mantissas são somadas e, como B'' foi deslocado, o expoente final é o mesmo expoente de A' e o sinal do resultado é o mesmo sinal de A'; o mesmo acontece caso haja subtração – as mantissas são subtraídas, o expoente final é o mesmo expoente de A' e o sinal do

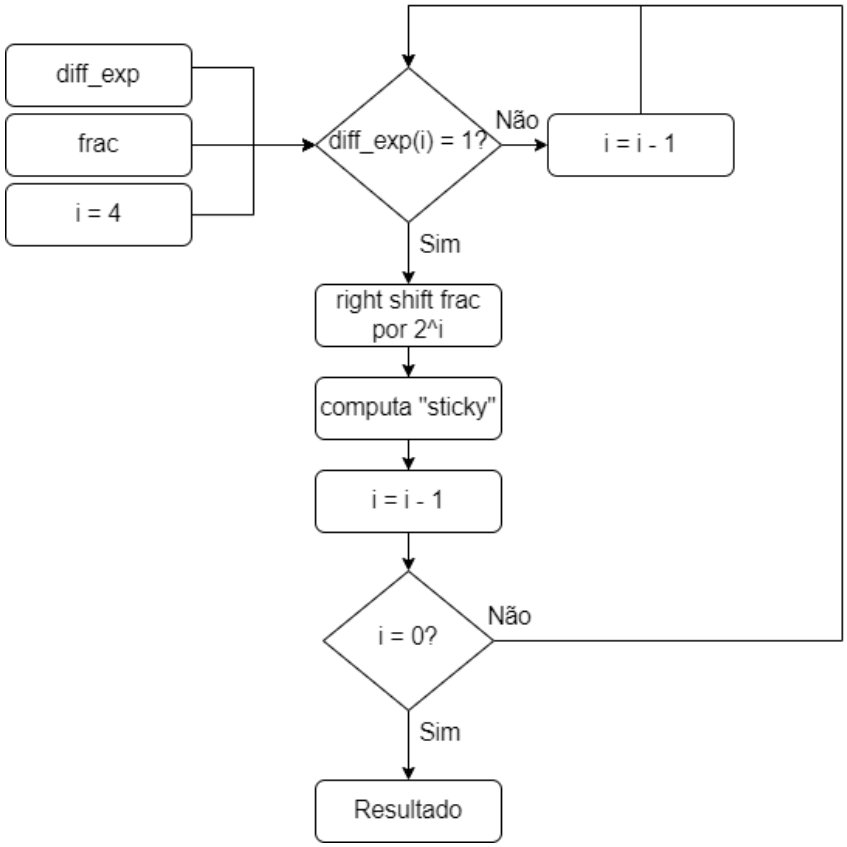
resultado é o mesmo sinal de A'. Consecutivamente, o ponto flutuante resultante é normalizado e arredondado. Por fim, *overflow* e *underflow* são verificados conforme a operação e, se não estiverem presentes, o resultado computado anteriormente é salvo.

4.1.5 Módulo right_shifter

Neste módulo (Figura 21) a mantissa é deslocada diff_exp vezes para a direita, ou seja, ela é dividida diff_exp vezes por 2. Ele é utilizado no módulo fp_add somente quando o alinhamento dos expoentes é necessário. Para cada '1' na i-ésima posição de diff_exp, a mantissa é deslocada. A cada deslocamento, o *sticky* bit é verificado – ou seja, se o bit que será descartado for '1', o *sticky* receberá '1', que, depois, será utilizado para arredondar o ponto flutuante para cima.

Destaca-se que, neste módulo, i inicia em 4. Isso se dá por i representar PLOG, isto é, $\log_2(P)$, no qual P, no formato 32 bits, é 27.

Figura 21 - Fluxograma do módulo right_shifter

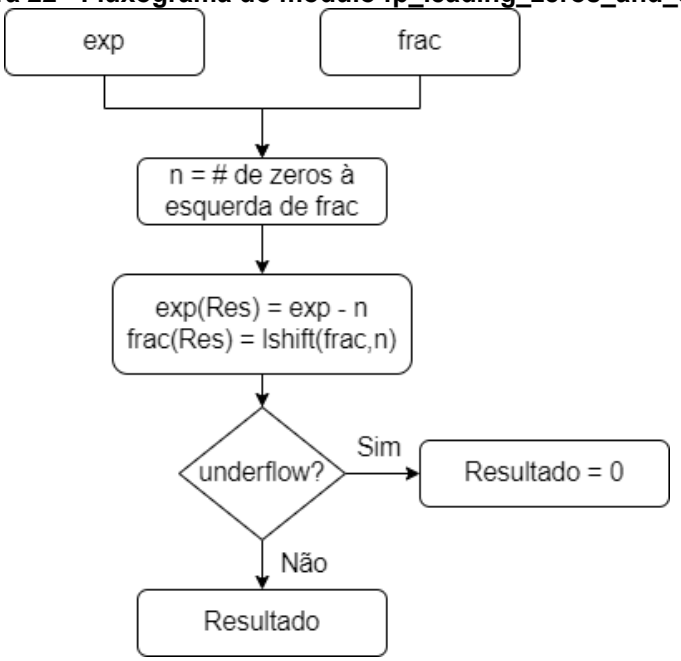


Fonte: Autoria própria, 2022

4.1.6 Módulo fp_leading_zeros_and_shift

Neste módulo (Figura 22) a mantissa e seu respectivo expoente são normalizados. Ele é usado no módulo fp_add caso a operação aritmética realizada tenha sido a subtração. Para tal, os zeros iniciais da mantissa são contados e, em seguida, deslocados para a esquerda. Posteriormente, o expoente é ajustado (subtraindo a quantidade de zeros à esquerda inicialmente contados). Por fim, *underflow* é verificado e, se não estiver presente, o resultado computado anteriormente é salvo.

Figura 22 - Fluxograma do módulo fp_leading_zeros_and_shift

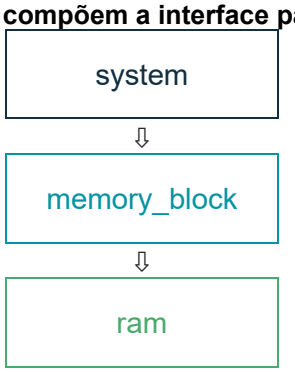


Fonte: Autoria própria, 2022

4.2 Interface paralela de comunicação

A comunicação do computador com o algoritmo de autocorrelação em VHDL é realizada através do Altera Nios II, no qual os pares complexos são inseridos manualmente. O Nios II é responsável por encaminhar os vetores (separadamente) da parte real e imaginária da demodulação complexa e, após a autocorrelação ser finalizada, mostrar o resultado em sua tela. É importante destacar que o algoritmo de comunicação foi escrito para computar 128 amostras de pares complexos, e, para tal, foi dividido em três módulos: (1) system; (2) memory_block; (3) ram, conforme a Figura 23.

Figura 23 - Módulos que compõem a interface paralela de comunicação



Fonte: Autoria própria, 2022

O módulo system recebe os valores do Nios II e encaminha para o submódulo memory_block, que irá salvá-los na *Random Access Memory* (RAM). Após salvos, o system inicia a leitura dos dados contidos na RAM para, em pares, serem multiplicados através do complex_multiplier. É importante destacar que, devido a autocorrelação ser a multiplicação de um complexo com um complexo conjugado, I2 recebe o oposto de seu sinal original. A cada multiplicação, o resultado é somado ao resultado anterior. Após a multiplicação dos pares das 128 amostras ser concluída, o Nios II irá requisitar a leitura dos endereços nos quais os resultados estão contidos e os exibirá em sua tela. A autocorrelação é controlada por uma máquina de estados, como mostra a Figura 24.

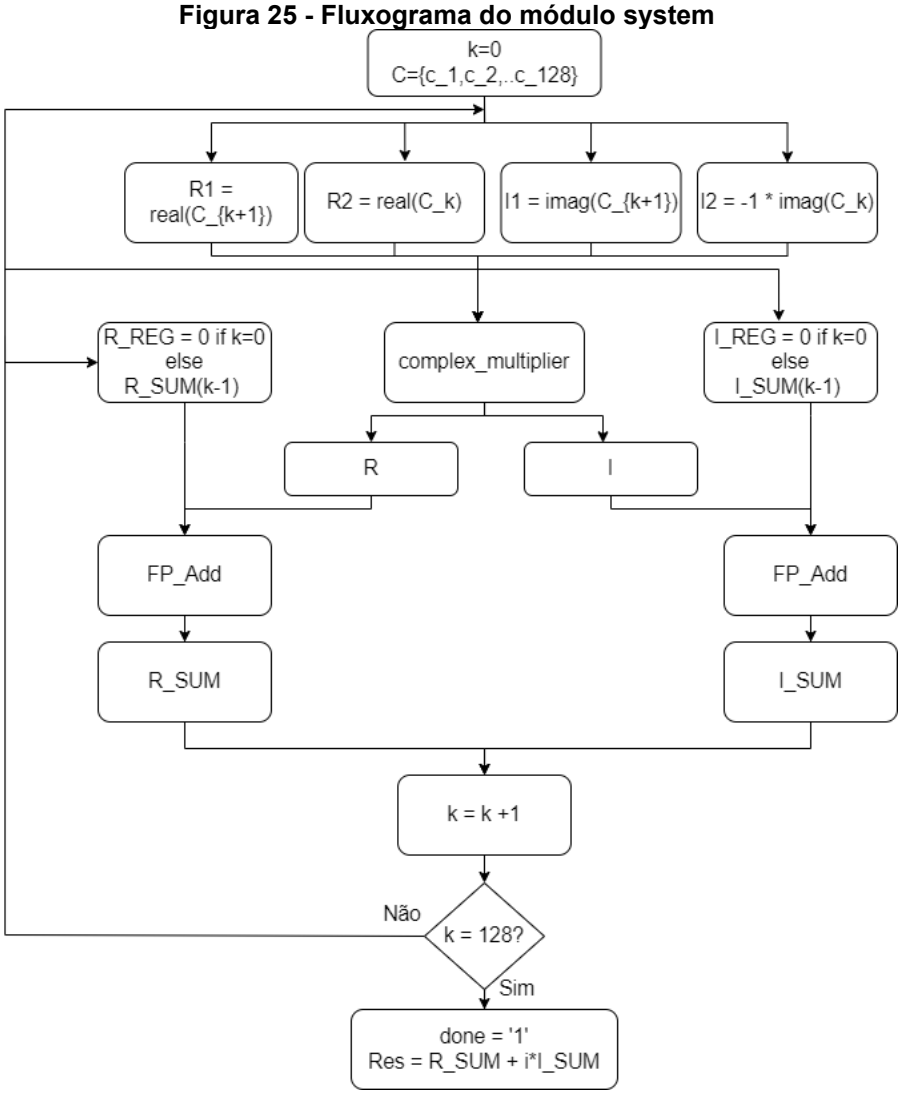
Figura 24 - Máquina de estados da autocorrelação

```
1  output_memory_we <= '0'; done <= '0';
2  case state is
3          when s0 =>
4                  R_REG <= (others=>'0'); I_REG <= (others=>'0');
5                  addr_b_1 <= (others=>'0'); addr_b_2 <= "0000001";
6                  if (start='1') then state <= s1; end if;
7          when s1 => state <= s2;
8          when s2 => state <= s3;
9          when s3 =>
10                 output_memory_we <= '1';
11                 R_REG <= R_SUM; I_REG <= I_SUM;
12         when s4 =>
13                 if (addr_b_2 >= 127) then state <= s5;
14                 else addr_b_2 <= addr_b_2 + 1; addr_b_1 <= addr_b_1 + 1;
15                 state <= s1; end if;
16         when s5 =>
17                 done <= '1';
18                 if (start='0') then state <= s0; else state <= s5; end if;
19         when others => state <= s0;
20 end case;
```

Fonte: Autoria própria, 2022

Primeiro, no estado inicial s0, os registradores nos quais as multiplicações serão salvas são zerados e os endereços de memória nos quais os pares estão contidos recebem ‘0’ e ‘1’ para iniciar a multiplicação com *lag* 1 (R = datad(2:end) * datad(1:end-1)). Depois, após todos os números repassados pelo Nios II serem salvos na memória RAM, inicia-se s1. Tanto s1 quanto s2 (isto é, dois ciclos) são utilizados para a leitura e, por conseguinte, multiplicação do par. O resultado parcial da operação, no estado s3, é salvo em R_REG – parte real – e I_REG – parte imaginária. Em s4 é verificado se todas as 128 amostras passaram pela autocorrelação; se sim, o resultado passa a estar disponível na memória para ser lido pelo Nios II; senão, é incrementado 1 aos endereços para a leitura e multiplicação do próximo par. O fluxograma do módulo system é apresentado na Figura 25.

Figura 25 - Fluxograma do módulo system

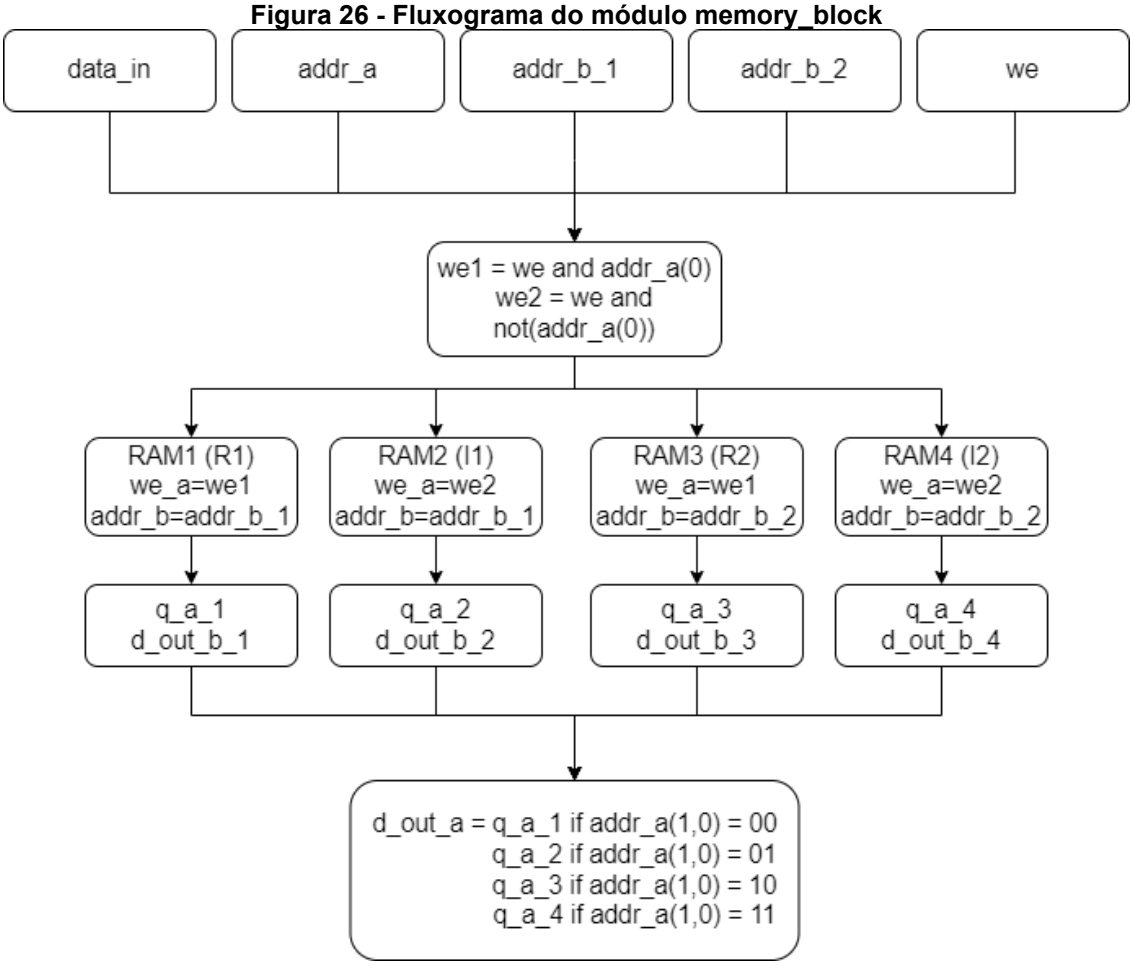


Fonte: Autoria própria, 2022

4.2.1 Módulo memory_block

Neste módulo (Figura 26) é feito o encaminhamento para ou armazenar ou ler os números que são utilizados no sistema, ou seja, R1, I1, R2 e I2. Caso seja armazenamento, we é igual a ‘1’ e data_in será utilizado, uma vez que contém o número que será armazenado. Ainda nesse caso, o bit menos significativo de addr_a indica o que será armazenado. Por outro lado, caso seja leitura, we é igual ‘0’, data_in não é utilizado e o bit menos significativo de addr_a indica de que RAM o número armazenado será lido em d_out_a. Além disso, d_out_b possibilita a leitura de um endereço adicional contido em addr_b.

Figura 26 - Fluxograma do módulo memory_block



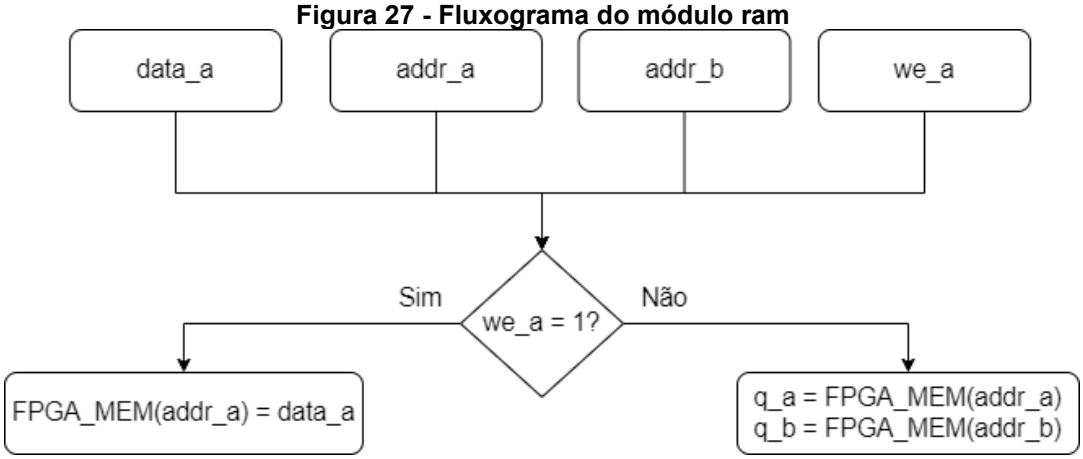
Fonte: Autoria própria, 2022

4.2.2 Módulo ram

Este módulo (Figura 27) é um bloco de memória. A entrada (data_a) contém o valor que será armazenado na memória da FPGA de acordo com we_a – variável que recebe we do módulo memory_block, a qual, se for ‘1’, armazena data_a no

endereço addr_a da memória. Por outro lado, se for ‘0’, o conteúdo da memória, tanto nos endereços addr_a quanto addr_b – variáveis que recebem addr_a e addr_b do módulo memory_block – são lidos nas saídas q_a e q_b.

Figura 27 - Fluxograma do módulo ram



Fonte: Autoria própria, 2022

4.3 Acurácia dos resultados

Os resultados exibidos na Tabela 1 e 2 foram produto da autocorrelação realizada com as amostras obtidas e pré-processadas no experimento da subseção 3.1, no qual a velocidade horizontal média do escoamento real é de 1 m/s.

Tabela 1 - Resultado dos testes práticos (autocorrelação)

Localização da amostra	Autocorrelação média (MATLAB®)	Autocorrelação média (FPGA)	Erro relativo
0,7 mm (parte real)	214217,2419131322	214217,296875	0,00002565707%
0,7 mm (parte imaginária)	-267815,3321544330i	-267815,312500i	0,0000073388%
1,4 mm (parte real)	214139,2297592207	214139,187500	0,00001973446%
1,4 mm (parte imaginária)	-858772,9651455213i	-858772,937500i	0,00000321918%
12,95 mm (parte real)	-532148,6982179735	-532148,562500	0,00002550376%
12,95 mm (parte imaginária)	-1685907,561075604i	-1685907,625000i	0,00000379169%

Fonte: Autoria própria, 2022

Tabela 2 - Resultado dos testes práticos (velocidade)

Localização da amostra	Velocidade estimada (MATLAB®)	Velocidade estimada (FPGA)	Erro relativo
0,7 mm	0,519407012587701 m/s	0,519406919299993 m/s	0,00001796042%
1,4 mm	0,768806383177909 m/s	0,768806405649854 m/s	0,00000292296%
12,95 mm	1,087659773018363 m/s	1,087659724278251 m/s	0,00000448119%

Fonte: Autoria própria, 2022

O erro entre as estimações, apesar de pequeno, se dá ao fato de que a autocorrelação realizada pelo MATLAB® considera precisão dupla (64 bits) ao passo que o algoritmo da FPGA utiliza da precisão simples (32 bits). Por outro lado, a diferença das velocidades entre as amostras se dá pelo fato de sua localização em relação ao transdutor (Figura 13). Ademais, devido a sua alta precisão, a memória utilizada da FPGA ultrapassou 65%, como mostra a Figura 28.

Figura 28 - Resumo do programa

1	Total logic elements:	7.653 / 49.760 (15 %)
2	Total memory bits:	1.128.704 / 1.677.312 (67 %)
3	Embedded Multiplier 9-bit elements:	36 / 288 (13 %)

Fonte: Autoria própria, 2022

5 CONCLUSÃO

Neste trabalho foi implementado em VHDL o método de autocorrelação para a medição de velocidade de escoamento de fluidos. Para tal, utilizou-se o ponto flutuante no padrão IEEE 754, com precisão simples. Por conseguinte, o erro máximo da velocidade comparando o resultado do MATLAB® para com a FPGA ficou abaixo de 0,00002%. Porém, com um erro tão baixo, a FPGA necessária para rodar esse algoritmo necessita de uma grande capacidade, e, consequentemente, tem um custo elevado. Para implementação em larga escala, a precisão utilizada teria de ser diminuída ao menos pela metade (16 bits).

REFERÊNCIAS

AGÊNCIA NACIONAL DE ÁGUAS (ANA). **Sistema de Informações Hidrológicas.** Brasília: ANA, 2017. Disponível em: <http://www.snirh.gov.br/hidroweb/publico/apresentacao.jsf>. Acesso em: 08 de jun. de 2022.

COUTINHO, F. R. **Um novo método ultrassônico para detecção da posição da interface em escoamentos bifásicos ar-água.** 138 f. 2014. Tese (Doutorado) - Universidade Tecnológica Federal do Paraná, Curitiba, 2014.

CRUZ, J.; BLANCO, C. J. C.; JUNIOR, A. Flow-velocity model for hydrokinetic energy availability assessment in the Amazon. **Acta Scientiarum Technology,** v. 42, p. 1-15, nov. 2019. Disponível em: <https://doi.org/10.4025/actascitechnol.v42i1.45703> Acesso em: 08 de jun. de 2022.

DESCHAMPS, J. P.; SUTTER, G.; CANTÓ, E. **Guide to FPGA Implementation of Arithmetic Functions.** Springer, 2012.

ELMORE, W. C.; HEALD, M. A. **Physics of waves.** North Chelmsford: Courier Corporation, 1985.

FIGUEIRÓ, I. C. **Arquitetura de uma unidade aritmética em ponto flutuante padrão IEEE754 32 bits.** 2011. 98 f. TCC (Graduação) - Bacharelado em Engenharia Elétrica, Universidade Federal do Pampa, Alegrete, 2011.

GONÇALVES, W. E. **Sistema para medição de distâncias horizontais na água utilizando transdutor ultrassônico.** 70 f. 2018. TCC (Graduação) - Bacharelado em Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná, Toledo, 2019.

HEXSEL, R. A. **Uma Breve Introdução à VHDL.** Universidade Federal do Paraná, p. 275-340, 2021. Disponível em: <https://www.inf.ufpr.br/roberto/ci210/vhdl.pdf>. Acesso em: 08 de jun. de 2022.

JÚNIOR, N. B. R. da. **Sistema de medição de profundidade baseado em transdutor ultrassônico.** 2017. 130 f. TCC (Graduação) - Bacharelado em Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná, Toledo, 2018.

KASAI, C.; NAMEKAWA, K.; KOYANO, A. et al. Real-Time Two-Dimensional Blood Flow Imaging Using an Auto correlation Technique. **IEEE Transactions on Sonics and Ultrasonics,** v. 32, p. 458-463, mai. 1985. Disponível em: <https://doi.org/10.1109/T-SU.1985.31615>. Acesso em: 08 de jun. de 2022.

LAUGIER, P.; HAIAT, G. Introduction to the Physics of Ultrasound. **Bone Quantitative Ultrasound,** p. 29-45, nov. 2010. Disponível em: <https://doi.org/10.1007/978-94-007-0017-8_2>. Acesso em: 08 jun. 2022.

MERRITT, C. Ultrasound Imaging. *In*: **Vascular Surgery:** Basic Science and Clinical Correlations, 2 ed., p. 315-324, out. 2007. Disponível em: <https://doi.org/10.1002/9780470987094.ch30>. Acesso em: 08 jun. 2022.

MUSTE, M.; YU, K.; SPASOJEVIC, M. Practical aspects of ADCP data use for quantification of mean river flow characteristics; Part I: moving-vessel measurements. **Flow measurement and instrumentation,** v. 15, p. 1-16, mar. 2004. Disponível em: <https://doi.org/10.1016/j.flowmeasinst.2003.09.001>. Acesso em: 08 jun. 2022.

PATTERSON, D. A.; HENNESY, J. L. Computer Architecture: **A Quantitative Approach,** 3 ed., 2003.

PEKTAŞ, A. O. Computational modeling with sensitivity analysis: case study velocity distribution of natural rivers. **Neural Computing and Applications,** v. 26, p. 1653-1667, fev. 2015. Disponível em: <https://doi.org/10.1007/s00521-015-1830-2>. Acesso em: 08 jun. 2022.

SCHMIDT, I. F. Z. TCC 2 Isadora Fernanda. **GitLab,** 2022. Disponível em: <https://gitlab.com/alissonhinacio/tcc-2-isadora-fernanda>. Acesso em: 16 dez. 2022.

TECCONCURSOS. **#472802 FUNDEP - Analista Judiciário (TJ MG)/Engenheiro Mecânico/2007.** Disponível em: <https://www.tecconcursos.com.br/questoes/472802>. Acesso em: 24 nov. 2022.

TERASIC. **DE10-Lite User Manual.** Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021&PartNo=4>. Acesso em: 08 jun. 2022.

TWINKL. **Ultrasound Wave Colouring Sheet.** Disponível em: <https://www.twinkl.com.br/resource/ultrasound-wave-colouring-sheet-t-tp-2670852>. Acesso em: 08 jun. 2022.