



Universidade Federal de Juiz de Fora

Graduação em Engenharia Elétrica

Habilitação em Sistemas Eletrônicos

**Melissa Santos Aguiar**

**Processamento Multicore Embarcado em FPGA de um Método Iterativo de  
Deconvolução Baseado em Representação Esparsa de Dados Visando a  
Reconstrução Online de Energia em Aceleradores de Partículas**

Juiz de Fora

2020

**Melissa Santos Aguiar**

**Processamento Multicore Embarcado em FPGA de um Método Iterativo de  
Deconvolução Baseado em Representação Esparsa de Dados Visando a  
Reconstrução Online de Energia em Aceleradores de Partículas**

Trabalho de Conclusão de Curso apresentado  
ao Programa de Graduação em Engenharia  
Elétrica da Universidade Federal de Juiz de  
Fora, na área de concentração em Sistemas  
Eletrônicos, como requisito parcial para a  
obtenção do título de Engenheira Eletricista.

Orientador: Luciano Manhães de Andrade Filho

Juiz de Fora

2020

Ficha catalográfica elaborada através do programa de geração  
automática da Biblioteca Universitária da UFJF,  
com os dados fornecidos pelo(a) autor(a)

Aguiar, Melissa Santos.

Processamento Multicore Embarcado em FPGA de um Método  
Iterativo de Deconvolução Baseado em Representação Esparsa de  
Dados Visando a Reconstrução Online de Energia em Aceleradores  
de Partículas / Melissa Santos Aguiar. -- 2020.

81 f.

Orientador: Luciano Manhães de Andrade Filho  
Trabalho de Conclusão de Curso (graduação) - Universidade  
Federal de Juiz de Fora, Faculdade de Engenharia, 2020.

1. Processamento Embarcado. 2. Deconvolução de Sinais. 3.  
Calorimetria. I. Andrade Filho, Luciano Manhães de, orient. II. Título.



## ATA DE APRESENTAÇÃO DE TRABALHO FINAL DE CURSO

DATA DA DEFESA: 25/11/2020 – 13:00H

CANDIDATO: MELISSA SANTOS AGUIAR

ORIENTADOR: PROF. LUCIANO MANHÃES DE ANDRADE FILHO

TÍTULO DO TRABALHO: *Processamento Multicore Embarcado em FPGA de um Método Iterativo de Deconvolução Baseado em Representação Esparsa de Dados Visando a Reconstrução Online de Energia em Aceleradores de Partículas*

BANCA EXAMINADORA/INSTITUIÇÃO:

PRESIDENTE: PROF. LUCIANO MANHÃES DE ANDRADE FILHO / UFJF

AVALIADOR 1: PROF. EDER BARBOZA KAPISCH / UFJF

AVALIADOR 2: MS. TIAGO APARECIDO TEIXEIRA / UFJF

LOCAL: POR WEBCONFERÊNCIA, CONFORME RESOLUÇÃO N° 24/2020-CONSU

Nesta data, em sessão pública, após exposição oral de 30 minutos, o candidato foi arguido pelos membros da banca. Em decorrência desta arguição, a banca considerou o candidato:

(  ) APROVADO

(  ) REPROVADO

Na forma regulamentar foi lavrada a presente Ata que é abaixo assinada pelos membros da banca na ordem determinada e pelo candidato:

PRESIDENTE: Lúcio M. de A. Filho

AVALIADOR 1: Eder Barboza Kapisch

AVALIADOR 2: Tiago Aperecido Teixeira

CANDIDATO: Melissa Santos Aguiar

*Dedico este trabalho ao Jake.*

## AGRADECIMENTOS

Eu gostaria de agradecer a todos que colaboraram de forma significativa para a concretização deste trabalho. Em especial ao Luciano, que me inspira por além de contribuir com a educação, sendo um pesquisador e professor muito presente, acessível e com alto padrão de ensino, ainda consegue se dedicar com muito talento à música. Sou muito grata e tenho sorte por você estar me orientando nesta jornada, onde a graduação é só o começo. Ao Tiago, por toda ajuda com as revisões do trabalho e também pela sua paciência e bom humor.

Agradeço ao professor Eder Barboza pelo seu comprometimento admirável com a avaliação do trabalho e também por todas as sugestões, elogios e críticas, que me motivaram e me ajudaram a melhorar a monografia para chegar na versão final.

Agradeço aos meus amigos Camila, Dabson e Jhei, que se tornaram a minha segunda família e me proporcionaram os melhores e mais divertidos momentos desde que me mudei para Juiz de Fora. A minha vida não seria a mesma sem vocês nela.

Aos amigos que fiz no decorrer da graduação, em especial ao João, Letícia, Elisa e Nathânia, que foram ótimas companhias tanto nas aulas e trabalhos quanto nos momentos de lazer. Aos amigos e colegas de pesquisa do laboratório NIPS e do grupo NIPS/CERN, em especial à Dayane, Lucca, Mariana Resende e Mariana Souza.

Agradeço também à minha família, principalmente os meus pais Simone e Rogério, que batalharam muito para me proporcionarem a oportunidade de me dedicar exclusivamente aos estudos. Eu sei que não foi nada fácil e que muitas vezes vocês precisaram abrir mão de diversas coisas para colocarem os filhos em primeiro lugar, eu espero um dia poder retribuir tudo isso. Se não fosse pelo apoio de vocês, eu não chegaria aqui.

Por fim, mas não menos importante, agradeço a cada professor e a cada professora que eu tive a honra de ser aluna nesta longa jornada. Também agradeço à Universidade Federal de Juiz de Fora pelas diversas oportunidades de ensino e inclusão, tais como o sistema de apoio estudantil e pelas bolsas de monitoria, iniciação científica e treinamento profissional, que além de me ajudarem a permanecer na cidade, foram importantes para que eu pudesse adquirir conhecimento e experiência.

“Na vida, não existe nada a temer, mas a entender”.

Marie Curie.

## RESUMO

Os laboratórios de pesquisa em Física Experimental de Altas Energias vêm colaborando com avanços significativos na ciência e tecnologia. Neles, são construídos modernos aparatos para a detecção e estudo de partículas, onde as interações podem acontecer de forma espontânea na atmosfera, quando raios cósmicos viajam em direção a Terra, ou podem também ser feitas de forma controlada, nos instrumentos denominados colisionadores de partículas, onde ocorrem interações entre partículas subatômicas com altas energias, tendo como principal objetivo o estudo de diversas características das partículas elementares. O LHC, que é atualmente o maior acelerador de partículas do mundo, está passando por um processo gradual de atualização, em que a energia das colisões está sendo aumentada, objetivando o aumento na probabilidade de ocorrerem eventos cada vez mais raros. Isto está impactando diretamente nos sistemas de instrumentação dos detectores, em especial no Calorímetro Hadrônico do Experimento ATLAS, onde o aumento na ocorrência de colisões adjacentes ocasiona o efeito de empilhamento nos sinais (*pile-up*). Devido ao fato de o algoritmo atualmente em uso na reconstrução dos sinais neste calorímetro não ser sensível ao efeito *pile-up*, diversos métodos iterativos de deconvolução de sinais baseados em Representação Esparsa de Dados vêm sendo propostos, apresentando resultados satisfatórios. No entanto, estes métodos apresentam custo computacional alto, de modo que o principal desafio atualmente é o desenvolvimento de algoritmos capazes de operar de forma *online*. Neste contexto, o presente trabalho descreve a implementação de um processador dedicado, em FPGA, com uma arquitetura *multicore*, que é capaz de executar um algoritmo iterativo baseado em Representação Esparsa de Dados visando a reconstrução *online* de energia em Calorímetros de Altas Energias, tendo como foco o Calorímetro Hadrônico do Experimento ATLAS, em cenários de empilhamento de sinais.

Palavras-chave: Processamento Embarcado. Deconvolução de Sinais. Calorimetria.

## ABSTRACT

Research laboratories in Experimental High Energy Physics have been collaborating with significant advances in science and technology. Modern apparatuses for the detection and study of particles are built in them, where interactions can happen spontaneously in the atmosphere, when cosmic rays travel towards Earth, or they can also be made in a controlled way, in instruments called particle colliders, where interactions between subatomic particles with high energies occur, in which the main objective is to study several characteristics of elementary particles. The LHC, which is currently the largest particle accelerator in the world, is undergoing a gradual process of updating, in which the energy of the collisions is being increased, aiming to increase the probability of occurring rarer events. This is having a direct impact on the instrumentation systems of the detectors, especially the Hadronic Calorimeter of the ATLAS Experiment, where the increase in the occurrence of adjacent collisions causes the pile-up effect. Due to the fact that the algorithm currently in use in the reconstruction of the signals in this calorimeter is not sensitive to the pile-up effect, several iterative methods of deconvolution of signals based on Sparse Data Representation have been proposed, presenting satisfactory results. However, these methods have a high computational cost, so the main challenge today is the development of algorithms capable of operating online. In this context, the present work describes the implementation of a dedicated processor, in FPGA, with a multicore architecture, which is capable of execute an iterative algorithm based on Sparse Data Representation aiming at the online reconstruction of energy in High Energy Calorimeters, focusing in the Hadronic Calorimeter of the ATLAS Experiment, in pile-up effect scenarios.

Key-words: Embedded Processing. Signal Deconvolution. Calorimetry.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Partículas elementares que compõem o Modelo Padrão [5]. . . . .	14
Figura 2 – Visão geral do LHC [28]. . . . .	19
Figura 3 – Complexo do LHC com todos detectores e sub-detectores [29]. . . . .	20
Figura 4 – O detector ATLAS e seus sub-sistemas [38]. . . . .	21
Figura 5 – Sistema de calorimetria do ATLAS [43]. . . . .	22
Figura 6 – Partículas interagindo com os sub-detectores do ATLAS [45]. . . . .	23
Figura 7 – Esquema de um módulo do TileCal [34]. . . . .	24
Figura 8 – Formato de um pulso característico do TileCal [47]. . . . .	24
Figura 9 – Sistema de <i>Trigger</i> do ATLAS [52]. . . . .	26
Figura 10 – Efeito <i>pile-up</i> no TileCal [10]. . . . .	27
Figura 11 – Simulação de um sinal ideal e um sinal com efeito <i>pile-up</i> . . . . .	28
Figura 12 – Função proposta para reconstrução de energia [65]. . . . .	31
Figura 13 – Diagrama do SAPHO [70]. . . . .	34
Figura 14 – Código em $C^+$ na IDE do SAPHO. . . . .	35
Figura 15 – Código em <i>Assembly</i> na IDE do SAPHO. . . . .	35
Figura 16 – Diagrama do SAPHO com o aprimoramento. . . . .	36
Figura 17 – Código em $C^+$ com o PSET. . . . .	36
Figura 18 – Código em <i>Assembly</i> com o PSET. . . . .	37
Figura 19 – Código em $C^+$ com o NORM. . . . .	38
Figura 20 – Código em <i>Assembly</i> com o NORM. . . . .	38
Figura 21 – Coeficientes da matriz de convolução. . . . .	39
Figura 22 – Compilação do Algoritmo 1 no SAPHO (versão 1.0). . . . .	40
Figura 23 – Compilação do Algoritmo 1 no Quartus (versão 1.0). . . . .	40
Figura 24 – Compilação do Algoritmo 2 no SAPHO (versão 1.1). . . . .	41
Figura 25 – Compilação do Algoritmo 2 no Quartus (versão 1.1). . . . .	41
Figura 26 – Coeficientes da matriz de autocorrelação. . . . .	42
Figura 27 – Compilação do Algoritmo 3 no SAPHO (versão 2.0). . . . .	43
Figura 28 – Compilação do Algoritmo 3 no Quartus (versão 2.0). . . . .	43
Figura 29 – Compilação do Algoritmo 3 no SAPHO (versão 2.1). . . . .	43
Figura 30 – Compilação do Algoritmo 3 no Quartus (versão 2.1). . . . .	44
Figura 31 – Compilação do Algoritmo 4 no SAPHO (versão 2.2). . . . .	44
Figura 32 – Compilação do Algoritmo 4 no Quartus (versão 2.2). . . . .	45
Figura 33 – Coeficientes da matriz pseudo-inversa. . . . .	45
Figura 34 – Erro RMS da estimativa de energia pelo método SSF [70]. . . . .	46
Figura 35 – Compilação do Algoritmo 5 no Quartus (versões 3.0 e 3.1). . . . .	47
Figura 36 – Compilação do Algoritmo 5 no SAPHO (versão 3.0). . . . .	47
Figura 37 – Compilação do Algoritmo 5 no SAPHO (versão 3.1). . . . .	47
Figura 38 – Relação entre o valor RMS do erro e o ganho. . . . .	48

Figura 39 – Compilação do método SSF em ponto fixo no SAPHO. . . . .	48
Figura 40 – Compilação do método SSF em ponto fixo no Quartus. . . . .	49
Figura 41 – Compilação do método SSF em ponto fixo (final) no SAPHO. . . . .	49
Figura 42 – Compilação do método SSF em ponto fixo (final) no Quartus. . . . .	50
Figura 43 – Diagrama da estrutura <i>multicore</i> implementada. . . . .	50
Figura 44 – Código para o preenchimento da janela de entrada de dados. . . . .	51
Figura 45 – Quantidade de processadores e tempo de atraso variando com a frequência. . . . .	54
Figura 46 – Elementos lógicos e multiplicadores variando com a frequência. . . . .	54
Figura 47 – Bits de memória variando com a frequência. . . . .	55
Figura 48 – Janela de entrada dos dados do método SSF no Modelsim-Altera. . . . .	56
Figura 49 – Comparação entre o sinal reconstruído, o alvo e o sinal ruidoso. . . . .	56
Figura 50 – Operações do Algoritmo 1. . . . .	68
Figura 51 – Operações do Algoritmo 2. . . . .	68
Figura 52 – Operações do Algoritmo 3. . . . .	68
Figura 53 – Operações do Algoritmo 4. . . . .	68
Figura 54 – Operações do Algoritmo 5. . . . .	68
Figura 55 – Tutorial SAPHO: criando um novo projeto. . . . .	76
Figura 56 – Tutorial SAPHO: configurando o diretório do projeto. . . . .	76
Figura 57 – Tutorial SAPHO: criando um processador. . . . .	76
Figura 58 – Tutorial SAPHO: configurando os parâmetros do processador. . . . .	77
Figura 59 – Tutorial SAPHO: interface de desenvolvimento em <b>C<sup>+</sup></b> . . . . .	77
Figura 60 – Tutorial SAPHO: compilando o código. . . . .	78
Figura 61 – Tutorial SAPHO: código em <i>Assembly</i> gerado. . . . .	78
Figura 62 – Tutorial SAPHO: abrindo o projeto no Quartus. . . . .	79
Figura 63 – Tutorial SAPHO: selecionando os arquivos de parametrização. . . . .	79
Figura 64 – Tutorial SAPHO: resumo do projeto criado no Quartus. . . . .	80
Figura 65 – Tutorial SAPHO: selecionando o arquivo principal. . . . .	80
Figura 66 – Tutorial SAPHO: compilando o projeto. . . . .	81
Figura 67 – Tutorial SAPHO: visualizando o processador criado. . . . .	81
Figura 68 – Tutorial SAPHO: diagrama do <i>top level</i> do processador desenvolvido. .	81

## **LISTA DE TABELAS**

Tabela 1 – Ferramentas e energia mínima necessárias para explorar distâncias [26].	18
Tabela 2 – Energia e luminosidade de alguns aceleradores de partículas [26]. . . . .	18
Tabela 3 – Comparação entre as implementações do método SSF. . . . . . . . .	53

## LISTA DE ABREVIATURAS E SIGLAS

ALICE	<i>A Large Ion Collider Experiment</i>
ATLAS	<i>A Toroidal LHC ApparatuS</i>
BC	<i>Bunch Crossing</i>
CD	<i>Coordinate Descent</i>
CERN	<i>Conseil Européen pour la Recherche Nucléaire</i>
CMS	<i>Compact Muon Solenoid</i>
FIR	<i>Finite Impulse Response</i>
FPGA	<i>Field Programmable Gate Array</i>
GD	<i>Gradient Descent</i>
IDE	<i>Integrated Development Environment</i>
LAr	<i>Liquid Argon</i>
LHC	<i>Large Hadron Collider</i>
LHCb	<i>A Large Ion Collider beauty</i>
LHCf	<i>The Large Hadron Collider forward</i>
LP	<i>Linear Programming</i>
NIPS	<i>Núcleo de Instrumentação e Processamento de Sinais</i>
PMT	<i>PhotoMultiplier Tube</i>
PSC	<i>Processador Soft-Core</i>
RAM	<i>Random Acess Memory</i>
RMS	<i>Root Mean Square</i>
SAPHO	<i>Scalable-Architecture Processor for Hardware Optimization</i>
SR	<i>Sparse Representation</i>
SSF	<i>Separable Surrogate Functionals</i>
TileCal	<i>Tile Calorimeter</i>
TOTEM	<i>TOtal Elastic and diffractive cross section Measurement</i>
ULA	<i>Unidade Lógico-Aritmética</i>

## LISTA DE SÍMBOLOS

$p$	Momento de uma partícula
$E$	Energia
$m$	Massa
$c$	Velocidade da luz no vácuo
$\Delta x$	Distância
$\mathcal{L}_u$	Luminosidade
$\leq$	Menor ou igual que
$P_\ell$	Problema- $\ell$
$ \mathbf{x} $	Módulo de $\mathbf{x}$
$\ \mathbf{x}\ _\ell^\ell$	Norma- $\ell$ de $\mathbf{x}$
$\sum_i \mathbf{x}_i$	Somatório em $i$ de $\mathbf{x}_i$
$\epsilon_0$	Erro quadrático
$\lambda$	Multiplicador de Lagrange
$\mathbf{x}_{opt}$	Valor ótimo de $\mathbf{x}$
$\mu$	Tamanho do passo em direção ao mínimo
$S_\lambda(\theta)$	Função de Shrinkage
$\ \mathbf{x}\ _2^2$	Norma quadrática euclidiana de $\mathbf{x}$
$\mathcal{L}(\mathbf{x})$	Lagrangiano de $\mathbf{x}$
$\partial$	Derivada parcial
$M^T$	Transposta da matriz $\mathbf{M}$
$M^{-1}$	Inversa da matriz $\mathbf{M}$
$M^+$	Pseudo-inversa da matriz $\mathbf{M}$
@	Função que implementa a instrução PSET
/ >	Função que implementa a instrução NORM
$\Delta t$	Tempo de atraso

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO E MOTIVAÇÃO . . . . .</b>	<b>14</b>
1.1	OBJETIVOS . . . . .	16
1.2	ESTRUTURA DO TRABALHO . . . . .	16
<b>2</b>	<b>ACELERADORES DE PARTÍCULAS . . . . .</b>	<b>17</b>
2.1	LHC - <i>LARGE HADRON COLLIDER</i> . . . . .	19
2.2	EXPERIMENTO ATLAS . . . . .	21
2.2.1	Calorímetro Hadronico . . . . .	23
2.2.2	Sistema de <i>Trigger</i> . . . . .	25
2.3	DESAFIO: O EFEITO <i>PILE-UP</i> . . . . .	27
<b>3</b>	<b>REVISÃO BIBLIOGRÁFICA . . . . .</b>	<b>29</b>
3.1	REPRESENTAÇÃO ESPARSA PARA ESTIMAÇÃO DE ENERGIA .	30
3.1.1	Aprimoramento na Inicialização do Algoritmo . . . . .	32
<b>4</b>	<b>IMPLEMENTAÇÃO . . . . .</b>	<b>33</b>
4.1	SAPHO: UM PROCESSADOR AUTO-ESCALÁVEL . . . . .	33
4.1.1	Otimizações na Estrutura do Processador . . . . .	36
4.2	MÉTODO OPERANDO EM PONTO FLUTUANTE . . . . .	39
4.2.1	Inicialização Otimizada do Algoritmo . . . . .	45
4.3	MÉTODO OPERANDO EM PONTO FIXO . . . . .	47
4.4	ESTRUTURA MULTICORE . . . . .	50
<b>5</b>	<b>RESULTADOS . . . . .</b>	<b>53</b>
<b>6</b>	<b>CONCLUSÕES . . . . .</b>	<b>57</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>58</b>
	<b>APÊNDICE A – Produção Bibliográfica . . . . .</b>	<b>64</b>
	<b>ANEXO A – Algoritmos Implementados no SAPHO . . . . .</b>	<b>68</b>
	<b>ANEXO B – Características das Implementações Realizadas .</b>	<b>74</b>
	<b>ANEXO C – Tutorial: Criando um Projeto com o SAPHO .</b>	<b>76</b>

## 1 INTRODUÇÃO E MOTIVAÇÃO

O processo da exploração de questões fundamentais sobre a composição do átomo e das partículas básicas que constituem a matéria necessita de estudos em Física de Altas Energias [1], por meio de experimentos envolvendo um refinado sistema de instrumentação: os colisionadores de partículas [2]. Tais experimentos consistem em acelerar e colidir feixes de partículas a velocidades próximas à da luz, gerando ambientes que se assemelham ao estado do universo nos instantes iniciais após o *Big Bang* [3]. O objetivo é que os feixes de partículas aceleradas colidam nas regiões de interesse onde estão presentes os detectores, de forma que é possível medir as propriedades das partículas subprodutos destas colisões.

As colisões entre partículas nos aceleradores geram uma grande quantidade de energia, permitindo a observação de elementos que antes eram desconhecidos para a ciência. Na Figura 1 é possível ver o diagrama das partículas que constituem o Modelo Padrão<sup>1</sup>, destacando-se o Bóson de Higgs, que havia sido previsto teoricamente por esse modelo e finalmente teve a existência confirmada [4], sendo atualmente um dos maiores exemplos de descobertas feitas por meio de experimentos realizados com aceleradores de partículas.

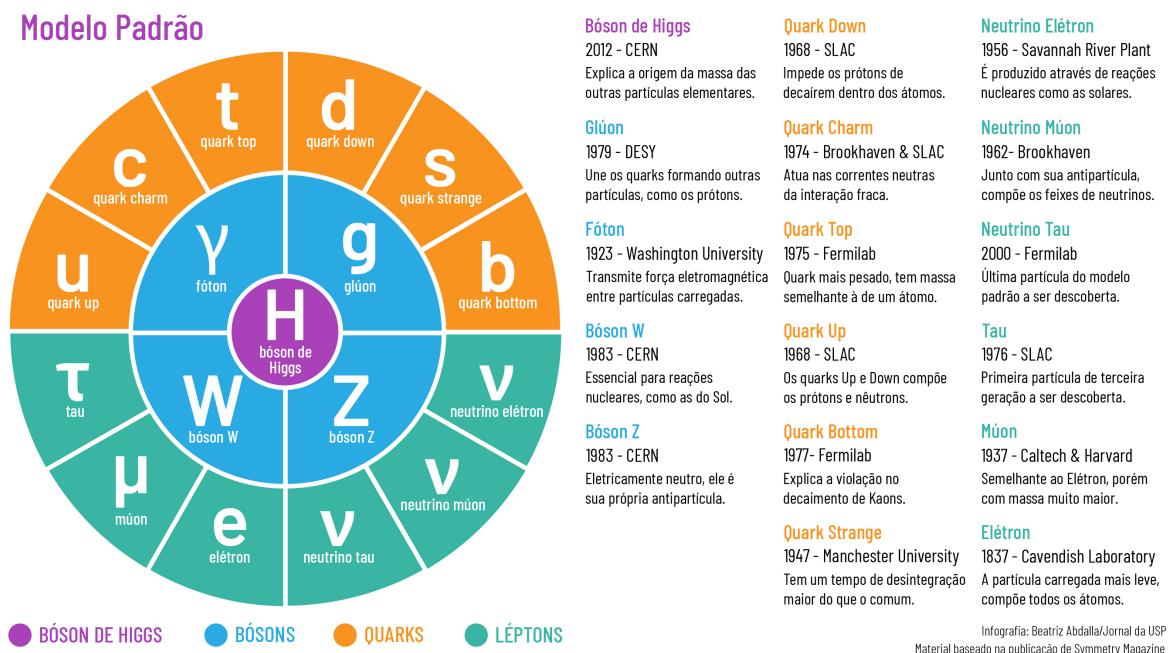


Figura 1 – Partículas elementares que compõem o Modelo Padrão [5].

Cerca de 5% da matéria conhecida do universo é explicada pelo Modelo Padrão, já os outros 95% são compostos pela matéria escura, que ainda não foi detectada diretamente pelos instrumentos e experimentos desenvolvidos até então [6].

<sup>1</sup> O Modelo Padrão é uma teoria da Física de Partículas que descreve as forças fundamentais fortes, fracas, eletromagnéticas e as partículas fundamentais que compõem a matéria [7].

Na prática, os aceleradores de partículas são equipamentos construídos para acelerar e aumentar a energia de feixes de partículas, através da geração de campos elétricos e campos magnéticos que são suficientemente fortes para orientar e focalizar os feixes [8]. Com isso, é possível mensurar a energia gerada pelas colisões, que é uma importante propriedade para a caracterização das partículas subprodutos destas colisões.

Para garantir a confiabilidade dos dados medidos, é necessário um sofisticado sistema de processamento, de forma a permitir a correta interpretação de determinados eventos físicos. Tal sistema pode ser subdividido entre dois tipos de processamento: o *online* e o *offline* [9]. O processamento *online* é responsável por realizar a seleção das informações na medida em que as colisões ocorrem, em tempo real, respeitando a latência do sistema, enquanto o *offline* analisa os dados já armazenados após as colisões [10].

No contexto desta demanda por complexos sistemas de instrumentação, que contribuem em explorações como as componentes fundamentais da matéria e suas dinâmicas de interação, a Organização Europeia para a Pesquisa Nuclear (*Conseil Européen pour la Recherche Nucléaire*), também conhecida como CERN, fundada no ano de 1954 [11], em Genebra, Suíça, vem se destacando mundialmente nos experimentos envolvendo a Física de Altas Energias. Tais experimentos são extremamente importantes na compreensão da evolução do universo desde o *Big Bang* até os dias atuais.

O Experimento ATLAS é responsável pela aquisição dos dados resultantes das colisões que ocorrem no LHC (*Large Hadron Collider*), que é um dos principais aceleradores de partículas do CERN. O detector ATLAS é formado por sub-detectores dispostos em camadas, sendo cada um destes responsável por medir propriedades específicas das partículas geradas pelas colisões. Uma importante característica a ser mensurada é a energia gerada nas colisões, que é obtida através dos calorímetros [12]. Tal energia é medida pela estimativa da amplitude dos sinais que são gerados na eletrônica de leitura dos calorímetros, quando as partículas interagem com seu material absorvedor.

No Calorímetro Hadrônico do ATLAS, devido ao fato de o período necessário para a identificação de um pulso conformado ser maior que o intervalo entre as colisões, ocorre um efeito de empilhamento enquanto o sinal se desenvolve. Buscando resolver este problema, foram propostos métodos que se baseiam na deconvolução destes sinais para recuperar a sua amplitude [10, 13]. Tais métodos têm sido implementados com filtros digitais simples do tipo FIR (*Finite Impulse Response*) [14]. Ainda no contexto de resolver o problema do empilhamento de sinais, uma abordagem por meio de métodos baseados em teoria de Representação Esparsa de dados tem se destacado quanto à eficiência na reconstrução dos sinais [15], porém, tais métodos possuem custo computacional muito elevado para implementação online.

## 1.1 OBJETIVOS

O objetivo deste trabalho é realizar a implementação em *hardware* do SSF (*Separable Surrogate Functionals*) [16], um método iterativo de deconvolução baseado em Representação Esparsa de dados, visando a reconstrução *online* de energia em Calorímetros de Altas Energias, tendo como foco o Calorímetro Hadrônico do Experimento ATLAS.

Para tal, foi projetado um processador dedicado, em FPGA (*Field Programmable Gate Array*) [17], capaz de realizar as operações do algoritmo proposto através de circuitos aritméticos em ponto fixo e em ponto flutuante. Além disso, foi desenvolvida uma arquitetura *multicore* (vários núcleos) para o processamento respeitar os requisitos temporais de operação, que são necessários devido à alta taxa de eventos no calorímetro.

## 1.2 ESTRUTURA DO TRABALHO

Esta monografia está organizada da seguinte forma: no Capítulo 1 foram apresentados a contextualização, motivação e os objetivos do trabalho.

O ambiente deste trabalho é introduzido no Capítulo 2, com destaque para o Experimento ATLAS e seus sistemas de calorimetria e de filtragem *online*, bem como a definição do problema do empilhamento de sinais.

É apresentada, no Capítulo 3, uma introdução sobre o método atualmente em uso na reconstrução dos sinais, suas limitações para lidar com o problema de empilhamento e o embasamento matemático da metodologia a ser implementada. É proposto também um aprimoramento para otimizar a inicialização deste método.

O processador embarcado desenvolvido e as modificações realizadas em sua estrutura para otimizar seu funcionamento são descritos no Capítulo 4. Neste capítulo, é proposta também uma arquitetura *multicore* para o processamento, visando estimar a energia respeitando os requisitos da taxa de eventos no calorímetro.

O ambiente de simulação utilizado é introduzido no Capítulo 5, onde os resultados obtidos são discutidos e as comparações entre as implementações propostas são realizadas.

No Capítulo 6 são apresentadas as conclusões gerais deste trabalho, as suas principais contribuições e propostas de trabalhos futuros para a continuidade da pesquisa.

O Apêndice A possui um resumo dos artigos apresentados em congressos nacionais que foram publicados no decorrer deste trabalho.

O Anexo A contém uma descrição dos algoritmos de cada implementação em FPGA do método SSF que foi realizada. O Anexo B contém um resumo com as principais características de cada versão do processador. O Anexo C contém um tutorial detalhando a criação de processadores utilizando a ferramenta apresentada no Capítulo 4 deste trabalho.

## 2 ACELERADORES DE PARTÍCULAS

Para realizar estudos de objetos cada vez menores, os limites da tradicional Mecânica Newtoniana [18] acabam sendo ultrapassados e, a partir desse ponto, as leis da Mecânica Quântica [19] passam a descrever melhor o comportamento destes objetos. Assim, os aceleradores de partículas podem ser comparados com grandes microscópios e, a partícula, regida pelas leis da Mecânica Quântica, passa a ser vista como uma entidade dual. Tal dualidade é conhecida como partícula-onda [20], e muitos aspectos do comportamento desta entidade só podem ser descritos em termos de probabilidades, onde esta entidade dual é modelada como sendo inherentemente espalhada, não podendo mais ser descrita como um ponto com posição e velocidade determinadas, como seria na Mecânica Newtoniana.

Esta entidade dual pode ser visualizada como uma nuvem de probabilidades, tendo suas dimensões comparáveis ao comprimento de onda, de forma que a partícula fisicamente visível pode ser encontrada em algum ponto desta nuvem de probabilidades. Quando o comprimento de onda é reduzido, o volume da nuvem também é reduzido. Porém, para isso ocorrer, faz-se necessário aumentar o momento da partícula, ou seja, sua energia precisa ser aumentada. Portanto, ao aumentar a energia do experimento, é possível estudar objetos com dimensões cada vez menores [21]. Nesse contexto, os aceleradores de partículas são ferramentas que permitem a observação de estruturas muito pequenas, através da produção de partículas com alto momento transverso e, consequentemente, comprimento de onda curto, uma vez que, segundo o princípio da incerteza de Heisenberg [22], o comprimento de onda associado é inversamente proporcional ao momento da partícula ( $p$ ) [23].

Além disso, segundo os estudos da relatividade de Albert Einstein [24], com a famosa equação  $E = mc^2$ , descobre-se que a energia pode ser convertida em matéria e que a recíproca também é válida. Assim, os aceleradores podem criar partículas ao fazer com que dois feixes de partículas se choquem, possibilitando então o estudo das partículas fundamentais [25]. Na prática, grandes detectores em diversas camadas são posicionados ao redor do ponto de colisão nos experimentos mais modernos, onde cada camada de detecção tem uma função específica na determinação da trajetória e na identificação de cada uma das muitas partículas que podem ser produzidas em uma única colisão. Assim, quanto maior a energia das partículas, ou seja, quanto maior a aceleração aplicada a estas, maior a eficiência na produção de partículas elementares.

Na Tabela 1 é possível observar algumas ordens de grandeza da energia mínima necessária para a exploração de distâncias  $\Delta x$  [26]. Como já era esperado, devido a teoria da dualidade entre partícula e onda, é possível notar que, para explorar as menores distâncias, são necessários valores de energia mais altos e, além disso, as ferramentas utilizadas para a obtenção das mais altas energias são os aceleradores de partículas.

Tabela 1 – Ferramentas e energia mínima necessárias para explorar distâncias [26].

Ferramentas Utilizadas	Distância (centímetro)	Energia (elétron-volt*)
Microscópios	$10^{-5}$	2 eV
Raios-X	$10^{-8}$	2 keV
Raios- $\gamma$	$10^{-11}$	2 MeV
Aceleradores de Partículas	$10^{-14}$	2 GeV
Aceleradores de Partículas	$10^{-16}$	200 GeV
Aceleradores de Partículas	$10^{-17}$	2 TeV

\* elétron-volt é a unidade de energia definida como o trabalho realizado ao se mover um elétron através de uma diferença de potencial de 1 volt. 1 eV equivale a  $1,6 \times 10^{-19}$  joules.

Um acelerador de partículas é basicamente composto por uma fonte de íons, um campo acelerador e um campo que força a partícula a se mover em uma órbita bem definida. O acelerador de alvo fixo lança um conjunto de partículas contra um alvo fixo de forma a promover uma colisão entre ambos. O espalhamento das partículas resultantes desta colisão fornece informações sobre a estrutura do feixe e do alvo. Já nos colisores de feixes, são acelerados e direcionados feixes de partículas para colidirem com igual momento e sinais opostos, utilizando melhor a energia envolvida na colisão. Estes são menos versáteis e produzem um número menor de interações por unidade de tempo quando comparados com os de alvo fixo, porém, eles têm a vantagem de obterem energias maiores [26].

Uma importante característica dos aceleradores de partículas é a luminosidade  $\mathcal{L}_u$ , que é definida como o número que, multiplicado pela seção transversal total de um dado processo  $\sigma$ , fornece o número total de colisões por unidade de tempo  $N$ , ou seja,  $N = \mathcal{L}_u \sigma$ .

Atingir uma alta luminosidade é um dos principais objetivos dos aceleradores de partículas, uma vez que a maioria dos eventos produzidos durante as colisões não são de interesse, sendo estes conhecidos como eventos de *minimum bias* [26].

Na Tabela 2 são indicadas características de energia e luminosidade de alguns dos principais aceleradores de partículas do CERN.

Tabela 2 – Energia e luminosidade de alguns aceleradores de partículas [26].

Acelerador	$\mathcal{L}_u (cm^{-2}s^{-1})$	Energia (elétron-volt)
Super Proton Synchrotron (1981)	$6 \times 10^{30}$	630-900 GeV
Large Electron-Positron (1989)	$10^{31-32}$	90-200 GeV
Large Hadron Collider (2010-2012)	$0,76 \times 10^{34}$	7-8 TeV
Super Large Hadron Collider (2015)	$10^{34}$	14 TeV

## 2.1 LHC - LARGE HADRON COLLIDER

O Grande Colisor de H  drons ou LHC, do ingl  s *Large Hadron Collider*,    atualmente o principal acelerador de part  culas do CERN, sendo considerado o maior e mais energ  tico acelerador de part  culas j   constru  ido [27]. Ele    composto por um anel de im  s supercondutores de 27 km de circunfer  ncia, com v  rias estruturas aceleradoras para aumentar a energia das part  culas ao longo do caminho, situando-se a 100 metros de profundidade entre as fronteiras da Fran  a e da Su  ica, como ilustrado na Figura 2.

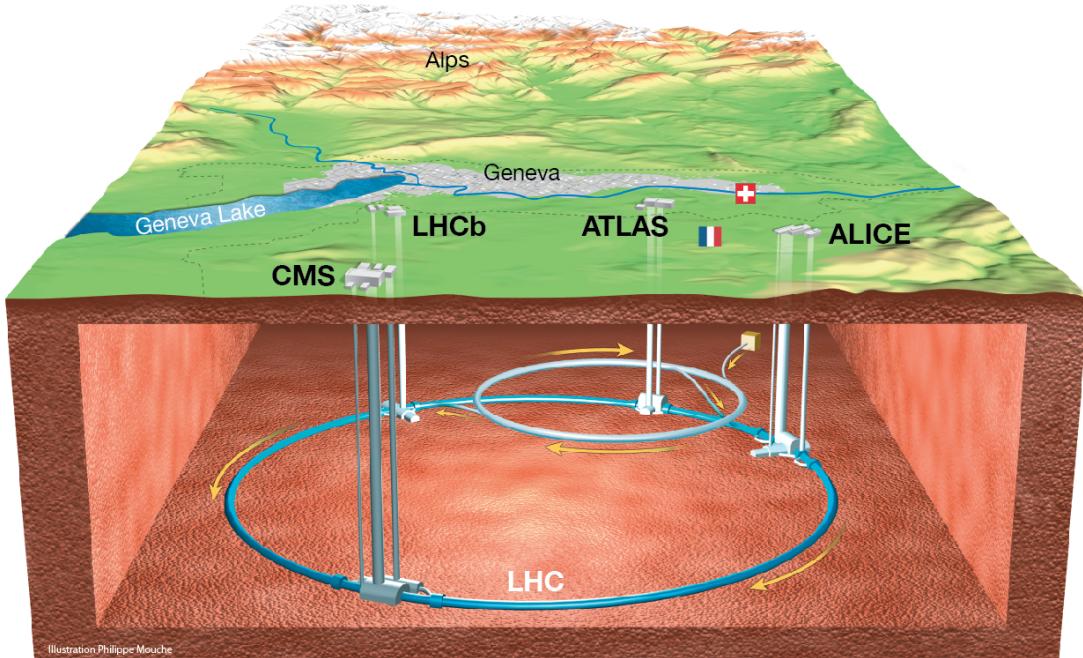


Figura 2 – Vis  o geral do LHC [28].

O complexo do LHC [29], que pode ser visto na Figura 3, possui seis detectores em pontos estrat  gicos (onde ocorrem as colisões) ao longo de sua circunfer  ncia, sendo estes, os instrumentos respons  veis pela aquisi  o de dados resultantes dos eventos, ao possibilitarem a reconstru  o das colisões ocorridas, de forma a permitir a detec  o das part  culas subprodutos destas colisões [30].

Estes detectores s  o baseados em um modelo de camadas, onde cada uma delas    respons  vel por estimar as propriedades espec  ficas dos diferentes tipos de part  culas. Dentre elas, h   as part  culas que interagem de forma eletromagn  tica, que incluem os el  trons, f  tons e p  ositrons, e h   tamb  m as part  culas hadr  nicas, que compreendem os pr  tons e n  utronos, que s  o part  culas que interagem mais evidentemente atrav  s de for  a forte. Com rela  o aos detectores do LHC, as principais caracter  sticas destacadas s  o:

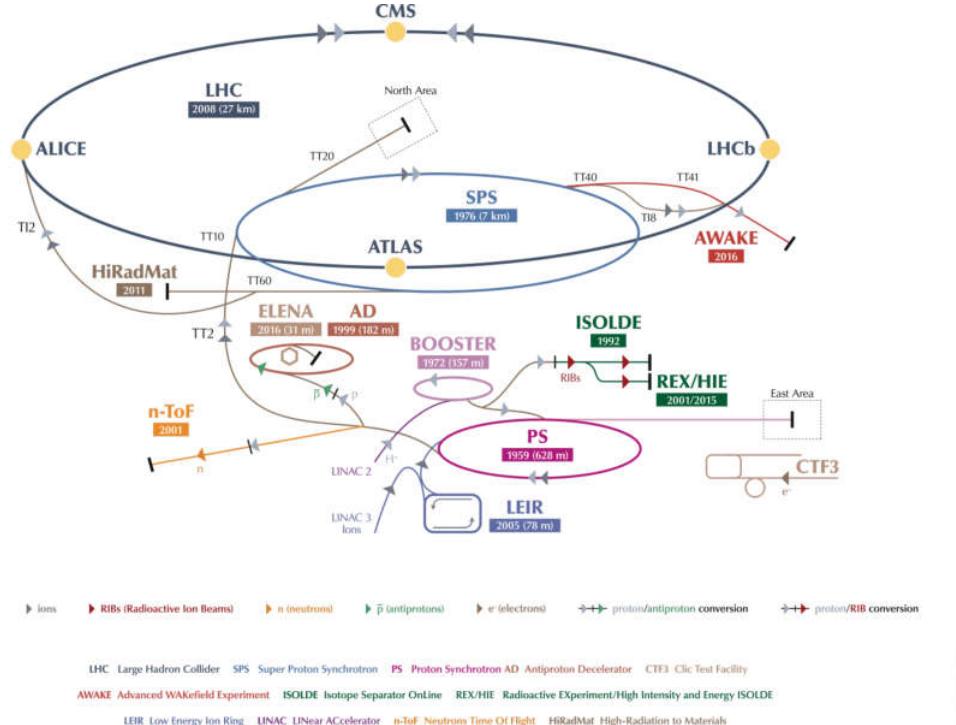


Figura 3 – Complexo do LHC com todos detectores e sub-detectores [29].

- **CMS – Compact Muon Solenoid** [31]

É um detector de propósito geral, que foi projetado para estudar o Bóson de Higgs, partículas supersimétricas e a física de íons pesados.

- **LHCb – A Large Ion Collider beauty** [32]

Detector dedicado a pesquisas sobre a simetria aparente entre a matéria e a anti-matéria presentes no universo.

- **ALICE – A Large Ion Collider Experiment** [33]

É o único detector especializado em colisões de íons de chumbo cujo principal objetivo é o estudo da formação e propriedades do plasma de *quark-gluon*, uma matéria que se acredita apenas ter existido por poucos instantes após o *Big Bang*.

- **ATLAS – A Toroidal LHC ApparatuS** [34]

Experimento de propósito geral, que foi otimizado para detectar o maior número possível de eventos físicos que ocorrerão no LHC.

- **TOTEM e LHCf – TOTAL Elastic and diffractive cross section Measurement** [35] e **The Large Hadron Collider forward** [36]

Experimentos de pequeno porte dedicados à física projetiva (*forward*) de prótons e íons pesados.

## 2.2 EXPERIMENTO ATLAS

O ATLAS é um dos principais experimentos do LHC, possuindo propósito geral para a detecção das colisões do tipo próton-próton. Sua colaboração envolve cerca de 38 países e 174 institutos de pesquisa, contando com mais de 3.000 cientistas ao redor do mundo, sendo projetado com foco no estudo da maior quantidade possível de fenômenos físicos passíveis de serem gerados em colisões no LHC [37], desde a busca pelo bóson de Higgs até dimensões extras e partículas que possam constituir a matéria escura.

O detector ATLAS, ilustrado na Figura 4, pesa cerca de 7.000 toneladas, e suas dimensões são de aproximadamente 25 metros de altura por 44 metros de largura. Ele tem formato cilíndrico e cobre um ângulo sólido próximo a  $4\pi$  ao redor da região de colisão das partículas. Ainda nesta figura, é possível notar que, além dos magnetos responsáveis pela geração de intensos campos magnéticos que auxiliam na medida de momento das partículas carregadas, o ATLAS é composto também por três sub-detectores básicos: o detector de trajetórias, os calorímetros eletromagnético e hadrônico e, por fim, o detector de mûons, respectivamente, ordenados do mais interno para o mais externo.

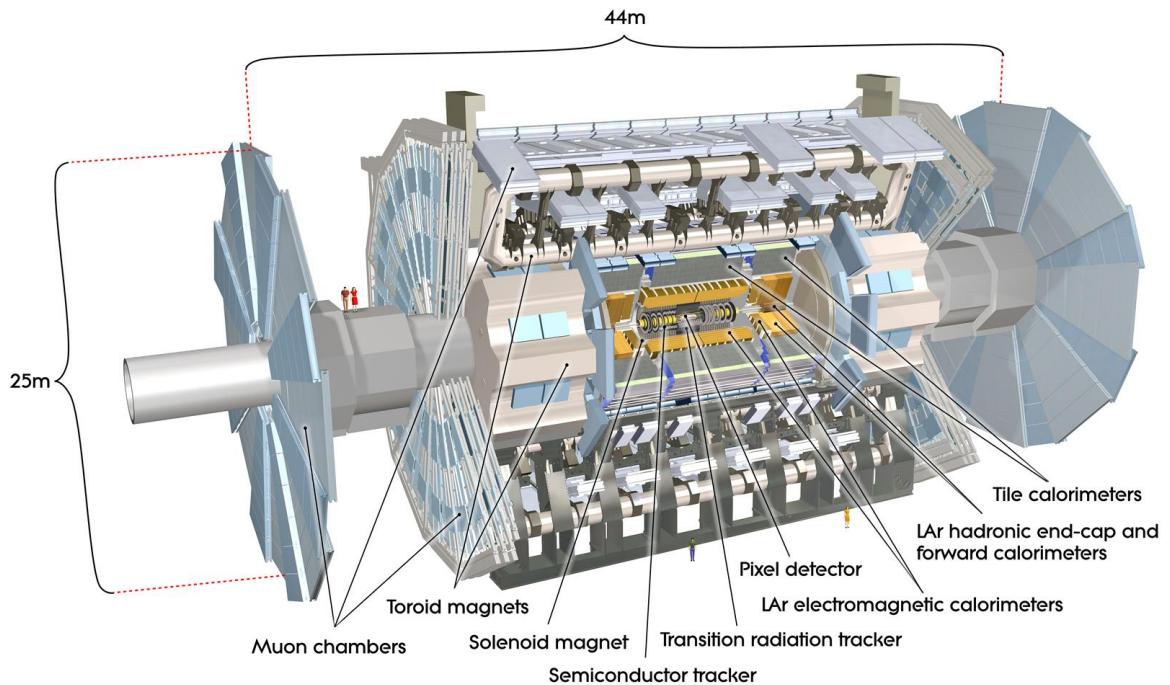


Figura 4 – O detector ATLAS e seus sub-sistemas [38].

O Detector de Trajetórias (*Inner Detector*), que está na camada mais interna, é subdividido em três sub-detectores (*Pixel Detector*, *Semiconductor Tracker* e *Transition Radiation Tracker*), com a função de determinar os traços das partículas carregadas, auxiliando na determinação do seu momento e posição [39].

O Espectrômetro de Múons (*Muon Spectrometer*) identifica e mede o momento dos mûons, se localizando na camada mais externa do detector, uma vez que os mûons são as únicas partículas detectáveis que alcançam distâncias tão grandes [40], além do ponto de colisão.

No sistema de calorimetria do ATLAS (Figura 5), que está localizado na camada intermediária do detector, o Calorímetro de Argônio Líquido (LAr - *Liquid Argon*), também chamado de Calorímetro Eletromagnético, foi projetado para medir a energia das partículas que interagem de forma eletromagnética (elétrons e fôtons) com seu material [41]. O Calorímetro Hadrônico (TileCal - *Tile Calorimeter*), que também é conhecido como Calorímetro de Telhas, foi projetado para mensurar a energia das partículas mais propícias a interagirem de forma hadrônica (principalmente hádrons neutros) com seu material [42].

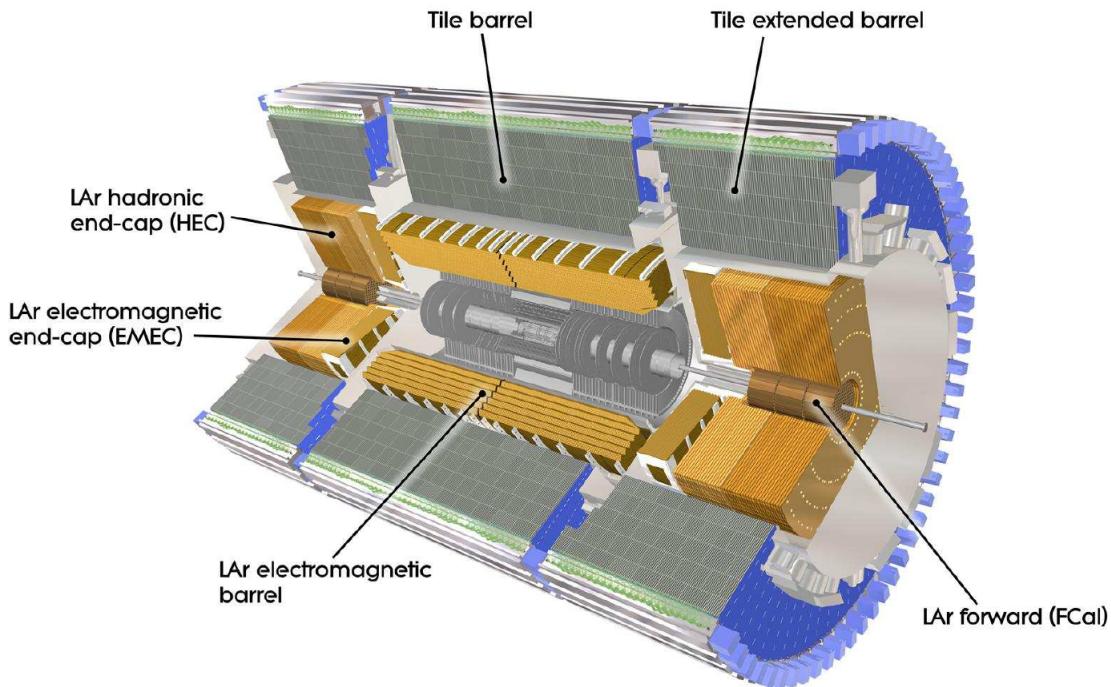


Figura 5 – Sistema de calorimetria do ATLAS [43].

Na prática, os calorímetros têm a função de absorver, amostrar e medir a energia das partículas que os incidem. Estas partículas, ao entrarem em contato com o material dos calorímetros, geram um chuveiro de partículas, onde parte de sua energia é depositada, coletada e medida, o que é possível devido aos calorímetros serem compostos por um material denso, formando barreiras que permitem que as partículas sejam absorvidas por completo [39]. Os mûons, de alta energia, não são absorvidos pelo experimento, depositando apenas uma pequena parte de sua energia nos calorímetros. Quando o processo de chuveiro é iniciado, as partículas sofrem decaimentos, produzindo partículas de menor energia, e este processo se dá até a absorção total da energia da partícula pelo calorímetro [44].

Na Figura 6 é possível observar um esquema com diversos tipos de partículas interagindo com os sub-detectores do ATLAS, após uma colisão, onde é possível identificar e caracterizar as partículas e acordo com as características mensuradas quando elas atravessam as camadas de detecção.

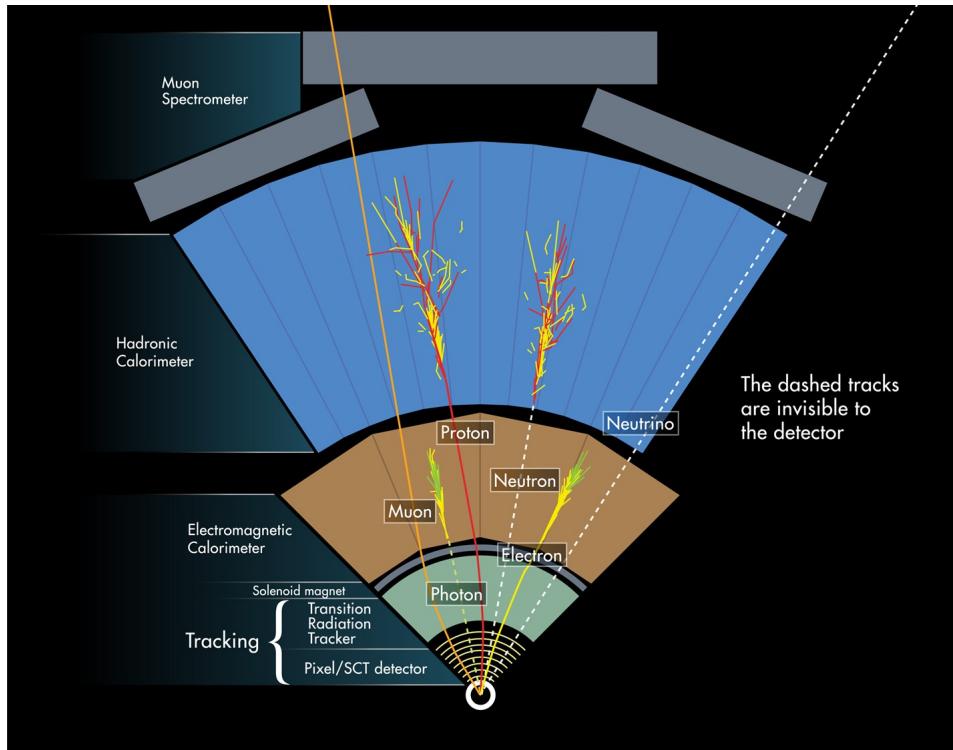


Figura 6 – Partículas interagindo com os sub-detectores do ATLAS [45].

### 2.2.1 Calorímetro Hadrônico

O Calorímetro Hadrônico do ATLAS possui o aço como material absorvedor e telhas cintilantes como material amostrador de energia. Estas telhas são excitadas quando partículas carregadas as atravessam, ocorrendo a produção de fôtons. Com isto, há a conversão de luz em sinal elétrico, que se dá por células fotomultiplicadoras também conhecidas como PMT's (*PhotoMultiplier Tube*). A luz gerada nos cintiladores é levada às células multiplicadoras por fibras óticas, localizadas nas duas extremidades das telhas, para que haja redundância na leitura e, consequentemente, maior confiabilidade [46].

Com a finalidade de se obter um sinal de pulso padrão, o sinal convertido pelo PMT é repassado a um circuito conformador de pulsos, e o pulso final apresenta um formato determinístico e uma amplitude proporcional à energia que a partícula depositou. As células do TileCal são formadas por conjuntos de telhas cintilantes, onde cada célula é lida por duas PMT's, visando redundância, e cada PMT corresponde a um canal de leitura, havendo, no total, cerca de 10.000 canais de leitura [44].

Na Figura 7, está ilustrado um módulo do TileCal com vista tri-dimensional, onde é possível observar as telhas dispostas perpendicularmente à direção do feixe de partículas.

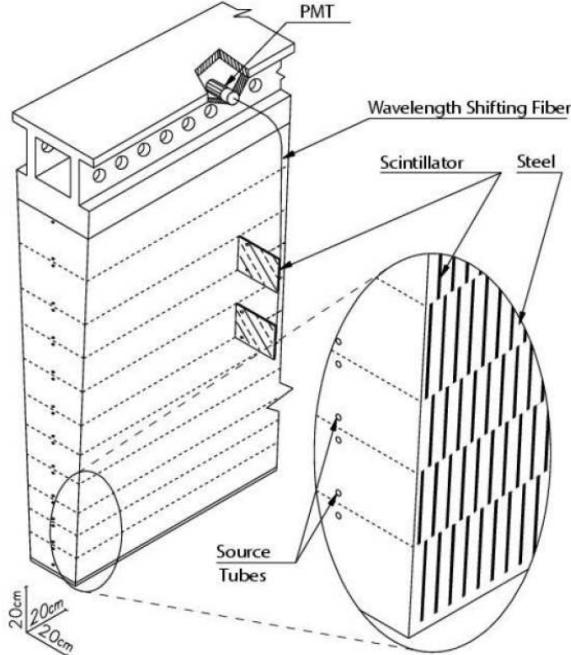


Figura 7 – Esquema de um módulo do TileCal [34].

O sinal luminoso é convertido em um sinal elétrico nos PMT's e transmitido para um circuito de condicionamento e amplificação de sinal analógico, resultando em um pulso de forma fixa, onde a amplitude é proporcional à energia depositada nas células [44]. Este pulso característico (Figura 8) é composto por 7 amostras espaçadas de 25 ns (durando cerca de 150 ns), digitalizado a uma taxa de 40 MHz, sincronizada com a taxa de colisão [47].

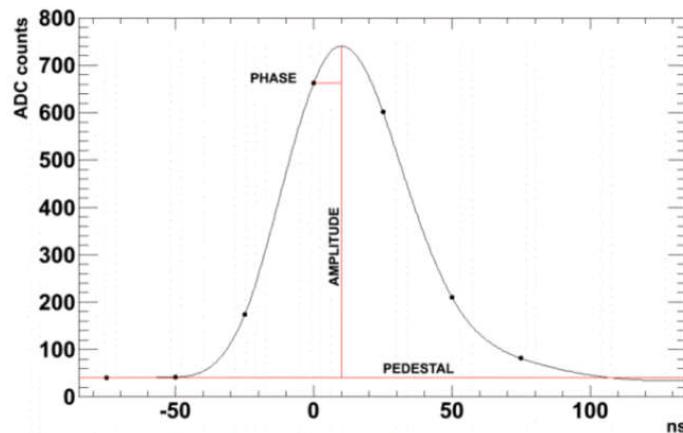


Figura 8 – Formato de um pulso característico do TileCal [47].

Em resumo, o principal objetivo do TileCal é contribuir na reconstrução da energia dos chuveiros produzidos pelas interações próton-próton e auxiliar nos cálculos de momento transverso. Sua eletrônica inclui circuitos de *front-end* e digitalizadores de sinais, que são projetados de acordo com as características de alta velocidade e baixo ruido das suas células fotomultiplicadoras. Por fim, os sinais digitalizados correspondentes aos eventos no calorímetro são transferidos para *buffers* através de fibras óticas [44].

### 2.2.2 Sistema de *Trigger*

As interações nos sub-detectores do ATLAS geram um enorme fluxo de dados: são cerca de 60 TB/s (terabytes por segundo) de informação, vindas da colisão entre dois feixes de prótons com 2.808 grupos de  $10^{11}$  partículas cada um, viajando em direções opostas, a uma velocidade de 99.9998% da velocidade da luz, colidindo a uma taxa constante de 40 MHz, de forma que a cada 25 ns ocorre um evento de colisão [48].

A alta taxa de eventos, em conjunto com os cerca de 10.000 canais de leitura do TileCal, fazem com que seja impossível realizar o armazenamento e processamento de todos estes dados. Assim, um sistema de filtragem (*trigger*) na aquisição dos dados é necessário, a fim de selecionar quais eventos podem ser armazenados e descartar os dados irrelevantes para os eventos físicos de interesse.

Geralmente, nos ambientes que operam em altas taxas de eventos, existem restrições temporais e, eventualmente, os eventos gerados podem exigir uma elevada quantidade de memória, necessitando assim de uma discriminação *online* de alta velocidade, tornando o processo de filtragem ainda mais complexo.

Essa filtragem é realizada pelo sistema de seleção de eventos e aquisição de dados do ATLAS, que é esquematizado na Figura 9. Este sistema de *trigger* é composto por três níveis conectados em cascata, que operam *online* e, em cada um deles, o critério de seleção é refinado [49]. Tal sistema é importante pois, a partir de uma taxa de *Bunch Crossing* (BC) inicial de 40 MHz (momento em que as partículas colidem), a taxa de eventos selecionados deve ser reduzida para até 200 Hz, visando o armazenamento para posterior análise [50].

O primeiro nível de *trigger* [51], também chamado de L1 (*Level 1*), é subdividido em *trigger* de calorimetria e *trigger* de mísulas, realizando a filtragem das informações coletadas dos sistemas de calorimetria e das câmaras de mísulas. Para executar suas rotinas de filtragem, ele identifica as assinaturas básicas da física de interesse, baseando sua decisão na multiplicidade de objetos encontrados, que podem ser objetos locais (mísulas, elétrons e jatos) ou globais (energia faltante e energia total). Devido a alta velocidade requerida de processamento, este nível é implementado em *hardware*, e o mesmo reduz a taxa de eventos de entrada de 40 MHz para até 100 kHz. Este nível é o foco do presente trabalho.

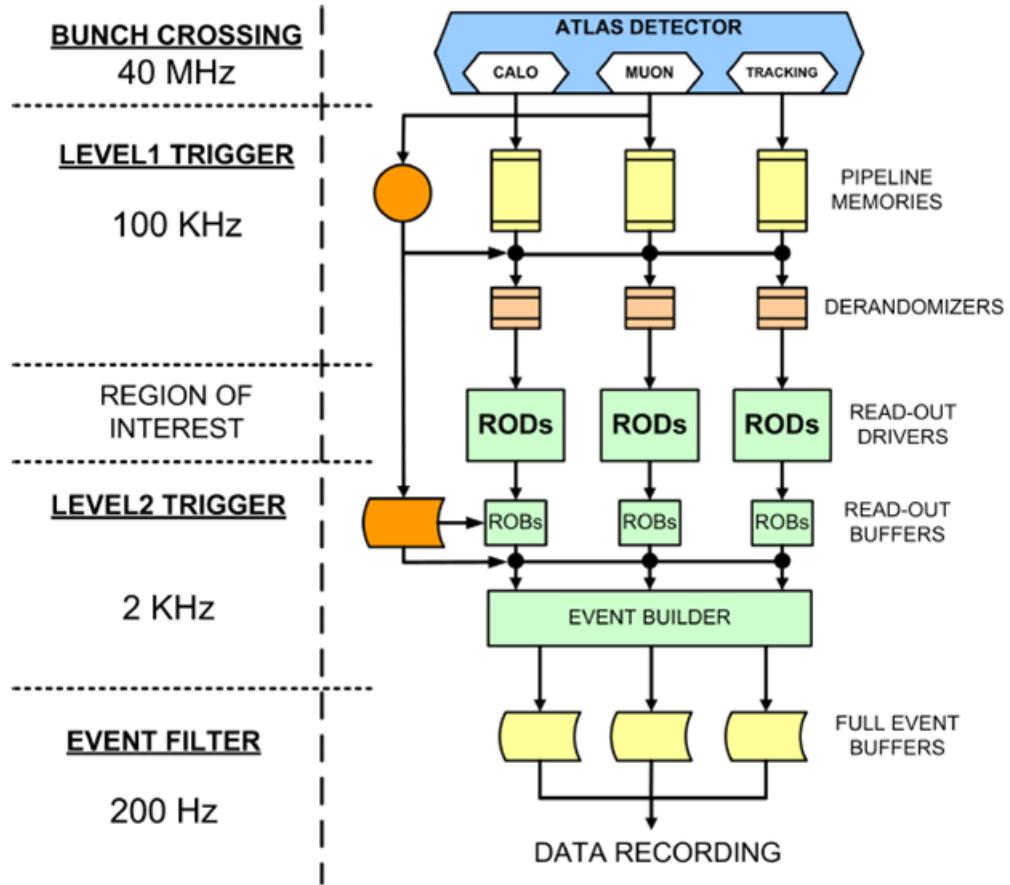


Figura 9 – Sistema de *Trigger* do ATLAS [52].

O segundo nível de filtragem [53], chamado de L2 (*Level 2*), é implementado por *software*, sendo responsável por refinar a filtragem realizada pelo L1, reduzindo ainda mais o número de eventos selecionados, de 100 kHz para não mais que 2 kHz. Para tal, ele conta com uma rede de computadores que processa os algoritmos de busca especializados nos diversos sub-detectores do ATLAS. Esta busca visa encontrar elementos que representem os possíveis decaimentos referentes a física de interesse.

O terceiro nível de filtragem [53] (chamado de EF - *Event Filter*) é implementado por processadores interligados por redes rápidas e reduz ainda mais a taxa de eventos. Diferentemente do L2, que analisa partes específicas dos detectores, o terceiro nível utiliza toda a informação disponível.

O segundo e terceiro níveis de filtragem são denominados *Trigger* de Alto nível ou HLT, do inglês *High Level Trigger* [50]. Ao final do processo de filtragem, a taxa de eventos é reduzida para até 200 por segundo (200 Hz).

### 2.3 DESAFIO: O EFEITO *PILE-UP*

Atualizações graduais estão previstas para ocorrer no LHC nos próximos anos, de forma a elevar a densidade de feixes de prótons, fenômeno que no âmbito de Física de Altas Energias é conhecido como o aumento de luminosidade do acelerador [54, 55]. Com isto, a taxa de interações entre as partículas e a quantidade de partículas elementares detectadas cresce, aumentando assim a probabilidade da ocorrência de fenômenos raros.

Como o tempo de resposta da eletrônica de leitura nos calorímetros é maior do que o período entre as colisões, o aumento na taxa de eventos intervirá principalmente na sobreposição entre sinais provenientes de eventos subsequentes, ocasionando a deformação do sinal recebido e dificultando a equalização do canal. Tal efeito é conhecido como empilhamento de sinais ou *pile-up* [10, 56], o qual é ilustrado na Figura 10.

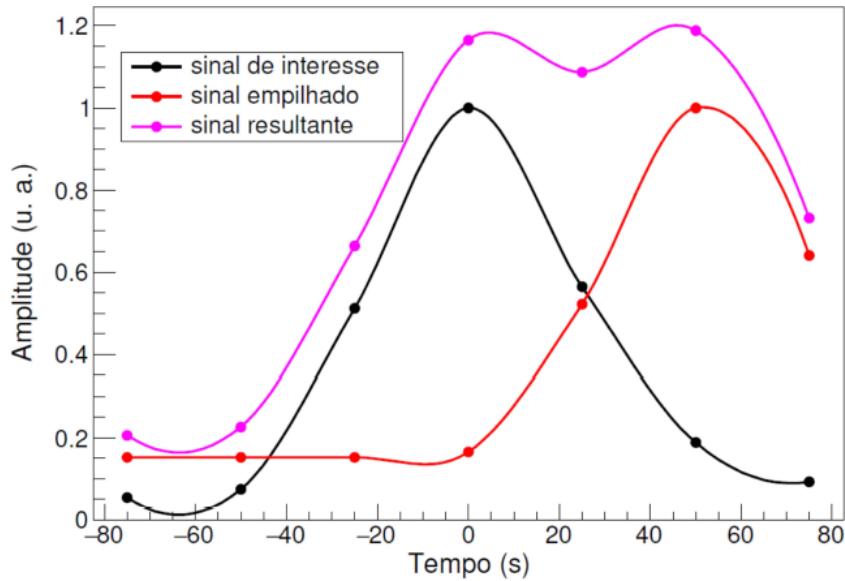


Figura 10 – Efeito *pile-up* no TileCal [10].

A Figura 10 exemplifica o *pile-up*, onde a princípio foi gerado o sinal de interesse (cor preta), devido a uma primeira colisão que sensibilizou uma determinada célula do TileCal e, 50 ns depois, ocorre uma segunda colisão e a mesma célula foi novamente sensibilizada, gerando um segundo sinal (cor vermelha). Como se tratam de eventos muito próximos e inferiores ao período de 150 ns (o necessário para a identificação de um pulso conformado), forma-se um terceiro sinal resultante (cor magenta), que é o efeito do empilhamento entre os outros dois sinais [10].

Os dados utilizados nas simulações e testes realizados neste trabalho são gerados por um *Toy Monte Carlo* [57], que já foi empregado em [58, 59, 60, 61]. Com isso, é possível simular sinais equivalentes aos gerados nos calorímetros do ATLAS, com a vantagem de possuir maior controle nos parâmetros relacionados ao processo de empilhamento de sinais.

Dentre estes parâmetros, um importante a ser ressaltado é o valor de ocupação, que representa a porcentagem da relação entre o número de *bunches* em que houve deposição de energia, em uma determinada célula do calorímetro, e o número de *bunches* total em uma janela de aquisição. Ou seja, uma ocupação de 50%, por exemplo, indica que um *bunch* a cada dois sofreu deposição de energia, em média.

Na Figura 11 é possível ver um gráfico com as características de empilhamento, cujos sinais foram gerados através de simulações com *Toy Monte Carlo* [57], realizadas no *software* MATLAB [62]. A amplitude do sinal em azul representa a energia ideal que desejamos mensurar no canal de leitura do TileCal, sem ruídos. Tal sinal de energia acaba sendo deformando devido ao efeito *pile-up*, como pode ser visto na cor verde.

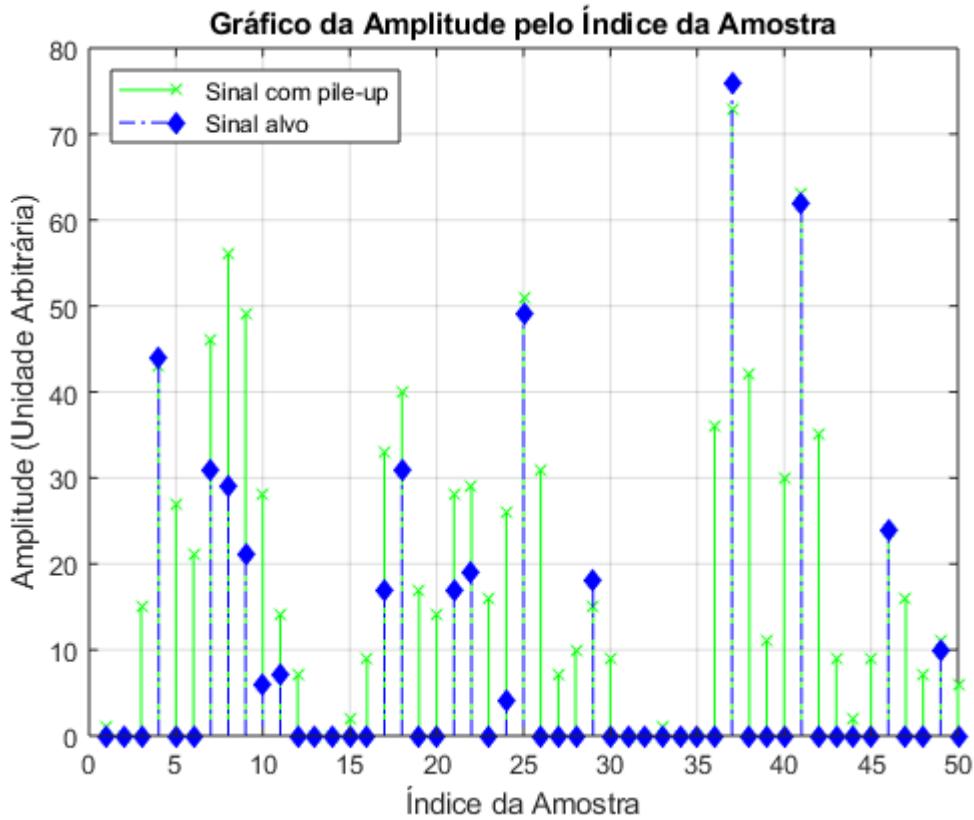


Figura 11 – Simulação de um sinal ideal e um sinal com efeito *pile-up*.

A sobreposição de sinais é um problema recorrente quando os dados são enviados através de um canal de informações à uma taxa superior ao tempo de resposta desse canal, dificultando a seleção de eventos de interesse, feita pelo sistema de *trigger* [49].

Neste contexto, o desafio do presente trabalho é realizar a implementação em FPGA de um processamento dedicado, visando a reconstrução *online* dos sinais no primeiro nível de *Trigger* do TileCal, através de um método sensível ao efeito *pile-up*.

### 3 REVISÃO BIBLIOGRÁFICA

O método atualmente utilizado para a reconstrução dos sinais no primeiro nível de *trigger* do TileCal é o Filtro Casado [47]. Esse tipo de filtro possui coeficientes proporcionais aos pesos de cada amostra do pulso característico do sinal, para sinais determinísticos, imersos em ruído branco gaussiano [63]. Tal técnica é baseada em correlacionar o pulso característico com o sinal do próprio pulso de interesse, uma vez que o efeito produzido pelo Filtro Casado é a maximização da relação sinal ruído [51].

Dessa forma, o desempenho do Filtro Casado depende do prévio conhecimento da forma do pulso em meio ao ruído. Tal método não é indicado para a operação em ambientes de alta luminosidade, uma vez que o efeito *pile-up* modifica a forma do pulso característico do sinal. Nesse contexto, novas técnicas de estimativa de energia e detecção de sinais precisam ser investigadas.

Propostas recentes para a detecção de sinais se baseiam na modelagem da cadeia eletrônica do TileCal como um canal de comunicação, de forma que a estimativa de energia é obtida pela deconvolução (ou equalização) desse canal [10].

No trabalho [13], é mostrado que técnicas de deconvolução baseadas em Filtros FIR (*Finite Impulse Response*) [14] possuem simples implementação, além de produzirem uma melhor estimativa em relação ao Filtro Casado para condições de alta taxa de eventos, apresentando bons resultados e baixo custo computacional. Esta metodologia usa a minimização do erro médio quadrático (RMS, do inglês *Root Mean Square*) entre os dados e o modelo para a determinação dos coeficientes do filtro [64]. Porém, mesmo com melhor desempenho nessas condições, a desvantagem de tal abordagem é que outros ruídos existentes em uma aplicação real não são modelados pelos Filtros FIR, tais como os ruídos aditivos de fundo e os de desvio de fase.

Outra abordagem, através de métodos baseados em um modelo matricial de sobreposição de sinais, determina a amplitude dos diversos sinais empilhados dentro de uma janela de aquisição, por meio de algoritmos iterativos, seguindo um critério pela busca da esparsidade dos dados reconstruídos [65]. Tal abordagem apresenta desempenho superior ao método baseado em Filtros FIR no que diz respeito à acurácia da reconstrução da informação [60], porém o custo computacional é alto, representando um desafio na proposição de algoritmos a serem implementados em ambiente embarcado.

Nesse contexto, teorias de SR (*Sparse Representation*), ou Representação Esparsa de dados, buscam soluções mais eficientes [66], em termos de implementação. O método SSF (*Separable Surrogate Functionals*) [16] implementa uma forma iterativa na busca pela SR, utilizando apenas operações de soma e produto, podendo, com isso, ser implementado de forma eficiente em FPGAs modernas.

### 3.1 REPRESENTAÇÃO ESPARSA PARA ESTIMAÇÃO DE ENERGIA

O trabalho [59] propõe um modelo onde, dado um pulso de referência normalizado<sup>1</sup> do calorímetro, representado pelo vetor  $\mathbf{h}$ , e uma sequência  $\mathbf{x}$ , que representa os valores de energia a serem reconstruídos, a convolução entre estes dois sinais é o sinal de leitura  $\mathbf{r}$  amostrado na eletrônica de *front-end* do calorímetro, cujo *clock* é síncrono com a taxa de colisões do acelerador.

Para a matriz de convolução  $\mathbf{H}$ , cujas colunas contêm versões deslocadas do sinal de referência normalizado  $\mathbf{h}$ , uma formulação matricial para o processo de convolução é:

$$\mathbf{r} = \mathbf{H}\mathbf{x} \quad (3.1)$$

O processo de deconvolução consiste em reconstruir a sequência  $\mathbf{x}$  quando  $\mathbf{r}$  e  $\mathbf{H}$  são conhecidos, o que é o caso no problema em questão. Como o tamanho do vetor  $\mathbf{r}$  é maior do que o do vetor  $\mathbf{x}$ , existem infinitas soluções para este sistema de equações. O trabalho [58] mostra que, para a deconvolução de sinais impulsivos, a representação mais esparsa de  $\mathbf{x}$  é a melhor escolha.

Em [66] é demonstrado que a solução SR da Equação 3.1 é obtida, resolvendo-se, para  $0 \leq \ell \leq 1$ , o problema:

$$(P_\ell) : \min_{\mathbf{x}} \|\mathbf{x}\|_\ell^\ell \quad \text{sujeto a} \quad \mathbf{r} = \mathbf{H}\mathbf{x} \quad (3.2)$$

Onde a norma- $\ell$  do vetor  $\mathbf{x}$  é dada por:

$$\|\mathbf{x}\|_\ell^\ell = \sum_i |x_i|^\ell \quad (3.3)$$

Fazendo  $\ell = 1$ , o Problema  $P_1$  resulta em um problema típico de Programação Linear ou LP, do inglês *Linear Programming* [67]. Nesse contexto, no trabalho [58] foi proposto o uso de LP em SR, tendo como foco a reconstrução de energia, onde o desempenho de tal método se mostrou superior a outros métodos janelados de deconvolução. Porém, destaca-se que o foco daquele trabalho foi a reconstrução *offline* de energia, uma vez que LP apresenta um custo computacional muito elevado.

Nos trabalhos [65] e [68], métodos modernos de SR, usando  $P_1$ , foram analisados e uma implementação em FPGA de uma versão adaptada do método conhecido como SSF mostrou-se bem sucedida. Esse método é baseado no Problema  $P_1$ , empregando-se uma relaxação na restrição, de modo a permitir um erro quadrático pequeno  $\epsilon_0$  para englobar problemas com adição de ruído.

---

<sup>1</sup> Pulso de referência com amplitude unitária.

Dessa forma, temos:

$$(P_{1,\epsilon_0}) : \min_{\mathbf{x}} \|\mathbf{x}\|_1^1 \quad \text{sujeito a} \quad |\mathbf{r} - \mathbf{Hx}|^2 < \epsilon_0 \quad (3.4)$$

O Problema  $(P_{1,\epsilon_0})$  pode ser transformado em um problema de otimização sem restrição, usando um multiplicador de Lagrange  $\lambda$  adequado, onde  $\epsilon_0$  é absorvido por  $\lambda$ :

$$(P_{1,\lambda}) : \min_{\mathbf{x}} \|\mathbf{x}\|_1^1 + \lambda |\mathbf{r} - \mathbf{Hx}|^2 \quad (3.5)$$

Na literatura existem diversos algoritmos que propõem uma solução para  $(P_{1,\lambda})$ , por meio de métodos iterativos de CD (Coordenadas Descendentes) [66]: dado um vetor inicial  $\mathbf{x}_0$ , o valor ideal  $\mathbf{x}_{opt}$  pode ser inferido recursivamente em pequenos passos. No entanto, o uso de métodos de CD padrão, como o método Newton-Raphson [69], torna-se proibitivo para  $(P_{1,\lambda})$ , devido a descontinuidade da norma  $\ell_1$ .

Em [16] é proposta a inserção de termos que não alteram a posição das coordenadas  $\mathbf{x}_{opt}$  em  $(P_{1,\lambda})$ , mas que são capazes de dividir o problema multivariado em problemas unidimensionais separados, que podem ser resolvidos por partes. A equação resultante é denominada uma função substituta e o respectivo método é o SSF.

Após manipulações algébricas e definindo o tamanho do passo em direção ao mínimo como  $\mu$ , o procedimento iterativo pode ser compactado como:

$$\mathbf{x}_{i+1} = S_\lambda \left( \mathbf{x}_i + \mu [\mathbf{H}^T (\mathbf{r} - \mathbf{Hx}_i)] \right) \quad (3.6)$$

É proposta, em [65], uma modificação na função de *Shrinkage*<sup>2</sup>,  $S_\lambda(\theta)$ , para permitir apenas a reconstrução positiva da energia, que é o caso na calorimetria (Figura 12).

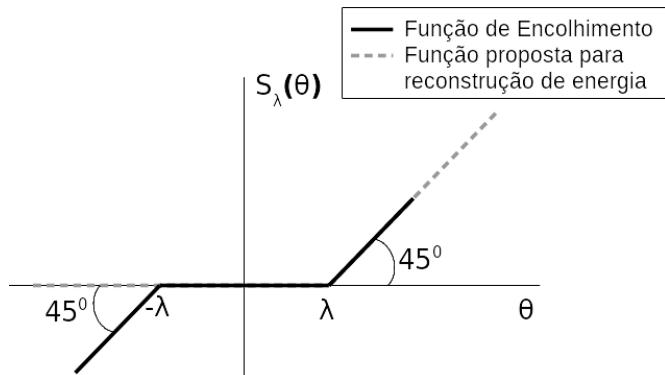


Figura 12 – Função proposta para reconstrução de energia [65].

---

<sup>2</sup> A função de *Shrinkage* é a função de encolhimento unidimensional, que deve ser aplicada a cada componente no argumento separadamente.

Com isso, a função resultante pode ser implementada simplesmente por meio de uma subtração seguida por uma operação de limiar. O argumento na Equação 3.6 é identificado como uma iteração do método GD (Gradiente Descendente) linear e comprehende apenas operações de soma e multiplicação. Os parâmetros  $\lambda$  e  $\mu$  da Equação 3.6 tiveram suas calibrações realizadas com dados de simulação do TileCal, sendo fixados em 0 e 0.25, respectivamente [65]. A vantagem de fixar tais parâmetros é que circuitos adicionais de multiplicação e de comparação no *hardware* desenvolvido não serão necessários.

### 3.1.1 Aprimoramento na Inicialização do Algoritmo

Para que o processo iterativo do método definido pela Equação 3.6 possa ser realizado, o vetor  $\mathbf{x}$  é inicializado com uma janela contendo os dados vindos do conversor analógico-digital do primeiro nível de *trigger* do TileCal. Porém, foi realizado um estudo no decorrer do presente trabalho, publicado em [70], que avalia uma forma alternativa para a inicialização do vetor  $\mathbf{x}$ , através de um pré-processamento a ser realizado, na respectiva janela de dados, antes da etapa iterativa do algoritmo.

Tal procedimento se baseia em, respeitando a restrição de manter o modelo já proposto, deve-se obter uma solução aproximada de  $\mathbf{x}$  através da minimização da sua norma quadrática euclidiana [66], uma vez que a matriz de convolução  $\mathbf{H}$  da Equação 3.1 não possui inversa. Assim, esse problema é definido por:

$$(P_2) : \min_{\mathbf{x}} \|\mathbf{x}\|_2^2 \quad \text{sujeto a} \quad \mathbf{r} = \mathbf{Hx} \quad (3.7)$$

Utilizando multiplicadores de Lagrange  $\lambda$  para o conjunto de restrições do Problema  $(P_2)$ , o Lagrangiano [71] pode então ser definido como:

$$\mathcal{L}(\mathbf{x}) = \|\mathbf{x}\|_2^2 + \lambda^T(\mathbf{Hx} - \mathbf{r}) \quad (3.8)$$

Realizando a derivada parcial de  $\mathcal{L}(\mathbf{x})$  em relação a  $\mathbf{x}$ , obtém-se:

$$\frac{\partial \mathcal{L}(\mathbf{x})}{\partial \mathbf{x}} = 2\mathbf{x} + \mathbf{H}^T\lambda \quad (3.9)$$

Cuja solução, obtida para  $\mathbf{x}$ , após realizadas manipulações algébricas, é:

$$\begin{aligned} \hat{\mathbf{x}}_{opt} &= -\frac{1}{2}\mathbf{H}^T\lambda \rightarrow \mathbf{H}\hat{\mathbf{x}}_{opt} = -\frac{1}{2}\mathbf{H}\mathbf{H}^T\lambda = \mathbf{r} \rightarrow \lambda = -2(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{r} \\ \hat{\mathbf{x}}_{opt} &= -\frac{1}{2}\mathbf{H}^T\lambda \rightarrow \hat{\mathbf{x}}_{opt} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{r} \rightarrow \hat{\mathbf{x}}_{opt} = \mathbf{H}^+\mathbf{r} \end{aligned} \quad (3.10)$$

Tal demonstração está detalhada em [66]. Nesta solução, a matriz  $\mathbf{H}^+$  é definida como matriz pseudo-inversa de  $\mathbf{H}$  e a proposta é realizar a inicialização do vetor  $\mathbf{x}$  como  $\mathbf{H}^+\mathbf{r}$ , uma vez que este pré-processamento garante que o vetor  $\mathbf{x}$  seja inicializado mais próximo da solução e, assim, a convergência do método SSF ocorre mais rapidamente [70].

## 4 IMPLEMENTAÇÃO

Nesta seção serão apresentados os passos seguidos para a implementação do presente trabalho. Para tal, a plataforma de desenvolvimento utilizada foi a FPGA DE2115 da família Cyclone IV E da Altera<sup>1</sup>. Dentre os principais recursos desse *kit*, destacam-se os 3,9 Mbits de RAM (*Random Access Memory*) e 114.480 elementos lógicos disponíveis. As ferramentas utilizadas para as simulações de operação do processador embarcado foram o Modelsim-Altera [72] e o Quartus da Intel<sup>2</sup>.

Primeiro será detalhado o desenvolvimento do processador dedicado e a principal ferramenta utilizada. Além disso, serão mostradas algumas modificações e aprimoramentos realizados na estrutura do processador, visando otimizar este tipo de implementação. Após isso, será apresentada cada implementação realizada dos algoritmos para chegar na versão otimizada do método SSF. Por fim, será detalhada a arquitetura proposta para um processamento que seja factível, respeitando a taxa de colisões do LHC.

### 4.1 SAPHO: UM PROCESSADOR AUTO-ESCALÁVEL

Para permitir uma implementação viável em *hardware* de algoritmos complexos, grande parte dos sistemas embarcados atuais utilizam Processadores *Soft-Core* (PSCs) [73]. Dentre os diversos PSCs disponíveis, tanto nos comerciais quanto nos de código aberto [74], uma característica comum a eles é a sua arquitetura fixa, de forma que, independente da complexidade do programa embarcado, a mesma quantidade de recursos em *hardware* é alocada e, além disso, o tamanho da palavra de dados é fixada (geralmente em 32 bits), sendo, na maioria das vezes, superdimensionada.

Para a implementação em FPGA do método SSF, foi utilizado o SAPHO<sup>3</sup> (*Scalable-Architecture Processor for Hardware Optimization*), um PSC parametrizável e de código aberto desenvolvido no Núcleo de Instrumentação e Processamento de Sinais da Universidade Federal de Juiz de Fora (NIPS/UFJF), capaz de realizar operações por meio de circuitos aritméticos em ponto fixo e em ponto flutuante [75].

Diferentemente dos diversos PSCs disponíveis, o SAPHO, cujo diagrama de sua estrutura é apresentado na Figura 13, não possui uma microarquitetura fixa. Nele, os recursos são alocados automaticamente, durante a compilação do programa embarcado. Assim, apenas os elementos lógicos necessários são utilizados, reduzindo o custo computacional do projeto. O SAPHO já foi utilizado em diversos sistemas já consolidados, como nos trabalhos [76, 77, 78, 79, 80], onde é possível obter detalhes mais aprofundados a respeito de sua estrutura e funcionamento.

<sup>1</sup> Mais detalhes em: <https://www.macnicadhw.com.br/produtos/kits/altera-de2115>

<sup>2</sup> Disponível em: <https://fpgasoftware.intel.com/>

<sup>3</sup> Código fonte disponível em: <https://github.com/nipscernufjf/SAPHO>

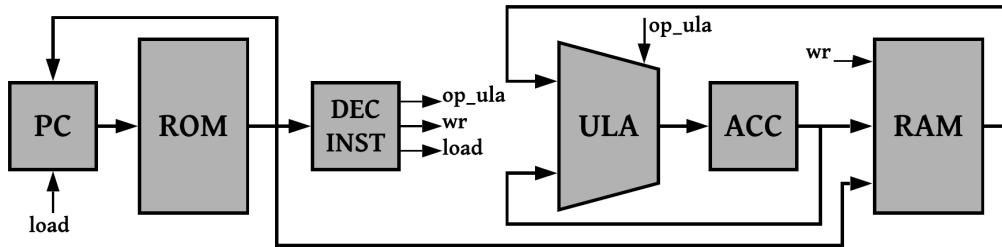


Figura 13 – Diagrama do SAPHO [70].

O ambiente de desenvolvimento integrado do SAPHO, também chamado de IDE (*Integrated Development Environment*), foi desenvolvido na linguagem **C#**, com o intuito de executar os compiladores **C** e **Assembler**<sup>4</sup> de forma transparente para o usuário. Ambos os compiladores foram desenvolvidos utilizando as ferramentas *flex* e *bison* da GNU [81].

O compilador **C** tem a função de gerar um arquivo em linguagem *Assembly* [82] ao interpretar o código do usuário, que é escrito na interface do SAPHO através de um subconjunto da linguagem **C**, também desenvolvido no NIPS/UFJF, batizado de **C<sup>+</sup>**. Esta linguagem também apresenta recursos não encontrados na linguagem C padrão, com o objetivo de aproveitar melhor as otimizações feitas em hardware.

O compilador **Assembler** tem o objetivo de interpretar o arquivo em *Assembly*, que foi previamente gerado pelo compilador **C**, fazendo o reconhecimento do *opcode* (código de operação) e dos operandos de cada uma das instruções, sendo responsável por gerar o arquivo de parametrização do processador em linguagem *Verilog* [83] com a descrição do *hardware* desenvolvido, podendo ser sintetizado por qualquer FPGA. O **Assembler** reconhece e informa a quantidade final de instruções do programa e o número de variáveis necessárias para sua execução, criando então as memórias de programa e de dados com o tamanho correto, além de alocar, em *hardware*, somente os recursos necessários, que são ajustados automaticamente em tempo de compilação do programa embarcado.

As memórias de programa e de dados possuem números de endereços auto-escaláveis e, além disso, a geração dos circuitos internos da Unidade Lógico-Aritmética (ULA) é automática. Na prática, a IDE do SAPHO possui ferramentas para auxiliar na parametrização e desenvolvimento com processadores, permitindo que parâmetros como o tipo de ULA (para operar com aritmética de ponto fixo ou ponto flutuante), o tamanho da palavra de dados e o número de endereços de entrada e saída possam ser facilmente ajustados pelo programador. Um tutorial para o desenvolvimento de processadores utilizando a versão mais recente do SAPHO, a qual inclui todas as otimizações na sua estrutura, a serem apresentadas nessa seção, é apresentado no Anexo C.

<sup>4</sup> Normalmente, Assembler se refere a um montador para linguagem de máquina. Porém, no contexto do SAPHO, refere-se a um compilador da linguagem Assembly para Verilog.

Na Figura 14 é possível ver a IDE do SAPHO, onde foi projetado um processador em ponto fixo de 16 bits, com duas portas de entrada e duas de saída.

No código, em linguagem  $C^+$ , a variável **a** foi declarada com valor -2 e é verificado se ela é negativa. Se **a** for negativa, a variável **b** recebe zero, caso contrário, **b** recebe o valor de **a**. Por fim, o valor de **b** é encaminhado à porta de saída de índice 0. Após esse código ser compilado, é gerado um código em linguagem de máquina *Assembly* (Figura 15), a ser compilado pelo **Assembler** para gerar a parametrização do *hardware*.

```

Sapho - IDE
File Edit Help
Hierarchy: teste
  teste
    teste.c
    teste.asm
  teste_pset
  teste_nom
teste.c
1 #PRNAME teste
2 #DIRNAME "C:\Users\melis\Desktop\teste\teste\Hardware\teste_H"
3 #DATATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10
11 void main()
12 {
13     int a = -2;
14     int b;
15
16     if(a<0)
17         b = 0;
18     else b = a;
19
20     out(0, b);
21 }
22

```

Console Output:

Errors:

[0] - ##### Total de instrucoes: 14  
[0] - ##### Total de variaveis : 8

Figura 14 – Código em  $C^+$  na IDE do SAPHO.

```

Sapho - IDE
File Edit Help
Hierarchy: teste
  teste
    teste.c
    teste.asm
  teste_pset
  teste_nom
teste.c teste.asm
1 #PRNAME teste
2 #DIRNAME "C:\Users\melis\Desktop\teste\teste\Hardware\teste_H"
3 #DATATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10 @main LOAD -2
11 SET maina
12 LOAD 0
13 LBS maina
14 JZ Lielse
15 LOAD 0
16 SET mainb
17 JMP Llend
18 @Lielse LOAD maina
19 SET mainb
20 @Llend LOAD 0
21 PLD mainb
22 OUT
23 @fim JMP fim
24

```

Console Output:

[0] - ##### Total de instrucoes: 14  
[0] - ##### Total de variaveis : 8

Figura 15 – Código em *Assembly* na IDE do SAPHO.

#### 4.1.1 Otimizações na Estrutura do Processador

A operação descrita nas Figuras 14 e 15 é de grande importância na implementação do método SSF proposto, uma vez que os valores obtidos na reconstrução de energia devem ser positivos [65] e, por isso, os termos menores que zero precisam ser anulados.

Nesse contexto, foi realizada uma modificação na estrutura do processador, através da inclusão de uma nova instrução em um novo bloco de processamento (PSET), de forma a permitir a sua realização diretamente em *hardware* em um único ciclo de *clock*. Tal modificação é destacada na Figura 16.

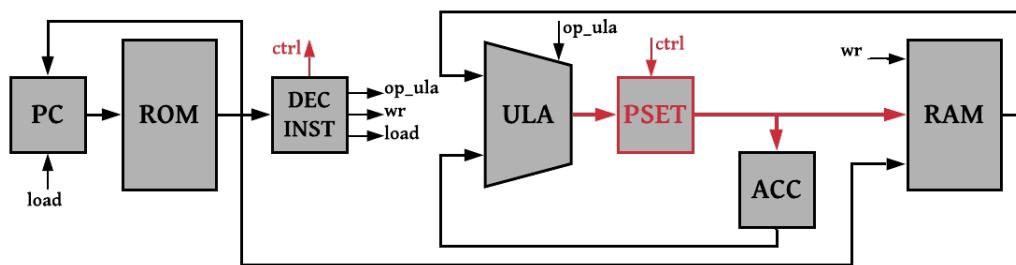


Figura 16 – Diagrama do SAPHO com o aprimoramento.

Na Figura 17 é possível ver a sintaxe necessária para utilizar a nova função, que foi definida pelo símbolo @, simplificando a sintaxe já conhecida das funções *if* e *else*.

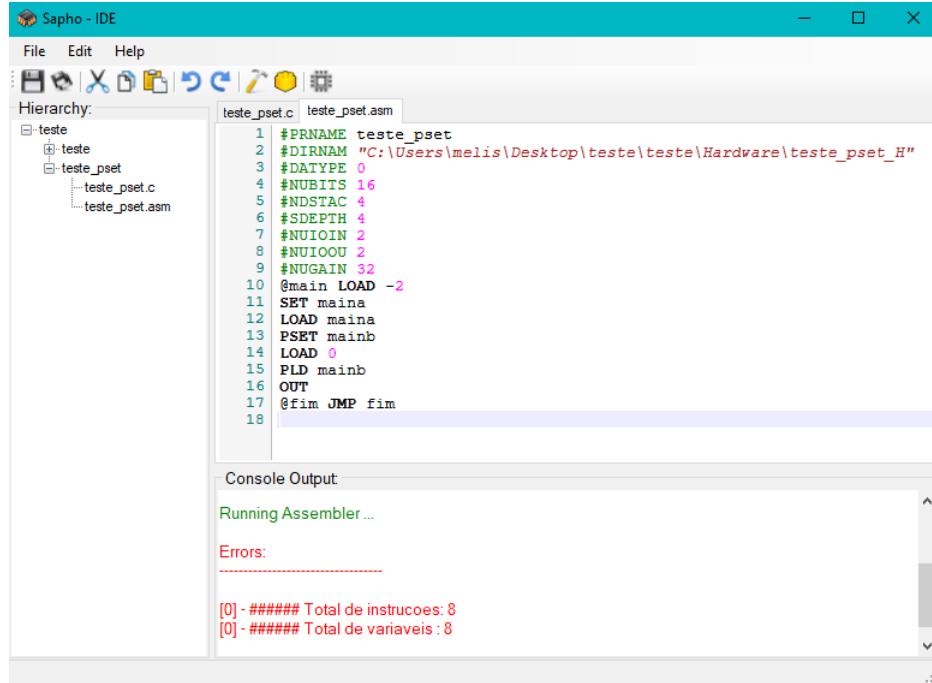
```

#include <stdio.h>
#include "teste_pset.h"

void main()
{
    int a = -2;
    int b;
    b @ a;
    out(0, b);
}
  
```

Figura 17 – Código em  $\mathbf{C}^+$  com o PSET.

Já na Figura 18, é possível ver que as instruções antes necessárias para realizar a operação agora se resumem apenas na nova instrução PSET, simplificando então o código em *Assembly* gerado pelo compilador C.



The screenshot shows the Sapho IDE interface. The top menu bar includes File, Edit, Help, and various tool icons. The left sidebar displays a project hierarchy under the 'teste' folder, containing 'teste', 'teste\_pset', 'teste\_pset.c', and 'teste\_pset.asm'. The main workspace shows the assembly file 'teste\_pset.asm' with the following code:

```

1 #PRNAME teste_pset
2 #DIRNAM "C:\Users\melis\Desktop\teste\teste\Hardware\teste_pset_H"
3 #DATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10 @main LOAD -2
11 SET maina
12 LOAD maina
13 PSET mainb
14 LOAD 0
15 PLD mainb
16 OUT
17 @fim JMP fim
18

```

Below the code editor is a 'Console Output' panel showing the message 'Running Assembler ...'. The 'Errors:' section is empty. At the bottom, status messages are displayed in red:

- [0] - ##### Total de instrucoes: 8
- [0] - ##### Total de variaveis : 8

Figura 18 – Código em *Assembly* com o PSET.

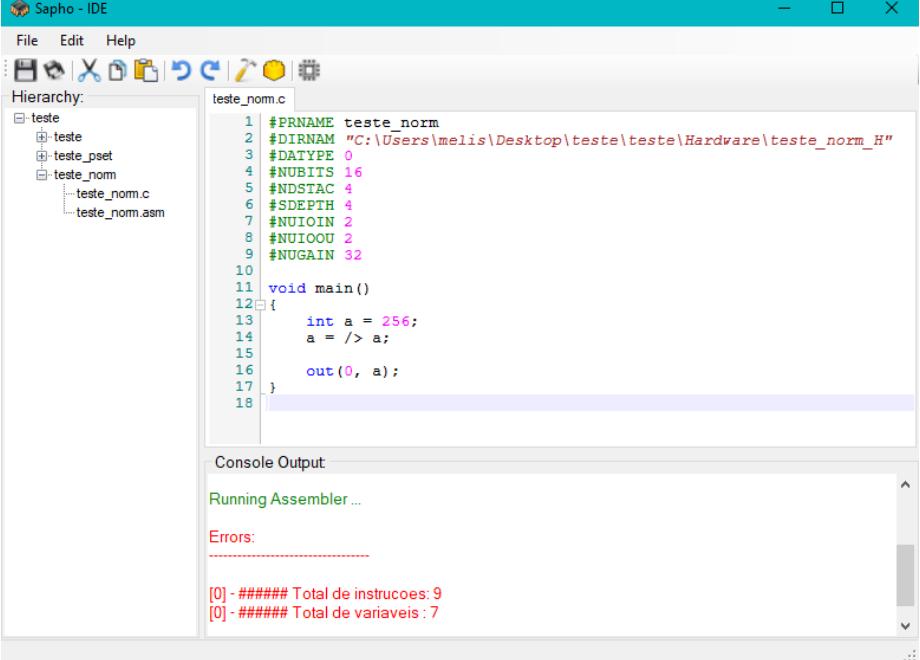
Uma outra operação de grande importância para o método SSF, na etapa de implementação em ponto fixo, é a de normalização. Essa operação é realizada através de uma divisão da variável que se deseja normalizar por um valor de ganho.

Uma vez observado que tal operação se repete no *loop* do algoritmo e que, além disso, o valor do ganho é constante, foi definida, na ULA, a nova instrução NORM. Esta instrução normaliza o número a direita do operador, de forma que os circuitos de divisão necessários em *hardware* são drasticamente simplificados, uma vez que a ULA não tem mais a necessidade de verificar o valor do dividendo nas operações de divisão pelo ganho.

Para isso, foi incluído o novo parâmetro NUGAIN, onde é definido o valor constante de ganho a ser utilizado para as operações de normalização. A sintaxe em C<sub>+</sub> dessa modificação é definida pelo operador / >, a qual pode ser vista na Figura 19.

Tal operador gera, em *Assembly*, a instrução NORM, que pode ser vista na Figura 20.

Após realizadas as modificações na estrutura do SAPHO, foi possível implementar o método SSF a partir de diversos algoritmos, para processamentos realizados tanto em ponto flutuante quanto em ponto fixo, com o objetivo de comparar e definir qual a melhor implementação em relação a frequência operacional, tempo de atraso e consumo lógico.



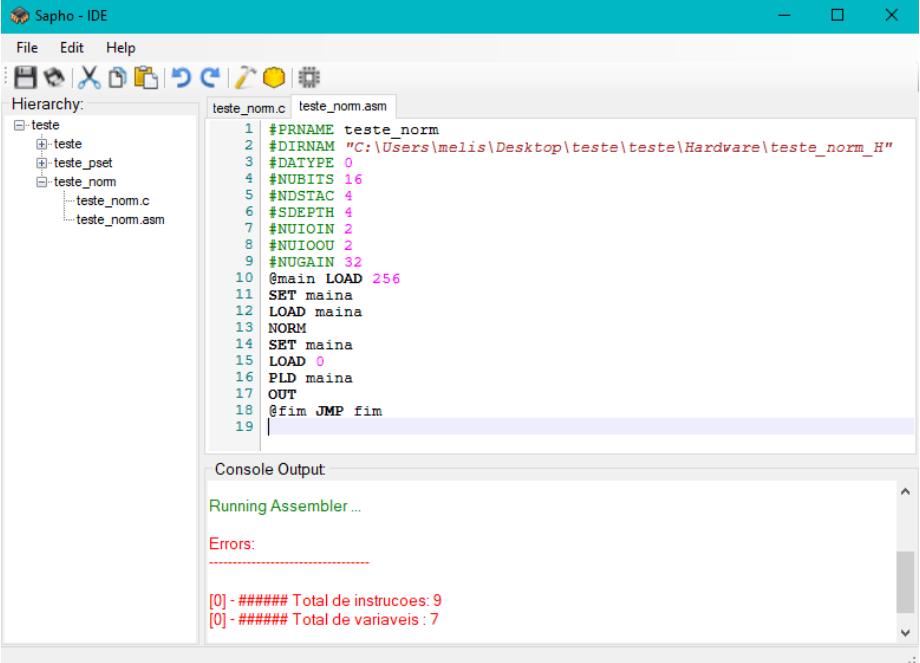
The screenshot shows the Sapho - IDE interface. The top menu bar includes File, Edit, Help, and several icons. The left sidebar displays a project hierarchy under the 'teste' folder, including 'teste', 'teste\_pset', 'teste\_norm', 'teste\_norm.c', and 'teste\_norm.asm'. The main editor window is titled 'teste\_norm.c' and contains the following C code:

```

1 #PRNAME teste_norm
2 #DIRNAM "C:\Users\melis\Desktop\teste\teste\Hardware\teste_norm_H"
3 #DATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10 void main()
11 {
12     int a = 256;
13     a = /> a;
14
15     out(0, a);
16 }
17
18

```

Below the code editor is a 'Console Output' panel. It shows the message 'Running Assembler ...' followed by 'Errors:' and two diagnostic messages in red: '[0] - ##### Total de instrucoes: 9' and '[0] - ##### Total de variaveis: 7'.

Figura 19 – Código em  $C^+$  com o NORM.


The screenshot shows the Sapho - IDE interface, similar to Figure 19. The project hierarchy is the same. The main editor window is titled 'teste\_norm.asm' and contains the following assembly code:

```

1 #PRNAME teste_norm
2 #DIRNAM "C:\Users\melis\Desktop\teste\teste\Hardware\teste_norm_H"
3 #DATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10 @main LOAD 256
11 SET maina
12 LOAD maina
13 NORM
14 SET maina
15 LOAD 0
16 PLD maina
17 OUT
18 @fim JMP fim
19

```

Below the code editor is a 'Console Output' panel. It shows the message 'Running Assembler ...' followed by 'Errors:' and two diagnostic messages in red: '[0] - ##### Total de instrucoes: 9' and '[0] - ##### Total de variaveis: 7'.

Figura 20 – Código em *Assembly* com o NORM.

## 4.2 MÉTODO OPERANDO EM PONTO FLUTUANTE

Dado o embasamento matemático do método SSF, descrito na Seção 3, a implementação do algoritmo para se obter o sinal representando a estimativa de energia pode ser resumida na resolução da Equação 4.1, realizando 150 iterações [65].

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu \mathbf{H}^T (\mathbf{r} - \mathbf{Hx}_i) \quad (4.1)$$

Vale ressaltar que no SAPHO não é possível escrever diretamente os cálculos matriciais em linguagem  $\mathbf{C}_+^+$ , portanto, as operações matriciais precisam ser escritas termo a termo, demandando uma grande quantidade de linhas de código. Visando simplificar esse trabalho manual, foram desenvolvidas rotinas na linguagem *Python* [84], criadas especialmente para escrever as operações necessárias para realizar os produtos matriciais desse método em linguagem  $\mathbf{C}_+^+$  no SAPHO. Uma outra característica importante a ser ressaltada é que a matriz de convolução  $\mathbf{H}$  possui diversos coeficientes nulos, como pode ser visto na Figura 21, o que facilita a implementação dos produtos matriciais em  $\mathbf{C}_+^+$ .

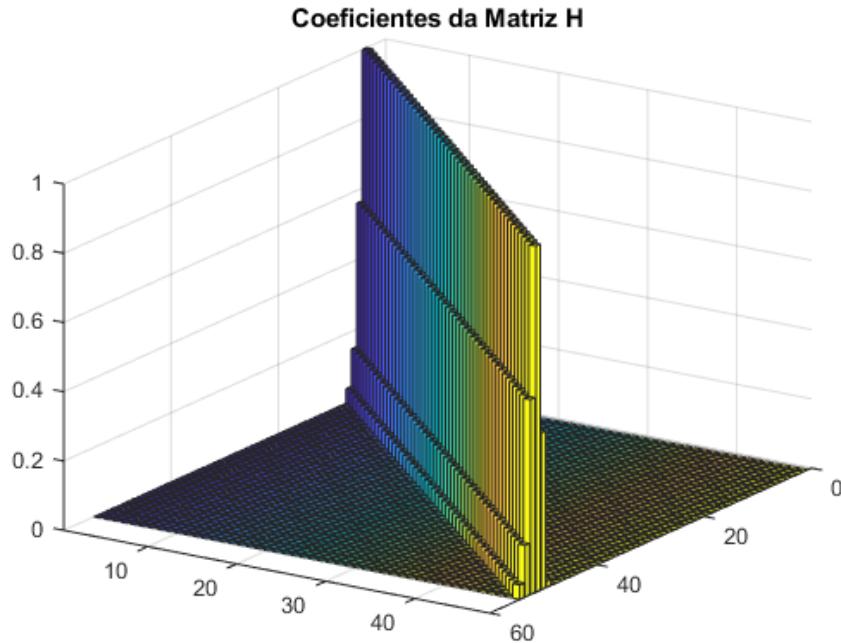
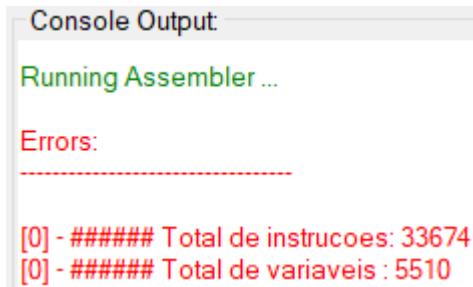


Figura 21 – Coeficientes da matriz de convolução.

Nas implementações que serão apresentadas a seguir, os parâmetros conhecidos são a matriz de convolução  $\mathbf{H}$ , a constante  $\mu$  (fixada por simulações em 0.25 [65]) e os dados de entrada com efeito de empilhamento  $\mathbf{r}$ . Além disso, o sinal sem o efeito de empilhamento (que é o sinal esperado após realizada a reconstrução de energia) também é conhecido, uma vez que para a realização dos testes foi utilizado o *Toy Monte Carlo* de [57].

A primeira implementação (versão 1.0) em ponto flutuante do método SSF foi realizada da forma mais genérica possível, onde cada termo da matriz  $\mathbf{H}$  na Equação 4.1 é declarado como uma variável. O código que foi implementado em  $\mathbf{C}_+^+$  pode ser visto de forma resumida no Algoritmo 1, Anexo A.

Os relatórios da compilação deste algoritmo no SAPHO e no Quartus podem ser vistos, respectivamente, nas Figuras 22 e 23.



The screenshot shows the SAPHO software interface with the title "Console Output". It displays the message "Running Assembler ..." in green. Below it, under the heading "Errors:", there are two red error messages: "[0] - ##### Total de instrucoes: 33674" and "[0] - ##### Total de variaveis : 5510".

Figura 22 – Compilação do Algoritmo 1 no SAPHO (versão 1.0).

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,405
Total combinational functions	1,389
Dedicated logic registers	108
Total registers	108
Total pins	59
Total virtual pins	0
Total memory bits	936,864
Embedded Multiplier 9-bit elements	2
Total PLLs	0

Figura 23 – Compilação do Algoritmo 1 no Quartus (versão 1.0).

Vale ressaltar que no total de instruções informadas no *Console Output* do SAPHO (Figura 22), as instruções só são contabilizadas uma vez cada, mesmo que estejam dentro de algum *loop*. Ao contabilizar as repetições de instruções devido ao *loop* de 150 iterações, o número total de *clocks* necessários para a realização deste algoritmo foi 14.057.029.

Como os dados de entrada ( $r[n]$ ) chegam a cada 25 ns e a janela de entrada para esse método possui 55 amostras consecutivas do conversor ADC, todo o processamento para uma janela precisa ocorrer num período de  $55 \times (25 \text{ ns}) = 1375 \text{ ns}$ , que é o tempo necessário para que a próxima janela de entrada de dados seja formada. Assim, todo o processamento ocorrerá de forma síncrona com a frequência do LHC, como é esperado. Desta forma, para que a versão 1.0 do processador implementado respeite a taxa de colisões do LHC, a frequência necessária ao processador é  $14.057.029 / (1.375 \text{ ns}) = 10,22 \text{ THz}$ .

Tal frequência não é factível para implementação em eletrônica digital [85], o que motivou na busca por implementações mais eficientes para tal processamento [86], [87].

Para a versão 1.1 da implementação do algoritmo, foi realizada a operação distributiva na Equação 4.1 de forma a reduzir os produtos matriciais pela matriz  $\mathbf{H}^T$ , que na implementação anterior estava em evidência no *loop* de 150 iterações. Dessa forma, ao se realizar a manipulação algébrica, a nova equação é:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu \mathbf{H}^T \mathbf{r} - \mu \mathbf{H}^T \mathbf{H} \mathbf{x}_i \quad (4.2)$$

Assim, a parcela  $\mu \mathbf{H}^T \mathbf{r}$  só precisa ser calculada uma vez, reduzindo o número de operações dentro do *loop*, ao comparar com a primeira versão da implementação.

Os relatórios da compilação no SAPHO e no Quartus dessa versão do algoritmo podem ser vistos, respectivamente, nas Figuras 24 e 25.

```

Console Output
Running Assembler ...
Errors:
-----
[0] - ##### Total de instrucoes: 34203
[0] - ##### Total de variaveis : 5557

```

Figura 24 – Compilação do Algoritmo 2 no SAPHO(versão 1.1).

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,407
Total combinational functions	1,391
Dedicated logic registers	108
Total registers	108
Total pins	59
Total virtual pins	0
Total memory bits	955,518
Embedded Multiplier 9-bit elements	2
Total PLLs	0

Figura 25 – Compilação do Algoritmo 2 no Quartus (versão 1.1).

A frequência de operação para esta implementação é 10,05 THz, ou seja, diminuiu em 174 GHz ao ser comparada com a implementação anterior. O Algoritmo 2, no Anexo A, resume o código implementado em  $\mathbf{C}_-^+$  para essa versão da implementação.

Visando reduzir ainda mais o número de instruções, foram realizadas mais manipulações algébricas na equação do método SSF. Como o produto  $\mathbf{H}^T \mathbf{H}$  é constante, este resultado passa a ser definido como a matriz simétrica  $\mathbf{A}$  (ilustrada na Figura 26).

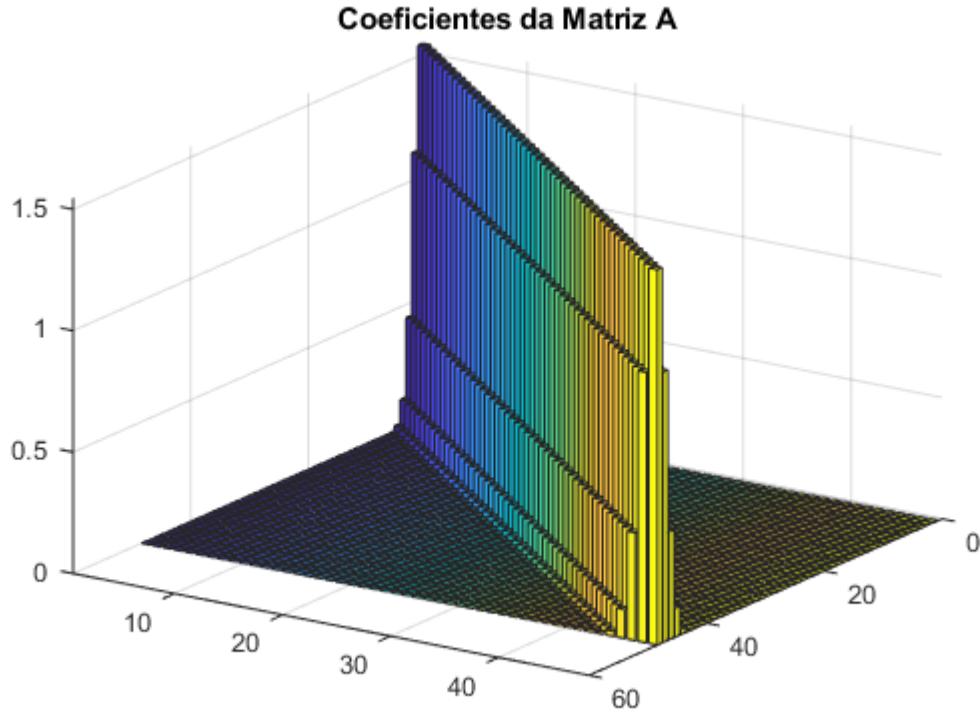


Figura 26 – Coeficientes da matriz de autocorrelação.

O produto  $\mathbf{H}^T \mathbf{r}$  não é constante, uma vez que  $\mathbf{r}$  sempre está se atualizando com os dados da respectiva janela de entrada, mas essa operação só precisa ser realizada uma vez, antes da primeira iteração. Portanto, tal resultado passa a ser definido como matriz  $\mathbf{H}_S$ . Além disso, os termos nulos de  $\mathbf{H}$  e  $\mathbf{A}$  passam agora a ser descartados. A equação pode então ser descrita como:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu(\mathbf{H}_S - \mathbf{A}\mathbf{x}_i) \quad (4.3)$$

A terceira implementação do método (versão 2.0) foi realizada com os termos não nulos de  $\mathbf{H}$  e  $\mathbf{A}$  sendo declarados de forma literal como variáveis. A quarta implementação (versão 2.1) foi realizada com esses valores escritos diretamente no código, como constantes, visando diminuir ainda mais o número de instruções da implementação. O Algoritmo 3, no Anexo A, resume como as operações da Equação 4.3 foram realizadas em  $\mathbf{C}^+$ .

Os relatórios da compilação no SAPHO e no Quartus da versão 2.0 do método podem ser vistos, respectivamente, nas Figuras 27 e 28. A frequência operacional necessária para esta implementação é 764 GHz, ou seja, caiu para menos de 8% do total da frequência operacional que era necessária na implementação anterior.

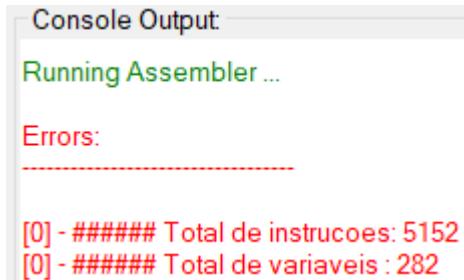


Figura 27 – Compilação do Algoritmo 3 no SAPHO (versão 2.0).

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,270
Total combinational functions	1,257
Dedicated logic registers	92
Total registers	92
Total pins	59
Total virtual pins	0
Total memory bits	102,294
Embedded Multiplier 9-bit elements	2
Total PLLs	0

Figura 28 – Compilação do Algoritmo 3 no Quartus (versão 2.0).

Para a versão 2.1, também foi utilizado o Algoritmo 3, porém os termos não nulos de **H** e **A** não são mais declarados de forma literal (variáveis), passando a ser escritos diretamente no código como constantes. Os relatórios da compilação no SAPHO e no Quartus da versão 2.1 do algoritmo podem ser vistos nas Figuras 29 e 30, respectivamente.

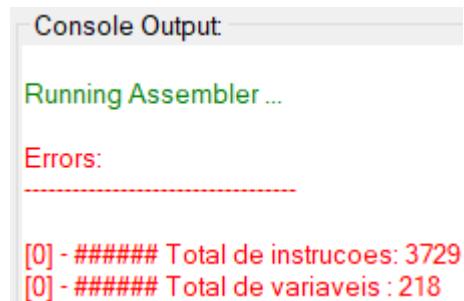


Figura 29 – Compilação do Algoritmo 3 no SAPHO (versão 2.1).

Após realizada a análise do número total de instruções, incluindo as que se repetem no *loop*, dividido pelo tamanho da janela de entrada de dados, foi possível constatar que a frequência de operação necessária para essa implementação é de 447 GHz.

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,254
Total combinational functions	1,241
Dedicated logic registers	91
Total registers	91
Total pins	59
Total virtual pins	0
Total memory bits	84,612
Embedded Multiplier 9-bit elements	2
Total PLLs	0

Figura 30 – Compilação do Algoritmo 3 no Quartus (versão 2.1).

Para a implementação da versão 2.2, descrita no Algoritmo 4 (Anexo A), a Equação 4.3 foi realizada de forma ainda mais direta, ao comparar com as implementações das versões anteriores. O sinal de subtração da expressão foi colocado em evidência e não há mais vetores. Nesse algoritmo, o sinal  $\mathbf{x}$  não é mais representado como um vetor, mas sim declarado termo a termo como 48 variáveis. O sinal de entrada  $\mathbf{r}$  (que no algoritmo é representado por  $\mathbf{d}$ ) também é declarado termo a termo, como 55 variáveis. Dessa forma, evita-se o acesso indireto a memória pelo índice dos *arrays* em  $\mathbf{C}$ , que precisa de mais ciclos de *clock* para executar.

Os relatórios da compilação no SAPHO e no Quartus da versão 2.2 do algoritmo estão nas Figuras 31 e 32, respectivamente.

Para esta implementação, a frequência necessária de operação é 258 GHz. Essa frequência é mais baixa, ao ser comparada com os 10,22 THz obtidos na primeira implementação do método SSF, porém ainda não é factível de implementação com as tecnologias atuais. Um dos principais motivos para esses valores altos da frequência de operação é o grande número de iterações necessárias ao algoritmo.

```
Console Output
Running Assembler ...
Errors:
-----
[0] - ##### Total de instrucoes: 3622
[0] - ##### Total de variaveis : 185
```

Figura 31 – Compilação do Algoritmo 4 no SAPHO (versão 2.2).

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1,253
Total combinational functions	1,240
Dedicated logic registers	91
Total registers	91
Total pins	59
Total virtual pins	0
Total memory bits	88,581
Embedded Multiplier 9-bit elements	2
Total PLLs	0

Figura 32 – Compilação do Algoritmo 4 no Quartus (versão 2.2).

#### 4.2.1 Inicialização Otimizada do Algoritmo

Visando otimizar a implementação do método, é introduzida a matriz pseudo-inversa de  $\mathbf{H}$  (matriz  $\mathbf{B}$  na Figura 33), que permite a inicialização do vetor iterativo  $\mathbf{x}$  com valores mais próximos da solução do problema [70], como visto na Seção 3.1.

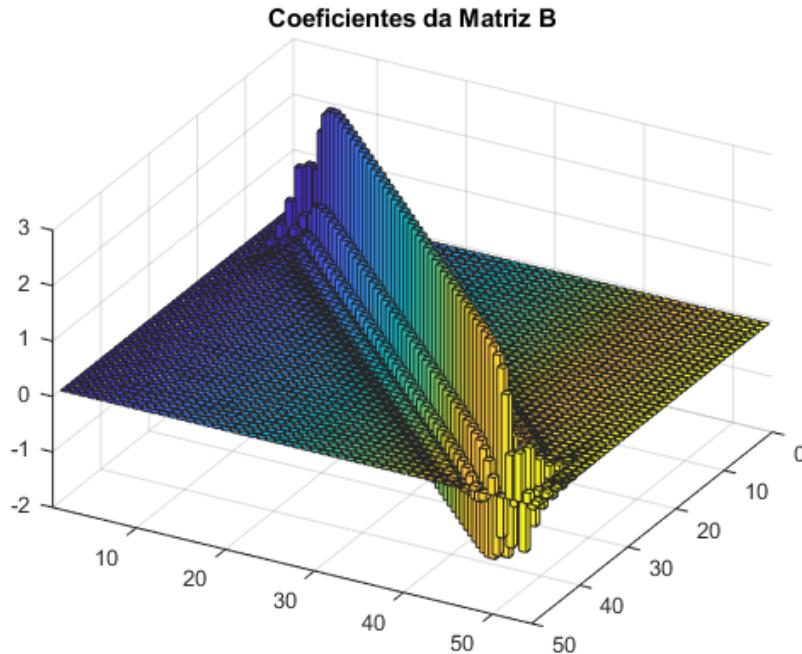


Figura 33 – Coeficientes da matriz pseudo-inversa.

Após realizadas as simulações para verificar o erro RMS pelo número de iterações, é possível analisar a convergência do método SSF no gráfico da Figura 34. Este gráfico mostra que o método SSF padrão converge para o resultado (com erro RMS abaixo de 2,5) para cerca de 150 iterações. Já o método SSF com o pré-processamento na inicialização de  $\mathbf{x}$ , por meio da matriz pseudo-inversa, converge para o resultado com cerca de 18 iterações.

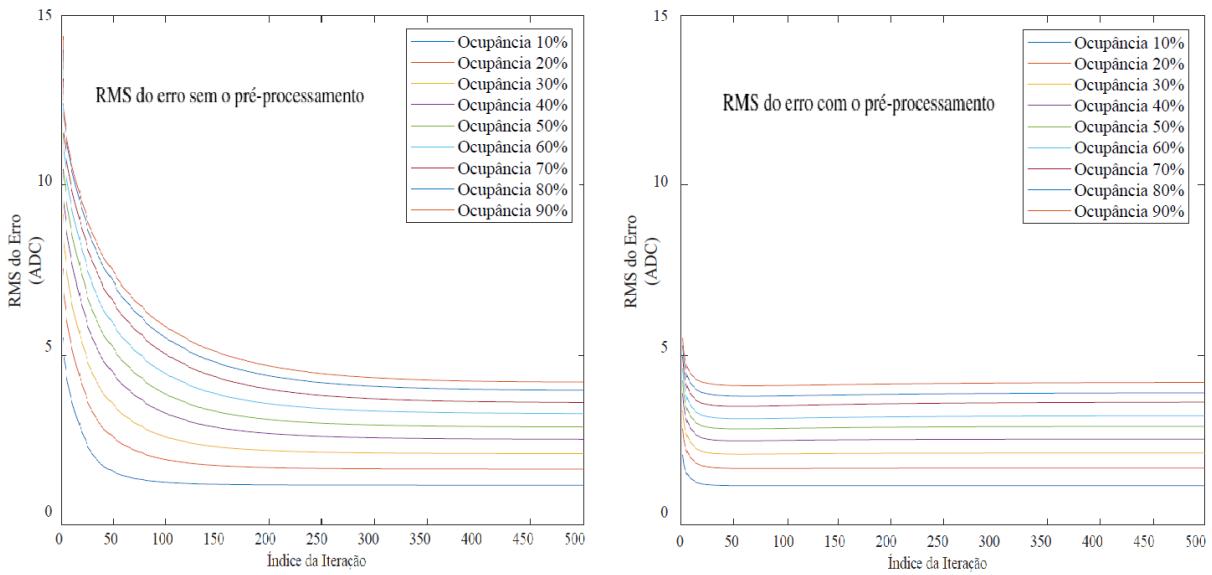


Figura 34 – Erro RMS da estimativa de energia pelo método SSF [70].

Além disso, estudos realizados em [70] mostram que é possível zerar coeficientes próximos de zero das matrizes **H**, **A** e **B**, respeitando um patamar de corte, reduzindo assim o número de operações a serem realizadas pelo processador. No trabalho [70] foi mostrado que a curva de convergência para o método SSF com coeficientes anulados nas matrizes e a curva do SSF para as matrizes originais praticamente se sobrepõem.

A implementação do método SSF com a matriz pseudo-inversa está descrita no Algoritmo 5 (Anexo A), onde os termos próximos de zero das matrizes **H**, **A** e **B** foram descartados. A versão 3.0 desta implementação foi realizada com o processo de inicialização de **x** antes de realizar o *loop* de 18 iterações. A versão 3.1 possui o mesmo algoritmo da versão 3.0, porém foi acrescentada a função PSET (substituindo as estruturas *if* e *else*), que foi aprimorada no SAPHO especialmente para reduzir o número de instruções nas partes do algoritmo em que os termos negativos são anulados. Assim, a diferença entre essas implementações está no número de instruções. O relatório da compilação no Quartus para as versões 3.0 e 3.1 do Algoritmo 5 (Anexo A) possui os mesmos parâmetros em elementos lógicos e pode ser visto na Figura 35.

Os relatórios da compilação no SAPHO das versões 3.0 e 3.1 do Algoritmo 5 (Anexo A) estão nas Figuras 36 e 37, respectivamente. Para a implementação da versão 3.0, a frequência necessária de operação é 23 GHz, o que é menos de 10% da frequência na melhor implementação sem o uso da matriz pseudo-inversa (versão 2.2). Para a implementação da versão 3.1, a frequência de operação caiu para 21 GHz.

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	782
Total combinational functions	782
Dedicated logic registers	50
Total registers	50
Total pins	43
Total virtual pins	0
Total memory bits	108,446
Embedded Multiplier 9-bit elements	2
<b>Total PLLs</b>	<b>0</b>

Figura 35 – Compilação do Algoritmo 5 no Quartus (versões 3.0 e 3.1).

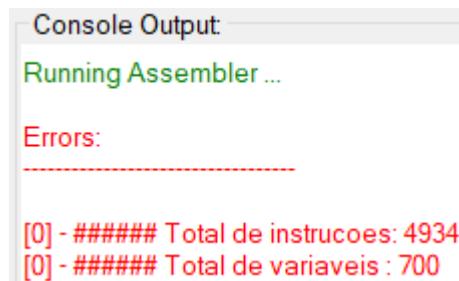


Figura 36 – Compilação do Algoritmo 5 no SAPHO (versão 3.0).

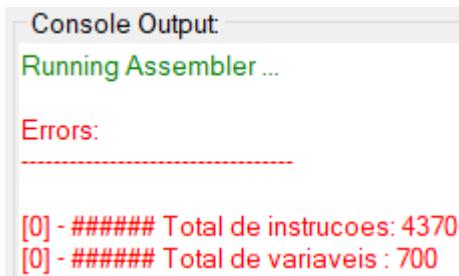


Figura 37 – Compilação do Algoritmo 5 no SAPHO (versão 3.1).

### 4.3 MÉTODO OPERANDO EM PONTO FIXO

Após ter sido realizada toda análise e simplificação do método em ponto flutuante, foi desenvolvido um processador em ponto fixo para operar o algoritmo, visando diminuir ainda mais o custo lógico do processador. Para tal, foi necessário realizar uma quantização dos elementos das matrizes de convolução e pseudo-inversa, de forma que os valores menores que 1 nas operações não fossem zerados quando truncados e também para que uma precisão mínima fosse mantida. No gráfico Figura 38 está a relação entre os valores para o ganho e os respectivos valores RMS do erro, onde é possível notar que um ganho de  $2^5$  já é suficiente para minimizar o erro. Em termos de representação numérica, isto é equivalente a dizer que os números são representados com precisão numérica de  $2^{-5}$ .

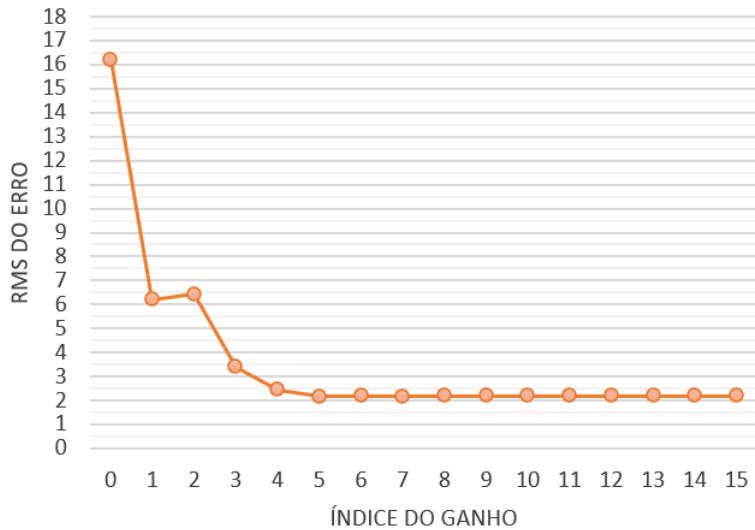


Figura 38 – Relação entre o valor RMS do erro e o ganho.

Com o intuito de garantir uma margem de segurança, o fator de quantização escolhido foi  $2^7$  (ou seja, um ganho de 128), garantindo que os coeficientes das matrizes **H**, **A** e **B** não sejam anulados nas operações.

Além disso, foram realizados testes variando o número de bits do processador, onde foi constatado que são necessários pelo menos 29 bits para que não haja comprometimento no desempenho. Assim, para o presente trabalho, o número de bits foi definido como 32.

Essa implementação seguiu a linha de raciocínio da melhor versão do processador operando em ponto flutuante com a função PSET (Algoritmo 5, Anexo A), porém todas as variáveis e constantes utilizadas devem possuir valores inteiros. Para isso, foi necessário acrescentar produtos e divisões pelo ganho escolhido ao desenvolver o código em **C<sub>+</sub>**.

Os relatórios da compilação no SAPHO e no Quartus para essa implementação do método SSF podem ser vistos, respectivamente, nas Figuras 39 e 40. A frequência de operação necessária é de 23 GHz e esse aumento já era esperado, devido ao aumento no numero de operações para a aplicação do ganho necessário ao algoritmo em ponto fixo.

```
Console Output
Running Assembler ...
Errors:
-----
[0] - ##### Total de instrucoes: 4473
[0] - ##### Total de variaveis : 407
```

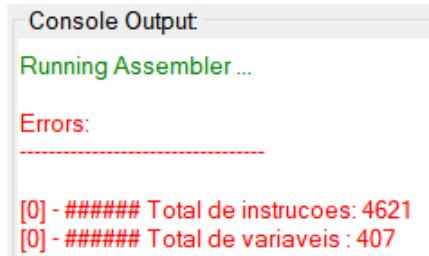
Figura 39 – Compilação do método SSF em ponto fixo no SAPHO.

<b>Family</b>	Cyclone IV E
<b>Device</b>	EP4CE115F29C7
<b>Timing Models</b>	Final
<b>Total logic elements</b>	1,621
<b>Total registers</b>	60
<b>Total pins</b>	70
<b>Total virtual pins</b>	0
<b>Total memory bits</b>	98,011
<b>Embedded Multiplier 9-bit elements</b>	6
<b>Total PLLs</b>	0

Figura 40 – Compilação do método SSF em ponto fixo no Quartus.

O valor definido para o ganho é constante, logo, esse termo é repetido em todas as operações de divisão. Visando reduzir o custo lógico dessas operações, foi criada no SAPHO a instrução NORM, já apresentada na em 4.1.1. Essa instrução é definida na ULA do processador e tem a função de normalizar o número desejado pelo valor constante do ganho ( $2^7$ ). Com isso, os circuitos de divisão são simplificados pois a ULA não precisa mais verificar ambas as entradas nessa operação.

Com isso, os relatórios da compilação no SAPHO e no Quartus para a implementação final realizada em ponto fixo do método SSF, que utiliza a função NORM, podem ser vistos, respectivamente, nas Figuras 41 e 42.



```
Console Output
Running Assembler ...
Errors:
-----
[0] - ##### Total de instrucoes: 4621
[0] - ##### Total de variaveis : 407
```

Figura 41 – Compilação do método SSF em ponto fixo (final) no SAPHO.

Como a diferença entre as duas implementações do método em ponto fixo está apenas na substituição do circuito de divisão pela operação de normalização, a frequência do processamento se manteve em cerca de 23 GHz.

O que foi alterado de forma significativa ao comparar essas duas implementações foi o número de elementos lógicos, que foi reduzido de 1.621 para 382.

Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	382
Total registers	61
Total pins	70
Total virtual pins	0
Total memory bits	100,823
Embedded Multiplier 9-bit elements	6

Figura 42 – Compilação do método SSF em ponto fixo (final) no Quartus.

#### 4.4 ESTRUTURA MULTICORE

Como a melhor versão da implementação do método SSF em ponto fixo possui uma frequência de processamento que ainda não é factível para que o processador possa operar de forma individual, foi projetada uma estrutura com múltiplas instâncias [88], visando respeitar a taxa de eventos do sistema de aquisição de dados do ATLAS.

Na Figura 43 está ilustrado um diagrama da estrutura *multicore* para uma quantidade  $n$  de processadores, em paralelo, com funcionamento em *pipeline* [89].

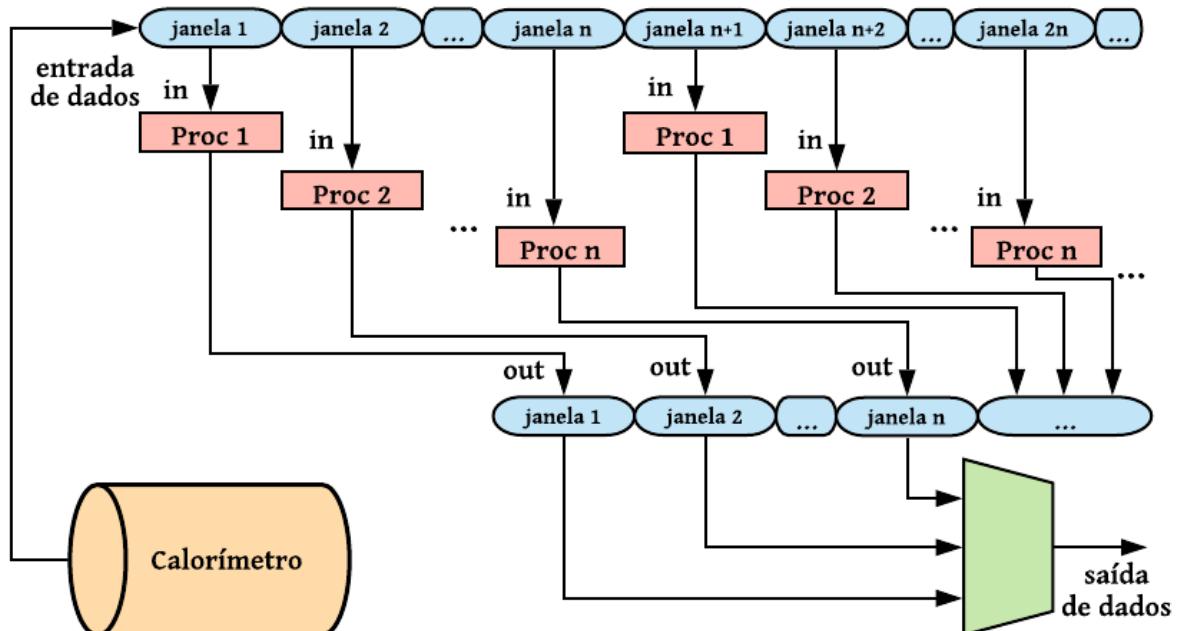


Figura 43 – Diagrama da estrutura *multicore* implementada.

Nesta estrutura, cada janela de dados é executada por um processador, mantendo assim um fluxo contínuo e sequencial de processamento. Após cada processador receber sua janela de dados de entrada, as operações dos mesmos são realizadas de forma independente entre si. Na saída, um multiplexador recebe as janelas contendo o sinal reconstruído por cada processador, respectivamente, no instante correto.

A janela de dados de saída do primeiro processador e a janela de dados de entrada do último processador da estrutura ocorrem de forma simultânea, assim, o primeiro processador pode então carregar a sua janela de dados de entrada novamente o processo se repete sem que ocorra perda de dados.

Com isso, foi possível sincronizar a estrutura *multicore* aqui apresentada com a taxa de eventos do LHC, de forma que os dados das janelas de entrada e de saída ocorrem a cada 25 ns. Além disso, essa estrutura foi implementada para diferentes quantidades de processadores, em paralelo, operando em diferentes frequências.

Para tal, foi necessário analisar a quantidade de *clocks* entre cada dado na respectiva janela de entrada. Na Figura 44 é possível observar, ao lado esquerdo, a janela de entrada de dados sendo carregada com a função `in()`, em  $C^+$ , que pega o dado da porta de entrada desejada e, ao lado direito, está destacado o respectivo código em *Assembly* gerado, que contém 4 instruções para cada entrada.

18	<code>int d_0 = in(0);</code>	13	<code>LOAD 0</code>
19	<code>int d_1 = in(0);</code>	14	<code>PUSH</code>
20	<code>int d_2 = in(0);</code>	15	<code>IN</code>
21	<code>int d_3 = in(0);</code>	16	<code>SET maind_0</code>
22	<code>int d_4 = in(0);</code>	17	<code>LOAD 0</code>
23	<code>int d_5 = in(0);</code>	18	<code>PUSH</code>
24	<code>int d_6 = in(0);</code>	19	<code>IN</code>
25	<code>int d_7 = in(0);</code>	20	<code>SET maind_1</code>
26	<code>int d_8 = in(0);</code>	21	<code>LOAD 0</code>
27	<code>int d_9 = in(0);</code>	22	<code>PUSH</code>
28	<code>int d_10 = in(0);</code>	23	<code>IN</code>
29	<code>int d_11 = in(0);</code>	24	<code>SET maind_2</code>
30	<code>int d_12 = in(0);</code>	25	<code>LOAD 0</code>

Figura 44 – Código para o preenchimento da janela de entrada de dados.

Foi possível observar, após simular no ModelSim-Altera, que o espaçamento entre cada pulso de entrada de dados é de 4 ciclos de *clock*. Portanto, para que os pulsos de entrada ocorram sincronizados com a taxa de eventos de 40 MHz do LHC (a cada 25 ns), foi contabilizado que é necessário que o processador opere a uma taxa de  $4 \times 2 \times (40 \text{ MHz}) = 320 \text{ MHz}$ . Além disso, sabe-se que o tamanho de uma janela de entrada é  $55 \times (25 \text{ ns}) = 1.375 \text{ ns}$  e, no ModelSim-Altera, foi observado que o processador levou 186.781 ns para realizar as operações, após ter preenchido sua janela de entrada, para então poder começar a preencher a janela de dados de saída.

Dessa forma, seriam necessários que outros  $(186.781 \text{ ns})/(1.375 \text{ ns}) = 136$ , 12 processadores completassem suas janelas de dados de entrada, sequencialmente, para que o primeiro processador tivesse tempo de terminar seu processamento e pudesse, enfim, carregar a sua nova janela de entrada. Foi então necessário adicionar instruções neutras<sup>5</sup> no código em *Assembly*, visando adicionar ciclos de *clock*, de forma que o tempo de processamento fosse suficiente para que 137 processadores carregassem suas janelas de entrada. Estas instruções neutras foram adicionadas ao final do código, antes de a janela de saída de dados começar a ser carregada. Tal procedimento foi possível devido a uma modificação na IDE e estrutura do SAPHO, que passou a permitir que o compilador C e o Assembler fossem executados de forma separada.

Vale ressaltar que, partir desta modificação no SAPHO, foi possível realizar a programação, edição e compilação do código *Assembly* de forma direta e independente do código  $C^+$  na IDE, uma vez que, antes disso, o código *Assembly* era gerado automaticamente, de acordo com o respectivo código  $C^+$ , não podendo então ser modificado.

Assim, o procedimento de adicionar ciclos de *clock* foi realizado de forma que o tempo de processamento pudesse ocorrer no período de  $137 \times (1.375 \text{ ns}) = 188.375 \text{ ns}$  e, com isso, a frequência de operação de 320 MHz precisou de 138 processadores, no total, operando em paralelo, como havia sido explicitado na Figura 43.

Seguindo a mesma lógica, este procedimento foi realizado para outros valores de frequência, uma vez que, ao adicionar ciclos de *clock* entre os pulsos de entrada, é necessário aumentar a frequência operacional de forma a manter o período de 25 ns entre estes pulsos. Por exemplo, ao adicionar 1 ciclo de *clock*, a frequência operacional necessária passa a ser  $5 \times 2 \times (40 \text{ MHz}) = 400 \text{ MHz}$  e, além disso, as operações passam a ser realizadas em um período menor, necessitando de menos processadores em paralelo, uma vez que a janela de dados de entrada se mantém em  $55 \times (25 \text{ ns}) = 1.375 \text{ ns}$ . Tais detalhes serão apresentados e comparados, a seguir, na seção de Resultados.

---

<sup>5</sup> A instrução neutra escolhida em *Assembly* foi a MLT 1, a qual não realiza nenhuma alteração nas operações, apenas faz um produto pelo elemento neutro da multiplicação 1.

## 5 RESULTADOS

Após realizadas as simulações de cada versão das implementações do método SSF em FPGA, foi possível conferir o funcionamento dos algoritmos e realizar as operações necessárias para obter a estimativa de energia.

A relação entre as características da implementação de cada versão dos processadores, operando de forma individual, pode ser vista na Tabela 3. Essa tabela mostra que a otimização da inicialização do método SSF com o uso da matriz pseudo-inversa, a aplicação do patamar de corte para os termos próximos de zero nas matrizes e o aprimoramento na estrutura do SAPHO para as novas funções (PSET e NORM) ajudaram a reduzir a frequência necessária de operação de forma significativa.

Tabela 3 – Comparaçāo entre as implementaçāes do mētodo SSF.

Versão*	Frequência (GHz)	Instruções	Variáveis	Elementos Lógicos	Bits de Memória
<b>v1.0</b>	10.224	33.674	5.510	1.405	936.864
<b>v1.1</b>	10.050	34.203	5.557	1.407	955.518
<b>v2.0</b>	764	5.152	282	1.270	102.294
<b>v2.1</b>	447	3.729	218	1.254	84.612
<b>v2.2</b>	258	3.622	185	1.253	88.581
<b>v3.0</b>	23	4.934	700	782	108.446
<b>v3.1</b>	21	4.370	700	782	108.446
<b>ponto fixo</b>	23	4.473	407	1.621	98.011
<b>versão final</b>	23	4.621	407	382	100.803

\*Uma descrição resumida das principais características de cada uma destas versões implementadas está apresentada no Anexo B.

Ainda na Tabela 3, é possível notar também que a melhor implementação ponto fixo do método SSF, apesar de ter seguido a mesma linha de raciocínio da melhor implementação em ponto flutuante, ficou com uma frequência de operação um pouco maior (devido ao aumento no número de operações para a aplicação do ganho), porém o custo lógico foi reduzido, o que é mais vantajoso.

Após realizada a análise das características dos processadores operando de forma individual, foi analisada também a estrutura *multicore*. No gráfico da Figura 45 é possível analisar como os parâmetros Número de Processadores e Tempo de Atraso  $\Delta t$  (que é o tempo necessário para que os dados do sinal reconstruído possam ser descarregados na saída em um fluxo contínuo e sequencial) variam de acordo com a frequência do *clock* de operação dos processadores.

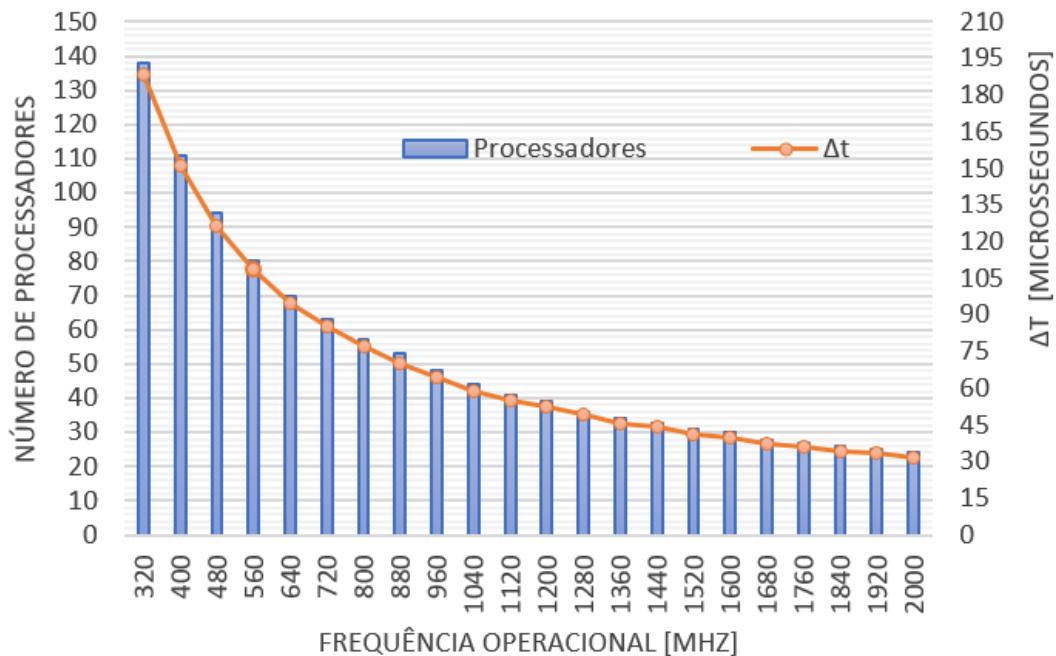


Figura 45 – Quantidade de processadores e tempo de atraso variando com a frequência.

Já na Figura 46 é possível observar como o consumo de recursos de *hardware* como elementos lógicos e blocos multiplicadores embarcados diminui com o aumento da frequência de operação. Vale ressaltar que o Quartus equilibra a alocação entre tais recursos de forma automática.

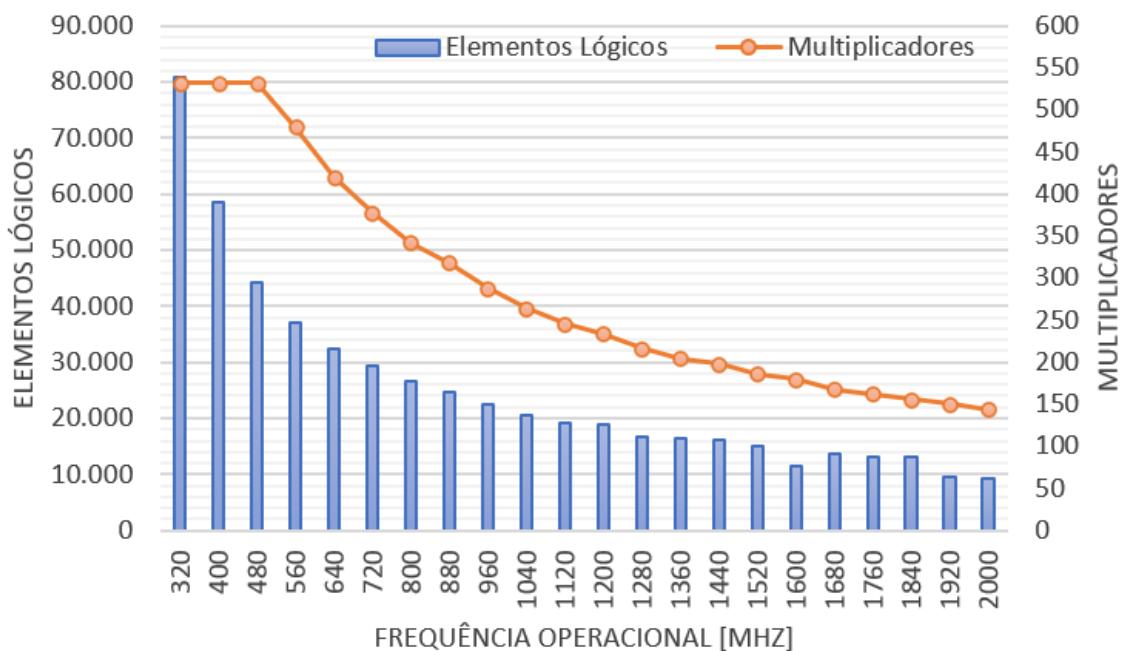


Figura 46 – Elementos lógicos e multiplicadores variando com a frequência.

Foi realizada também uma análise dos bits de memória necessários para a implementação da estrutura *multicore* desenvolvida de acordo com a frequência de operação. Os resultados são apresentados na Figura 47.

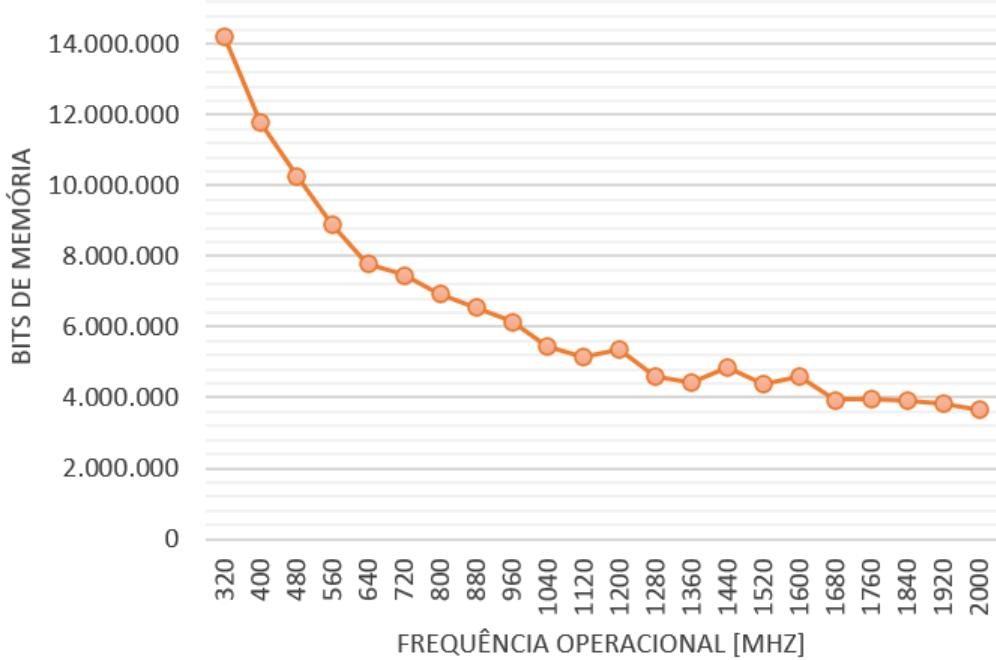


Figura 47 – Bits de memória variando com a frequência.

Mesmo para a configuração da estrutura *multicore* que possui o maior requisito de memória (cerca de 14 Mbits), a implementação é factível para operação em FPGAs atuais, que podem possuir mais de 200 Mbits de memória disponíveis [85].

Na Figura 48 é possível observar, na interface do ModelSim-Altera, o sinal a ser reconstruído pelo processador (com frequência operacional de 320 MHz), vindo dos canais de leitura do TileCal, onde cada pulso representa o instante em que um dado chega e, como pode-se notar, os pulsos ocorrem a cada 25 ns, de forma que está sincronizado com a taxa de eventos do LHC.

Por fim, para efetivar a proposta do presente trabalho de realizar a estimação de energia, no gráfico da Figura 49 é possível observar três sinais: (i) o sinal com *pile-up*, representando os dados vindos dos canais de leitura do TileCal à taxa de 40 MHz; (ii) o sinal alvo, que representa o sinal do calorímetro sem nenhum efeito de empilhamento, ou seja, seria o sinal ideal que se deseja obter; (iii) o sinal reconstruído pelo processador desenvolvido neste trabalho, que opera o método SSF com a inicialização otimizada do algoritmo e um *loop* de 18 iterações. Destaca-se o RMS abaixo de 2,5 que foi explicitado na Figura 38, para a escolha do ganho referente ao processamento em ponto fixo.

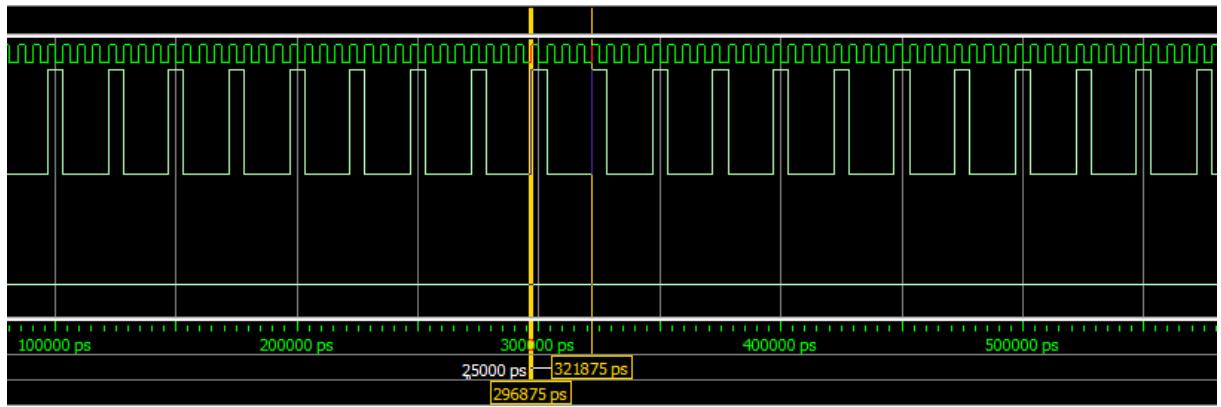


Figura 48 – Janela de entrada dos dados do método SSF no Modelsim-Altera.

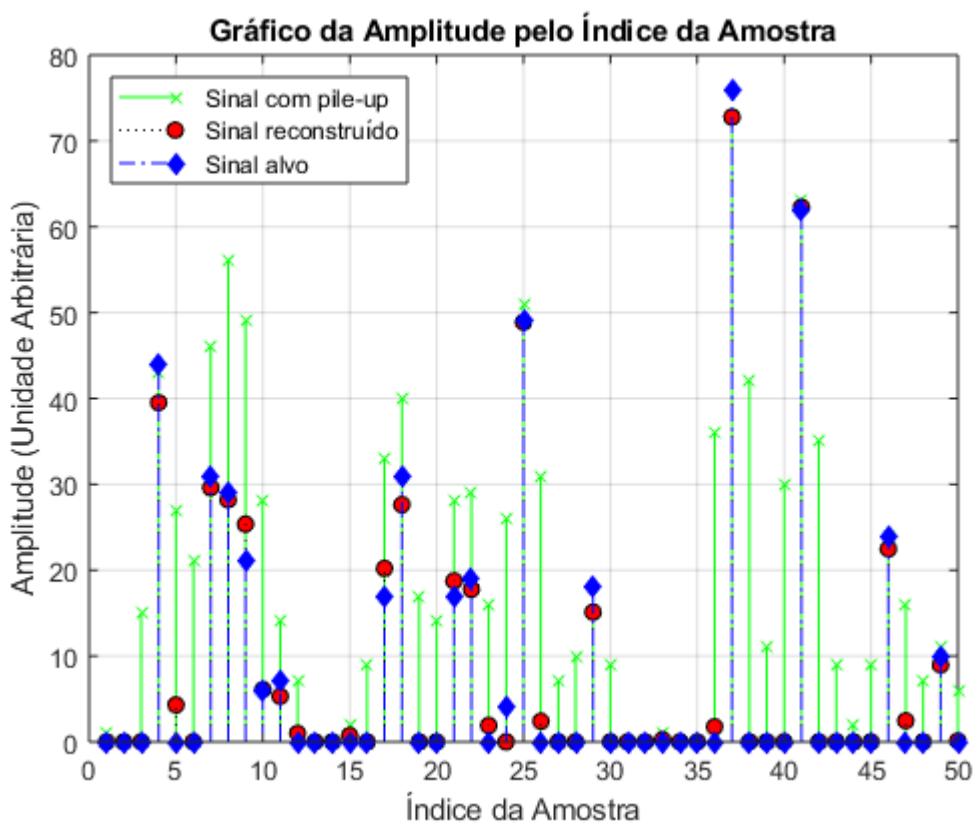


Figura 49 – Comparação entre o sinal reconstruído, o alvo e o sinal ruidoso.

## 6 CONCLUSÕES

Nesse trabalho foram apresentadas diversas topologias para uma implementação com baixo consumo de recursos lógicos em FPGA, tanto no formato de ponto fixo quanto no de ponto flutuante, de processadores capazes de operar um método iterativo de deconvolução baseado em representação esparsa de dados, o SSF, visando realizar a estimativa de energia em Calorímetros de Altas Energias, com foco no Calorímetro Hadrônico do ATLAS. Tal método consiste basicamente em operações matriciais de soma e multiplicação, além disso, grande parte dos termos das matrizes utilizadas são nulos, reduzindo o número de operações necessárias para a implementação do algoritmo.

Um dos destaques na implementação dos algoritmos foi a otimização na inicialização do vetor iterativo com o uso do pré-processamento com a matriz pseudo-inversa. Apesar da necessidade de um cálculo matricial adicional, foi possível diminuir de o número de iterações de 150 para 18, diminuindo assim o número de ciclos de *clock* em uma ordem de grandeza. Além disso, o fato de diversos coeficientes das matrizes do problema terem sido anulados, após análise e aplicação de um patamar de corte, tornou possível diminuir consideravelmente o número de multiplicações a serem executadas pelo processador.

Outro destaque foi o fato de que as modificações na estrutura do SAPHO (para as novas instruções PSET e NORM) permitiram diminuir relativamente a quantidade de instruções e custo lógico dos processadores e isso poderá ser utilizado em implementações futuras, uma vez que o SAPHO possui código aberto.

Assim, é possível concluir que o presente trabalho contribuiu com diversas questões como: (i) a estimativa *online* de energia no primeiro nível de *trigger* do TileCal; (ii) o desenvolvimento de um processador embarcado customizável em FPGA; (iii) a implementação em FPGA de forma dedicada do método SSF; (iv) um aprimoramento na inicialização do método SSF; (v) otimizações na estrutura do SAPHO; (vi) o desenvolvimento de uma estrutura *multicore* de processamento; (vii) comparações entre diversas implementações de algoritmos em FPGA com o processador customizado que foi aprimorado neste trabalho.

Para trabalhos futuros, a proposta é realizar novas modificações na Unidade Lógico-Aritmética e na estrutura do processador para otimizar o tempo de atraso  $\Delta t$  e o custo lógico da arquitetura *multicore* apresentada. Além disso, outros métodos capazes de realizar a reconstrução de sinais para a calorimetria de altas energias estão sendo estudados para implementações de forma dedicada no SAPHO, assim como foi realizado no presente trabalho, com destaque para um método baseado em Redes Neurais Artificiais [90], que está sendo implementado atualmente e tem apresentado resultados promissores.

## REFERÊNCIAS

- [1] PERKINS, D. H. *Introduction to High Energy Physics*. Cambridge University Press, April 2000.
- [2] WILLE, K. *The Physics of Particle Accelerators: An Introduction*. Clarendon Press, 2000.
- [3] LINDE, A.; LINDE, D.; MEZHLUMIAN, A. *From the Big Bang Theory to the Theory of a Stationary Universe*. Physical Review D, v. 49, n. 4, p. 1783, 1994.
- [4] THE ATLAS COLLABORATION. *Measurements of Higgs boson Production and Couplings in Diboson Final States with the ATLAS Detector at the LHC*. Physics Letters B, vol. 726, pp. 88-119, Oct 2013.
- [5] ABDALLA, B. *O Maior Acelerador de Partículas do Mundo passa por um Upgrade*. Jornal da USP, 2019.
- [6] VIEIRA, C. L. *História da Física*. Rio de Janeiro, CBPF, 2015.
- [7] BEZRUKOV, F. and SHAPOSHNIKOV, M. *The Standard Model Higgs Boson as the Inflaton*. Physics Letters B, v. 659, n. 3, p. 703-706, 2008.
- [8] CERN. *About CERN*. CERN Document Server, 2012.
- [9] NAKAHAMA, Y. *The ATLAS Trigger System: Ready for Run-2*. CERN Document Server, ATL-DAQ-PROC-2015-006, Geneva, 2015.
- [10] ANDRADE FILHO, L. M.; PERALVA, B. S.; SEIXAS, J. M.; CERQUEIRA A. S. *Calorimeter Response Deconvolution for Energy Estimation in High-Luminosity Conditions*. IEEE Transactions on Nuclear Science, 62(6):3265–3273, Dec 2015.
- [11] ANTHONY, K. *Celebrating the First of a Kind*. Sep 2014.
- [12] WIGMANS, R. *Calorimetry*. Oxford University Press, 2018.
- [13] DUARTE, J. P. B. S. *Técnicas de Deconvolução Aplicadas à Estimação de Energia Online em Calorimetria de Altas Energias em Condições de Alta Taxa de Eventos*. Anais do XX Congresso Brasileiro de Automática, pp. 1-6, 2016.
- [14] MITRA, S. K. *Digital Signal Processing : A Computer-Based Approach*. McGraw-Hill Higher Education, New York, 2001.
- [15] BARBOSA D. P.; ANDRADE FILHO, L. M.; CERQUEIRA, A. S.; SEIXAS, J. M. *Sparse Representation for Signal Reconstruction in Calorimeters Operating in High Luminosity*. IEEE Transactions on Nuclear Science, 64(7):1942-1949, Jul 2017.
- [16] DAUBECHIES, I.; DEFRISE, M.; DE MOL; C. *An Iterative Thresholding Algorithm for Linear Inverse Problems with a Sparsity Constraint*. Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences, vol 57, pp. 1413-1457, 2004.
- [17] MEYER, B. U. *Digital Signal Processing with Field Programmable Gate Arrays*. Heidelberg, 2007.

- [18] CALLAN, C.; DICKE, R. H.; PEEBLES, P. J. E. *Cosmology and Newtonian Mechanics*. American Journal of Physics, v. 33, n. 2, p. 105-108, 1965.
- [19] SAKURAI, J. J.; COMMINS, E. D. *Modern Quantum Mechanics*. Revised Edition. 1995.
- [20] AFSHAR S. S. *Paradox in Wave-Particle Duality*. Foundations of Physics, v. 37, n. 2, p. 295–305, 2007.
- [21] PEREIRA R. A. *Estimação de Energia em um Calorímetro Finamente Segmentado*. Tese de Mestrado, UFRJ/COPPE, 2014.
- [22] BUSCH, P.; HEINONEN, T.; LAHTI, P. *Heisenberg's uncertainty principle*. Physics Reports, v. 452, n. 6, p. 155-176, 2007.
- [23] WILLIAMS, E. R. *Accurate Measurement of the Planck Constant*. Physical Review Letters, v. 81, n. 12, p. 2404, 1998.
- [24] EINSTEIN, A. *The Meaning of Relativity: Including the Relativistic Theory of the Non-Symmetric Field*. Princeton University Press, 2014.
- [25] ROONEY, A. *A História da Física*. São Paulo, M. Books do Brasil Editora Ltda, 2013.
- [26] BRAIBANT, S.; GIACOMELLI, G.; SPURIO, M. *Particles and Fundamental Interactions*. New York, Springer-Verlag GMBH, 2011.
- [27] CERN. *The Large Hadron Collider*. CERN Accelerating Science. Disponível em: <https://home.cern/science/accelerators/large-hadron-collider>
- [28] MOUCHE, P. *Overall View of the LHC*. CERN Document Server, OPEN-PHO-ACCEL-2014-001, Jun, 2014.
- [29] BRICE, M. *LHC Restart Run 2*. CERN Document Server, April, 2015.
- [30] AAD, G. *Observation of a New Particle in the Search for the Standard Model Higgs boson with the ATLAS*. Phys. Lett. vol B716, pp. 1–29, 2012.
- [31] AGGLETON, R.; CMS COLLABORATION. *An FPGA Based Track Finder for the L1 Trigger of the CMS Experiment at the High Luminosity LHC*. Journal of Instrumentation, vol 12, p. P12019, 2017.
- [32] BOURGEOIS, D.; FITZPATRICK, C.; STAHL, S. *Using Holistic Event Information in the Trigger*. CERN Document Server, LHCb-PUB-2018-010, Geneva, 2018.
- [33] RONCHETTI, F.; BLANCO, F.; FIGUEIREDO, M.; KNOSPE, A. G.; XAPLANTERIS, L. *The ALICE Electromagnetic Calorimeter High Level Triggers*. JJ. Phys. Conf. Ser. vol 396, 2012.
- [34] AAD, G. *The ATLAS Experiment at the CERN Large Hadron Collider*. JINST, vol. 3, pp. S08003, 2008.
- [35] THE TOTEM COLLABORATION. *TOTEM Technical Design Report*. Technical Report, CERN/LHCC 2004-002, 2004.

- [36] THE LHCf COLLABORATION. *Technical Design Report of the LHCf Experiment*. Technical Report, CERN/LHCC 2006-004, 2006.
- [37] EVANS, L. R.; BRYANT, P. *LHC Machine*. Journal of Instrumentation, vol. 3, pp. S08001.164, 2008.
- [38] PEQUENAO, J. *Computer Generated Image of the Whole ATLAS Detector*. CERN Document Server, CERN-GE-0803012, Geneva, 2008.
- [39] ROS, E. *ATLAS Inner Detector*. Nuclear Physics B Proceedings Supplements, vol. 120, pp. 235–345, 2003.
- [40] PALESTINI, S. *The Muon Spectrometer of the ATLAS Experiment*. Nuclear Physics B Proceedings Supplements, vol. 125, pp. 237–345, 2003.
- [41] ALEKSA, M.; CLELAND, W.; ENARI, Y. *ATLAS Liquid Argon Calorimeter Phase-I Upgrade Technical Design Report*. CERN Document Server, CERN-LHCC-2013-017 and ATLAS-TDR-022, Geneva, 2013.
- [42] CARRIO, F.; KIM, H. Y.; MORENO, P.; REED, R.; SANDROK, C. *Design of an FPGA-Based Embedded System for the ATLAS Tile Calorimeter Front-End Electronics Test-Bench*. CERN Document Server, ATL-TILECAL-PROC-2013-017, Geneva, 2013.
- [43] PEQUENAO, J. *Computer Generated Image of the ATLAS Calorimeter*. CERN Document Server, CERN-GE-0803015, 2008.
- [44] ANDRADE FILHO, L. M. *Detecção e Reconstrução de Raios Cósmicos Usando Calorimetria de Altas Energias*. Tese de Doutorado, COPPE/UFRJ, Março, 2009.
- [45] PEQUENAO, J.; SCHAFFNER, P. *How ATLAS Detects Particles: Diagram of Particle Paths in the Detector*. CERN Document Server, CERN-EX-1301009, Geneva, Jan 2013.
- [46] PASCHOALIN, T. C. *Reconstrução de Energia em Calorímetros Operando em Alta Taxa de Luminosidade Usando Estimadores de Máxima Verossimilhança*. Tese de Mestrado, UFJF, Março, 2016.
- [47] PERALVA, B. S. M.; ANDRADE FILHO, L. M. A.; CERQUEIRA, A. S. *The TileCal Energy Reconstruction for Collision Data Using the Matched Filter*. CERN Document Server, ATL-TILECAL-PROC-2013-023, Geneva, 2013.
- [48] KLOUS, S. *Event Streaming in the Online System*. CERN Document Server, ATL-DAQ-PROC-2010-017, Geneva, 2010.
- [49] WATTS, G. *Review of Triggering* Nuclear Science Symposium Conference Record, IEEE, v. 1, pp. 282-287, 2003.
- [50] THE ATLAS HLT/DAQ/DCS GROUP. *ATLAS High-Level Trigger, Data Acquisition and Controls*. ATLAS Technical Report, TDR-016, CERN, 2001.
- [51] HADLEY, D. R. *Digital Filtering Performance in the ATLAS Level-1 Calorimeter Trigger*. Real Time Conference (RT), 7th IEEE-NPSS, pp. 1–6. IEEE, 2010.

- [52] SANCHEZ, C. A. S. *Implementation of the ROD Crate DAQ Software for the ATLAS Tile Calorimeter and a Search for a MSSM Higgs Boson Decaying into Tau Pairs*. PhD Thesis, Universitat de Valencia - CSIC, Valencia, Spain, 2010.
- [53] ARMSTRONG, S.; ANJOS, A.; BAINES, J. *Implementation and Performance of the High Level Trigger Electron and Photon Selection for the ATLAS Experiment at the LHC*. Nuclear Science Symposium Conference Record, IEEE, vol. 4, pp. 2038–2042, oct, 2004.
- [54] RUGGIERO, F. *LHC Accelerator R&D and Upgrade Scenarios*. The European Physical Journal C - Particles and Fields, v. 34, pp. 433-442, 2004.
- [55] CEPEDA, M.; GORI, S.; ILTEN, P. *Report from Working Group 2: Higgs Physics at the HL-LHC and HE-LHC*. CERN Yellow Rep. Monogr., v. 7, pp. 221-584. 364 p., 2018.
- [56] CLELAND, W. E.; STERN, E. G. *Signal Processing Considerations for Liquid Ionization Calorimeters in a High Rate Environment*. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 338, pp. 467–497, Jan 1994.
- [57] CHAPMAN, J. *ATLAS Simulation Computing Performance and Pile-up Simulation in ATLAS*. LPCC Detector Simulation Workshop CERN, 2011.
- [58] BARBOSA D. P. *Estudo de Técnicas de Otimização para Reconstrução de Energia de Jatos no Primeiro Nível de Seleção de Eventos do Experimento ATLAS*. Tese de Mestrado, Juiz de Fora, MG, 2012.
- [59] PERALVA, B. S. M. *Detecção de Sinais de Estimação de Energia para Calorimetria de Altas Energias*. Tese de Mestrado, Juiz de Fora, MG, 2012.
- [60] DUARTE, J. P. B. S. *Estudo de Técnicas de Deconvolução para Reconstrução de Energia Online no Calorímetro Hadronico do ATLAS*. Tese de Mestrado, UFJF, Juiz de Fora, MG, 2015.
- [61] BARBOSA D. P. *Estimação de Energia para Calorimetria em Física de Altas Energias Baseada em Representação Esparsa*. Tese de Doutorado, Juiz de Fora, MG, 2017.
- [62] GILAT, A. *Matlab com Aplicações em Engenharia* Bookman Editora, 2009.
- [63] GARVEY, J.; REES, D. *Bunch Crossing Identification for the ATLAS Level-1 Calorimeter Trigger*. CERN Document Server, CERN-ATL-DAQ-96-051, Geneva, 1996.
- [64] KAY, S. M. *Fundamentals of Statistical Signal Processing*. Prentice Hall, vol 1, New Jersey, 1993.
- [65] TEIXEIRA, T. A.; DUARTE, J. P. B. S.; ANDRADE FILHO, L. M.; SEIXAS, J. M. *Implementação em FPGA de um Método Recursivo de Deconvolução Aplicado em Calorímetros Operando a Alta Taxa de Eventos*. XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, Campina Grande, Paraíba, 2018.
- [66] ELAD, M. *Sparse and Redundant Representations*. Springer New York, New York, 2010.

- [67] MATOUSEK, J.; GARTNER, B. *Understanding and Using Linear Programming*. Springer-Verlag Berlin Heidelberg, 1st Ed., Heidelberg, 2007.
- [68] TEIXEIRA, T. A. *Implementação de Métodos Iterativos de Deconvolução para Processamento On-line no Calorímetro Hadrônico do ATLAS*. Tese de Doutorado, UFJF, Juiz de Fora, MG, 2019.
- [69] YPMA, T. J. *Historical Development of the Newton–Raphson Method*. SIAM Review, v. 37, n. 4, p. 531–551, 1995.
- [70] AGUIAR, M. S.; RESENDE, M. O.; TEIXEIRA, T.; ANDRADE FILHO, L. M; SEIXAS, J. M. *Implementação em FPGA de um Método Iterativo de Deconvolução para Operar no Sistema de Trigger do Experimento ATLAS*. XXII Encontro Nacional de Modelagem Computacional e X Encontro de Ciências e Tecnologia de Materiais, Juiz de Fora, MG, 2018.
- [71] ROCKAFELLAR, R. T. *Lagrange Multipliers and Optimality*. SIAM Review, v. 35, n. 2, p. 183–238, 1993.
- [72] MODELSIM. *Intel FPGA Edition Simulation Quick-Start*. Intel, UG-01102, 2019.
- [73] YIANNACOURAS, P., STEFFAN, J. G., ROSE, J. *Exploration and Customization of FPGA-Based Soft Processors*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 26, n. 2, pp. 266–277, Feb, 2007.
- [74] MAKNI, M., BAKLOUTI, M., NIAR, S., JMAL, M. W., ABID, M. *A Comparison and Performance Evaluation of FPGA Soft-Cores for Embedded Multi-Core Systems*. 11th International Design Test Symposium (IDT), pp. 154–159, Dec, 2016.
- [75] SANTOS, V. A. M.; SILVA, L. R. M.; ANDRADE FILHO, L. M. *Implementação de Circuitos Aritméticos em Ponto Flutuante Utilizando Formato com Número de bits Configurável*. Congresso Brasileiro de Automática, pp. 1–2, João Pessoa, 2018.
- [76] KAPISCH, E. B.; SILVA, L. R. M.; MARTINS, C. H. N.; BARBOSA, A. S.; ANDRADE FILHO, L. M.; DUQUE, C. A.; TAVIL, A. E.; De SOUZA, L. A. R. *An Implementation of a Power System Smart Waveform Recorder using FPGA and ARM cores*. Measurement, London Print, 2016.
- [77] KAPISCH, E. B.; SILVA, L. R. M.; CERQUEIRA, A. S.; ANDRADE FILHO, L. M.; DUQUE, C. A.; RIBEIRO, P. F. *A Gapless Waveform Recorder for Monitoring Smart Grids*. 17th International Conference on Harmonics and Quality of Power (ICHQP), pp.130-136, 2016.
- [78] KAPISCH, E. B.; SILVA, L. R. M.; MARTINS, C. H. N.; BARBOSA, A. S.; DUQUE, C. A.; ANDRADE FILHO, L. M.; CERQUEIRA, A. S. *An Electrical Signal Disturbance Detector and Compressor Based on FPGA Platform*. 16th International Conference on Harmonics and Quality of Power (ICHQP), pp. 278-282, 2014.
- [79] MARTINS, C. H.; MONTEIRO, H. L. M.; OLIVEIRA, M. M.; SILVA; L. R. M.; DUQUE, C. A.; RIBEIRO, P. F. *A Virtual Instrument for Time Varying Harmonic Analysis*. IEEE Power and Energy Society General Meeting (PESGM), pp. 1-5, 2016.

- [80] OLIVEIRA, M. M.; SILVA L. R. M.; DUQUE, C. A.; ANDRADE FILHO L. M.; RIVEIRO, P. F. *Implementation of an Electrical Signal Compression System Using Sparce Representation*. 18th International Conference on Harmonics and Quality of Power (ICHQP), pp. 1-5, 2018.
- [81] LEVINE, J. *Flex & Bison: Text Processing Tools*. O'Reilly Media, Inc., 2009.
- [82] CILETTI, M. D. *The Art of Assembly Language*. No Starch Press, 2003.
- [83] THOMAS, D.; MOORBY, P. *The Verilog Hardware Description Language*. Springer Science & Business Media, 2008.
- [84] OLIPHANT, T. E. *Python for Scientific Computing*. Computing in Science & Engineering, v. 9, n. 3, pp. 10–20, 2007.
- [85] Intel Corporation. *Intel FPGAs and Programmable Devices*. Disponível em: <https://www.intel.com.br/content/www/br/pt/products/programmable.html>
- [86] AGUIAR, M. S.; ANDRADE FILHO, L. M.; SEIXAS, J. M. *Processamento Embocado para Implementação de um Método Iterativo de Deconvolução Visando a Reconstrução de Energia em Calorímetros de Altas Energias*. XXXVII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, 2019, Petrópolis, RJ. SBrT, 2019.
- [87] AGUIAR, M. S.; ANDRADE FILHO, L. M.; SEIXAS, J. M. *Processamento Embocado em FPGA para Implementação de Métodos Iterativos de Desconvolução no Sistema de Trigger do ATLAS*. XL Encontro Nacional de Física de Partículas e Campos (ENFPC) e XLII Reunião de Trabalho sobre Física Nuclear no Brasil (RTFNB), 2019, Campos do Jordão, SP. XL ENFPC e XLII RTFNB, 2019.
- [88] AGUIAR, M. S.; RESENDE, M. O.; VICCINI, L. O. F.; DIAS, K.; TEIXEIRA, T. A.; ANDRADE FILHO, L. M.; SEIXAS, J. M. *Arquitetura Multi-Core de Processadores Reconfiguráveis para Reconstrução Online de Energia no Calorímetro Hadrônico do ATLAS*. XXIII Congresso Brasileiro de Automática (a ser publicado), CBA, 2020.
- [89] BAER, J. L. *Arquitetura de Microprocessadores - Do Simples Pipeline ao Multiprocessador em Chip*. LTC, 2013.
- [90] AGUIAR, M. S.; VICCINI, L. O. F.; SANTOS, D. F.; Resende, M. O.; FARIA, M.; ANDRADE FILHO, L. M.; SEIXAS, J. M. *Processamento Multicore para Reconstrução Online de Energia por meio de Redes Neurais*. XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (a ser publicado), SBrT, 2020.

## APÊNDICE A – Produção Bibliográfica

### Publicações em Congressos Nacionais

AGUIAR, M. S.; ANDRADE FILHO, L. M. ; SEIXAS, J. M. “*Processamento Embarcado para Implementação de um Método Iterativo de Deconvolução Visando a Reconstrução de Energia em Calorímetros de Altas Energias*”. XXXVII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, 2019, Petrópolis, RJ. SBrT, 2019.

**Resumo:** Este trabalho propõe a implementação, em FPGA, de um processador customizado que opera um método iterativo, baseado em Gradiente Descendente Positivo, visando sua aplicação na reconstrução online de energia em calorímetros de altas energias, respeitando a latência de operação necessária. Este algoritmo é mais eficiente na reconstrução dos sinais que o atual método em uso, o qual não é tolerante ao efeito de empilhamento de sinais que ocorre em colisionadores de partículas modernos, que operam em elevada colimação dos feixes de colisão.

AGUIAR, M. S.; RESENDE, M. O. ; TEIXEIRA, T. ; ANDRADE FILHO, L. M.; SEIXAS, J. M. “*Implementação em FPGA de um Método Iterativo de Deconvolução para Operar no Sistema de Trigger do Experimento ATLAS*”. XXII Encontro Nacional de Modelagem Computacional e X Encontro de Ciências e Tecnologia de Materiais, 2019, Juiz de Fora, MG. Anais do XXII ENMC X ECTM, 2019.

**Resumo:** Aceleradores de partículas, como o LHC, em Genebra, na Suíça, colidem partículas subatômicas cujos subprodutos (partículas fundamentais da natureza) são analisados por detectores ao redor do ponto de colisão. O LHC vem passando por um processo de atualização, em que parâmetros como a energia das colisões e a quantidade de partículas por *bunch* são aumentados, impactando diretamente nos sistemas de instrumentação dos detectores, como, por exemplo, o Experimento ATLAS presente neste acelerador. Este aumento na probabilidade de ocorrência de colisões adjacentes produz um efeito conhecido como *pile-up* (empilhamento de sinais) na eletrônica de leitura dos sub-detectores do ATLAS. Para lidar com este novo desafio, métodos iterativos de deconvolução de sinais, baseados em Representação Esparsa de Dados, como o SSF, vêm sendo propostos, apresentando resultados satisfatórios. No entanto, tais métodos apresentam um alto custo computacional, de modo que o principal desafio atual é o desenvolvimento de algoritmos capazes de operar de forma online. Este trabalho apresenta um aprimoramento proposto ao SSF, permitindo reduzir a quantidade de iterações deste algoritmo em uma ordem de grandeza, facilitando, assim, a sua implementação online.

AGUIAR, M. S.; ANDRADE FILHO, L. M. ; SEIXAS, J. M. “*Processamento Embocado em FPGA para Implementação de Métodos Iterativos de Desconvolução no Sistema de Trigger do ATLAS*”. XL Encontro Nacional de Física de Partículas e Campos (ENFPC) e XLII Reunião de Trabalho sobre Física Nuclear no Brasil (RTFNB), 2019, Campos do Jordão, SP. XL ENFPC e XLII RTFNB, 2019.

**Resumo:** O LHC (*Large Hadron Collider*) é o principal acelerador de partículas do CERN, sendo o maior e mais energético acelerador de partículas já construído. A aquisição dos dados resultantes das colisões no LHC é realizada pelos seus detectores. Dentro eles, o ATLAS (*A Toroidal LHC Apparatus*), que é composto por sub-dectores dispostos em camadas, sendo cada uma responsável por medir propriedades específicas das partículas geradas pelas colisões. O Calorímetro Hadrônico do ATLAS (*TileCal*) opera a uma alta taxa de eventos e a estimativa de energia sofre com a sobreposição de sinais em seus canais de leitura, uma vez que o tempo de resposta (largura do pulso de  $150\text{ ns}$ ) é maior que o período de colisão ( $25\text{ ns}$ ). Este fenômeno é conhecido como empilhamento de sinais (*pileup*) e a ocorrência do efeito se intensificará à medida que a luminosidade das colisões do LHC aumenta com as constantes atualizações que vêm sendo feitas. O sistema atual de processamento de sinais do detector não está preparado para lidar com o *pileup*, o que pode ocasionar baixa eficiência para a detecção (*trigger*) de eventos. Neste contexto, para lidar com o problema obtendo mais eficiência na reconstrução dos sinais, foi proposta uma abordagem de separação dos sinais empilhados usando métodos de desconvolução, muito empregados em comunicação digital de dados. Neste trabalho, a proposta é implementar em FPGA (*Field-Programmable Gate Array*) um processador customizado que opera um método iterativo de desconvolução baseado em Gradiente Descendente, realizando o estudo da viabilidade de sua aplicação na reconstrução de energia no Calorímetro Hadrônico do ATLAS, tendo baixo custo em elementos lógicos e melhor eficiência que os métodos baseados em filtros FIR. Dada esta validação da topologia, a proposta para trabalhos futuros é modificar a estrutura da Unidade Lógica Aritmética (ULA) do processador de forma a realizar os cálculos matriciais dentro do tempo requerido pelo sistema de aquisição do ATLAS.

AGUIAR, M. S.; RESENDE, M. O.; VICCINI, L. O. F. ; DIAS, K.; TEIXEIRA, T. A.; ANDRADE FILHO, L. M.; SEIXAS, J. M. “*Arquitetura Multi-Core de Processadores Reconfiguráveis para Reconstrução Online de Energia no Calorímetro Hadrônico do ATLAS.*” In: XXIII Congresso Brasileiro de Automática, CBA, novembro, 2020.

**Resumo:** Calorímetros são sistemas usados para medir a energia de partículas fundamentais que atravessam o seu material. Para tal, pulsos elétricos são gerados por sensores posicionados ao longo do material absorvedor do calorímetro. Técnicas de processamento digital de sinais são empregadas para detectar e estimar parâmetros destes pulsos de forma a inferir a energia das partículas. Tais técnicas, quando implementadas online, necessitam ser de baixa complexidade para que seja possível a sua implementação em *hardware* dedicado. No entanto, técnicas baseadas em teoria de Representação Esparsa (RE) de dados vêm se destacando quanto à eficiência de reconstrução, mas, devido ao seu alto custo computacional, ainda não são utilizadas como uma opção para processamento online. Neste trabalho, é apresentada a customização de um processador e sua utilização em uma arquitetura *multi-core* em FPGA, possibilitando a implementação online de técnicas baseadas em RE. Para demonstrar seu funcionamento, foi utilizado o calorímetro hadrônico do Experimento ATLAS como ambiente de aplicação.

AGUIAR, M. S.; VICCINI, L. O. F. ; SANTOS, D. F. ; RESENDE, M. O. ; FARIA, M. ; ANDRADE FILHO, L. M. ; SEIXAS, J. M. “*Processamento Multicore para Reconstrução Online de Energia por meio de Redes Neurais.*” In: XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, SBrT, novembro, 2020.

**Resumo:** A reconstrução de energia em calorímetros é um processamento executado em pulsos gerados em seus eletrodos para estimar a energia de partículas subatômicas que atravessam seu material. Características não-gaussianas do ruído de calorímetros operando em altas taxas de evento demonstram que métodos não-lineares de estimação são mais indicados. No entanto, tais métodos tendem a ter um alto custo computacional, o que dificulta sua implementação online. Apresenta-se, com este trabalho, a implementação de uma rede neural embarcada em um processamento *multicore* em FPGA capaz de operar a uma taxa de aquisição acima de 40 MHz, bem como uma implementação no Calorímetro do Experimento ATLAS.

TEIXEIRA, T. A.; AGUIAR, M. S.; ANDRADE FILHO, L. M. ; SEIXAS, J. M. “Método Iterativo de Representação Esparsa Implementado em FPGA para Aplicação em Calorimetria.” In: XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, SBrT, novembro, 2020.

**Resumo:** O acelerador de partículas LHC vem passando por um processo de atualização, produzindo o efeito conhecido como *pile-up* (empilhamento de sinais) na eletrônica de leitura dos detectores. Como o algoritmo atualmente utilizado para a estimativa da amplitude dos sinais gerados nessas colisões não é tolerante a esse efeito, este trabalho propõe a implementação em *hardware* de um método iterativo baseado em uma variante do método *Separable Surrogate Functionals* que recupera a informação da amplitude dos sinais sobrepostos dentro de uma janela de aquisição. Tal implementação permite que dezenas de canais possam ser implementados em paralelo dentro de uma única FPGA, além de respeitar a latência de operação necessária a sua implementação *online*.

## ANEXO A – Algoritmos Implementados no SAPHO

Nesse anexo serão descritos os algoritmos que contém os passos seguidos, de forma simplificada, para cada implementação que foi realizada no SAPHO, em linguagem  $C^+$ .

Para o melhor entendimento desta etapa, nas Figuras 50, 51, 52, 53 e 54 podem ser observadas as equações onde está destacada a ordem de procedência a qual cada operação matricial foi calculada, nos respectivos algoritmos, de forma que as operações foram realizadas na direção dos blocos mais internos para os mais externos, ou seja, primeiro os de cor azul, seguidos pela cor vermelha, depois a cor roxo e, por fim, os de cor verde.

Vale ressaltar que devem ser adicionados zeros ao sinal que contém os dados de saída de forma a padronizá-lo, mais especificamente, são adicionados três zeros no início da janela de saída e quatro zeros no final da mesma, de forma que, ao observar o sinal de saída contendo diversas janelas em sequência, ocorrem 48 intervalos temporais de 25 ns com colisões e 7 intervalos temporais de 25 ns sem colisões, de acordo com o padrão de colisões mais recente adotado pelo LHC, conhecido como 48b7e [65]. Com isso, as janelas de saída e de entrada de dados possuem o mesmo tamanho (55 termos).

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu \mathbf{H}^T (\mathbf{r} - \mathbf{Hx}_i)$$

Figura 50 – Operações do Algoritmo 1.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu \mathbf{H}^T \mathbf{r} - \mu \mathbf{H}^T \mathbf{Hx}_i$$

Figura 51 – Operações do Algoritmo 2.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu (\mathbf{Hs} - \mathbf{Ax}_i)$$

Figura 52 – Operações do Algoritmo 3.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu (\mathbf{Hs} - \mathbf{Ax}_i)$$

Figura 53 – Operações do Algoritmo 4.

$$\begin{aligned} \mathbf{x}_i &= \mathbf{B}^T \mathbf{x}_i \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mu (\mathbf{Hs} - \mathbf{Ax}_i) \end{aligned}$$

Figura 54 – Operações do Algoritmo 5.

---

**Algoritmo 1:** Implementação em ponto flutuante (versão 1.0).
 

---

```

1 int i;
2 int mi = 0.25;
3 float x(48);
4 float aux(48);
5 float y(55);
6 float d(55);
7 float h(55,48) = load(matrizH);
8 float mih(55,48);
9 for (i = 0:1:47) do
10   | x(i) = 0;
11 end
12 for (i = 0:1:54) do
13   | d(i) = in(0);
14   %% vetor d recebeu os dados de entrada vindos da porta de índice 0.
15 end
16 for (i = 0:1:149) do
17   for (i = 0:1:54) do
18     | y(i) = 0;
19   end
20   for (i = 0:1:47) do
21     | aux(i) = 0;
22   end
23   mih = mi * hT;
24   y = d - h * x;
25   aux = mih * y;
26   x = x + aux;
27   for (i = 0:1:47) do
28     | if (x(i) < 0) then
29       | | x(i) = 0;
30     end
31   end
32 end
33 %% abaixo está a janela de saída, no formato padrão, na porta de índice 0.
34 out(0, 0);
35 out(0, 0);
36 out(0, 0);
37 for (i = 0:1:47) do
38   | out(0, x(i));
39 end
40 out(0, 0);
41 out(0, 0);
42 out(0, 0);
43 out(0, 0);

```

---

---

**Algoritmo 2:** Implementação em ponto flutuante (versão 1.1).
 

---

```

44 int i;
45 float mi = 0.25;
46 float x(48);
47 float aux(48);
48 float mihd(48);
49 float y(55);
50 float d(55);
51 float h(55,48) = load(matrizH);
52 float mih(55,48);
53 for (i = 0:1:47) do
54   | x(i) = 0;
55 end
56 for (i = 0:1:54) do
57   | d(i) = in(0);
58   | %% vetor d recebeu os dados de entrada vindos da porta de índice 0.
59 end
60 mih = mi * hT;
61 mihd = mih * d;
62 for (i = 0:1:149) do
63   for (i = 0:1:54) do
64     | y(i) = 0;
65   end
66   for (i = 0:1:47) do
67     | aux(i) = 0;
68   end
69   y = h * x;
70   y = d - y;
71   aux = mih * y;
72   x = x + mihd - aux;
73   for (i = 0:1:47) do
74     if (x(i) < 0) then
75       | x(i) = 0;
76     end
77   end
78 end
79 %% abaixo está a janela de saída, no formato padrão, na porta de índice 0.
80 out(0, 0);
81 out(0, 0);
82 out(0, 0);
83 for (i = 0:1:47) do
84   | out(0, x(i));
85 end
86 out(0, 0);
87 out(0, 0);
88 out(0, 0);
89 out(0, 0);

```

---

---

**Algoritmo 3:** Implementação em ponto flutuante (versões 2.0 e 2.1).

---

```

90 int i;
91 float mi = 0.25;
92 float Hs(48);
93 float x(48);
94 float aux(48);
95 float d(55);
96 float A(48,48) = load(matrixA);
97 %% a matriz A tem 13 termos não nulos em cada coluna.
98 float H(55,48) = load(matrizH);
99 for (i = 0:1:47) do
100   | x(i) = 0;
101 end
102 for (i = 0:1:47) do
103   | aux(i) = 0;
104 end
105 for (i = 0:1:54) do
106   | d(i) = in(0);
107   | %% vetor d recebeu os dados de entrada vindos da porta de índice 0.
108 end
109 Hs = HT * d;
110 for (i = 0:1:149) do
111   %% o produto abaixo foi declarado/realizado termo a termo, ignorando as
      %% multiplicações por zero. Na versão 2.0 da implementação os termos não nulos
      %% de A são escritos como variáveis. Na versão 2.1 eles são escritos como
      %% constantes, ou seja, seus valores literais.
112   aux = A * x;
113   aux = Hs - aux;
114   x = x + mi * aux;
115   for (i = 0:1:47) do
116     | if (x(i) < 0) then
117     |   | x(i) = 0;
118     | end
119   end
120 end
121 %% abaixo está a janela de saída, no formato padrão, na porta de índice 0.
122 out(0, 0);
123 out(0, 0);
124 out(0, 0);
125 for (i = 0:1:47) do
126   | out(0, x(i));
127 end
128 out(0, 0);
129 out(0, 0);
130 out(0, 0);
131 out(0, 0);

```

---

---

**Algoritmo 4:** Implementação em ponto flutuante (versão 2.2).

---

```

132 int i;
133 float Hs0, Hs1, Hs2, ... Hs46, Hs47;
134 float x0 = 0, x1 = 0, x2 = 0, ... x46 = 0, x47 = 0;
135 %% o sinal x é declarado como variaveis e inicializado com zeros termo a termo ao
   ser declarado.
136 float d0 = 0, d1 = 0, d2 = 0, ... d53 = 0, d54 = 0;
137 %% Para H, A e B abaixo, apenas os termos não nulos serão utilizados nas operações
   e os mesmos são escritos termo a termo de forma literal para todos os cálculos.
138 d0 = in(0);
139 d1 = in(0);
140 ...
141 d53 = in(0);
142 d54 = in(0);
143 %% variaveis d recebem os dados de entrada vindos da porta de índice 0.
144 Hs = HT * d;
145 %% no SAPHO, essa operação do calculo de Hs é escrita termo a termo e os valores
   de HT são inseridos diretamente, como constantes.
146 for (i = 0:1:149) do
147   aux = x + 0.25*(Hs - A*x);
148   %% no SAPHO, essa operação do calculo de aux é escrita termo a termo e os
      valores de  $\mu$ , Hs e A são inseridos diretamente, como constantes.
149   if (x0 < 0) then
150     | x0 = 0;
151   else x0 = aux0;
152 end
153   if (x1 < 0) then
154     | x1 = 0;
155   else x1 = aux1;
156 end
157 ...
158   if (x47 < 0) then
159     | x47 = 0;
160   else x47 = aux47;
161 end
162 end
163 %% abaixo está a janela de saída, no formato padrão, na porta de índice 0.
164 out(0, 0);
165 out(0, 0);
166 out(0, 0);
167 out(0, x0);
168 ...
169 out(0, x47);
170 out(0, 0);
171 out(0, 0);
172 out(0, 0);
173 out(0, 0);

```

---

---

**Algoritmo 5:** Implementação em ponto flutuante (versões 3.0 e 3.1).

---

```

174 int i;
175 float Hs0, Hs1, Hs2, ... Hs46, Hs47;
176 float x0 = 0, x1 = 0, x2 = 0, ... x46 = 0, x47 = 0;
177 %% a função PSET (ou @) é responsável por zerar os termos negativos. A diferença
entre as versões 3.0 e 3.1 é que a 3.1 tem o PSET implementado, a 3.0 não tem.
178 float d0 = 0, d1 = 0, d2 = 0, ... d53 = 0, d54 = 0;
179 d0 = in(0);
180 d1 = in(0);
181 ...
182 d53 = in(0);
183 d54 = in(0);
184 %% variaveis d recebem os dados de entrada vindos da porta de índice 0.
185 Hs = HT * d;
186 %% os valores de  $H^T$  são inseridos diretamente, como constantes.
187 x0 = d3;
188 x1 = d4;
189 ...
190 x46 = d49;
191 x47 = d50;
192 aux = BT*x;
193 %% esta operação do calculo de aux é realizada termo a termo, e os valores de  $B^T$ 
são inseridos diretamente, como constantes.
194 x0 @ aux0;
195 ...
196 x47 @ aux47;
197 for (i = 0:1:17) do
198     aux = x + 0.25*(Hs - A*x); %% esta operação do calculo de aux é realizada
termo a termo, e os valores de  $Hs$  e  $A$  são inseridos diretamente, como
constantes.
199     x0 @ aux0;
200     ...
201     x47 @ aux47;
202 end
203 %% abaixo está a janela de saída, no formato padrão, na porta de índice 0.
204 out(0, 0);
205 out(0, 0);
206 out(0, 0);
207 out(0, x0);
208 out(0, x1);
209 ...
210 out(0, x46);
211 out(0, x47);
212 out(0, 0);
213 out(0, 0);
214 out(0, 0);
215 out(0, 0);

```

---

## ANEXO B – Características das Implementações Realizadas

Neste anexo estão descritas, de forma resumida, as principais características dos algoritmos de cada versão que foi implementada do método SSF no presente trabalho.

### 1. Implementação em Ponto Flutuante - Versão 1.0:

- Realizada de forma genérica.
- Todos os termos de  $\mathbf{H}$  declarados como variáveis.
- O sinal  $\mathbf{x}$  é declarado como um vetor.
- Todas operações são realizadas dentro de um *loop* de 150 iterações.

### 2. Implementação em Ponto Flutuante - Versão 1.1:

- Realizada a operação distributiva na equação.
- Todos os termos de  $\mathbf{H}$  declarados como variáveis.
- O sinal  $\mathbf{x}$  é declarado como um vetor.
- O produto  $\mu\mathbf{H}^T$  é calculado só uma vez e está fora do *loop*.

### 3. Implementação em Ponto Flutuante - Versão 2.0:

- Os termos nulos de  $\mathbf{H}$  e de  $\mathbf{A}$  passam a ser descartados.
- Os termos não nulos de  $\mathbf{H}$  e de  $\mathbf{A}$  são declarados como variáveis.
- O sinal  $\mathbf{x}$  é declarado como um vetor.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T\mathbf{r}$  é calculada só uma vez antes de o *loop* começar.

### 4. Implementação em Ponto Flutuante - Versão 2.1:

- Os termos nulos de  $\mathbf{H}$  e de  $\mathbf{A}$  são descartados.
- Os termos não nulos de  $\mathbf{H}$  e de  $\mathbf{A}$  passam a ser escritos diretamente no código como constantes.
- O sinal  $\mathbf{x}$  é declarado como um vetor.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T\mathbf{r}$  é calculada uma vez antes de o *loop* começar.

### 5. Implementação em Ponto Flutuante - Versão 2.2:

- O sinal  $\mathbf{x}$  passa a ser representado por 48 variáveis.
- Todos os termos constantes são escritos diretamente no código.
- Sinal de subtração colocado em evidência (contantes positivas).
- A matriz  $\mathbf{Hs} = \mathbf{H}^T\mathbf{r}$  é calculada uma vez antes de o *loop* começar.

6. Implementação em Ponto Flutuante - Versão 3.0:

- Implementação da inicialização otimizada de  $\mathbf{x}$ .
- O *loop* passa a ter 18 iterações.
- Todos os termos constantes são escritos diretamente no código.
- Aplicação do patamar de corte para anular termos próximos de zero.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T \mathbf{r}$  e a inicialização de  $\mathbf{x}$  são calculados antes do *loop*.

7. Implementação em Ponto Flutuante - Versão 3.1:

- Implementação da inicialização otimizada de  $\mathbf{x}$ .
- Constantes escritas diretamente no código e patamar de corte aplicado.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T \mathbf{r}$  e a inicialização de  $\mathbf{x}$  são calculados antes do *loop*.
- Uso da nova instrução PSET.

8. Implementação em Ponto Fixo - Primeira Versão:

- Implementação da inicialização otimizada de  $\mathbf{x}$ .
- Constantes escritas diretamente no código e patamar de corte aplicado.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T \mathbf{r}$  e a inicialização de  $\mathbf{x}$  são calculados antes do *loop*.
- Uso da nova instrução PSET.
- Foi aplicado um ganho de  $2^7$ .

9. Implementação em Ponto Fixo - Versão Final:

- Implementação da inicialização otimizada de  $\mathbf{x}$ .
- Constantes escritas diretamente no código e patamar de corte aplicado.
- A matriz  $\mathbf{Hs} = \mathbf{H}^T \mathbf{r}$  e a inicialização de  $\mathbf{x}$  são calculados antes do *loop*.
- Uso das novas instruções PSET e NORM.
- Foi aplicado um ganho de  $2^7$ .

## ANEXO C – Tutorial: Criando um Projeto com o SAPHO

Neste tutorial são ilustrados os passos desde a criação de um projeto na interface de desenvolvimento do SAPHO até a configuração e visualização do processador no Quartus.

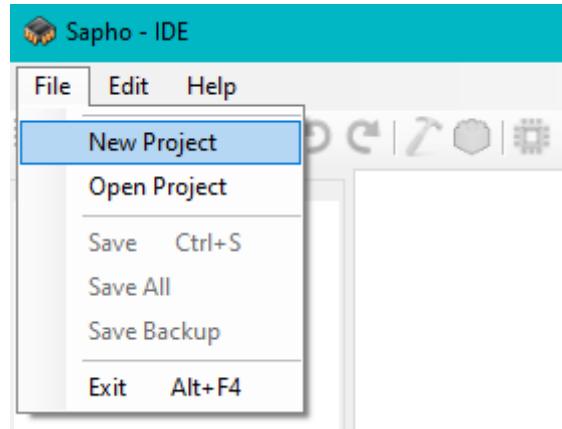


Figura 55 – Tutorial SAPHO: criando um novo projeto.

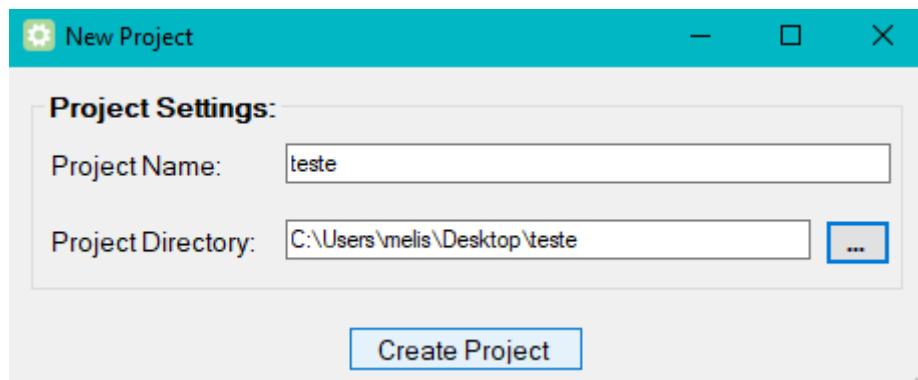


Figura 56 – Tutorial SAPHO: configurando o diretório do projeto.

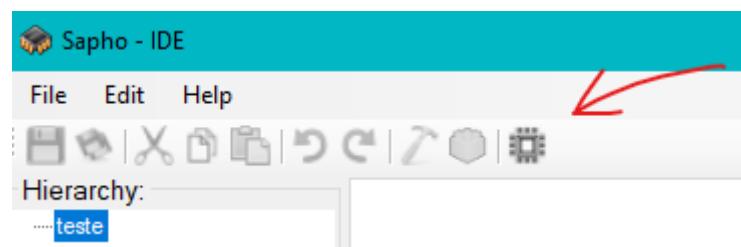


Figura 57 – Tutorial SAPHO: criando um processador.

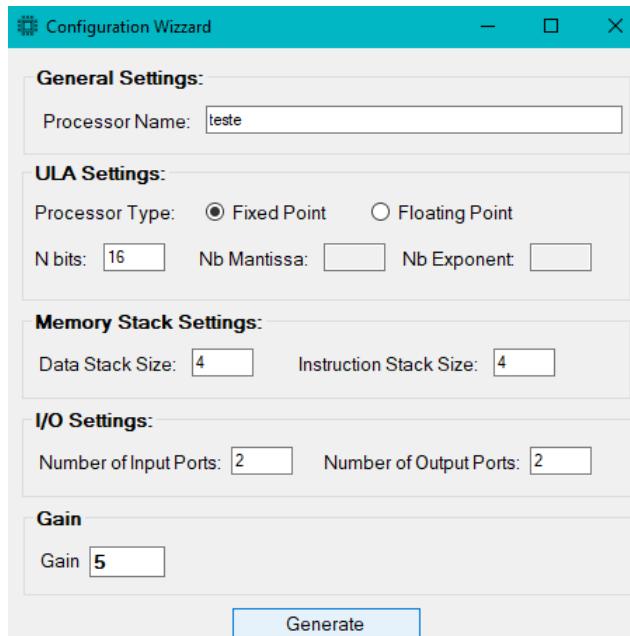


Figura 58 – Tutorial SAPHO: configurando os parâmetros do processador.

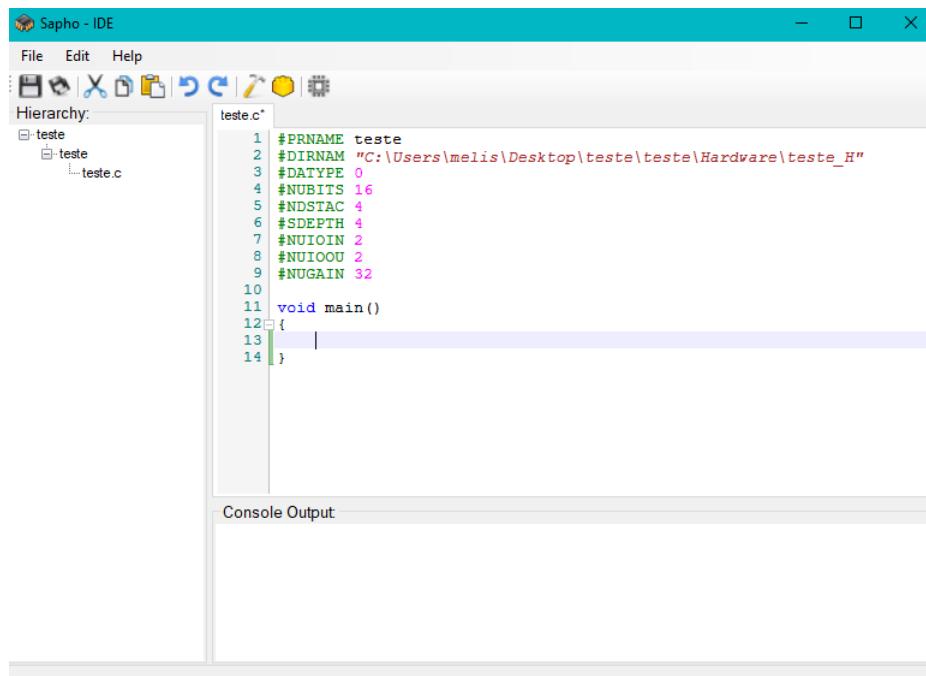
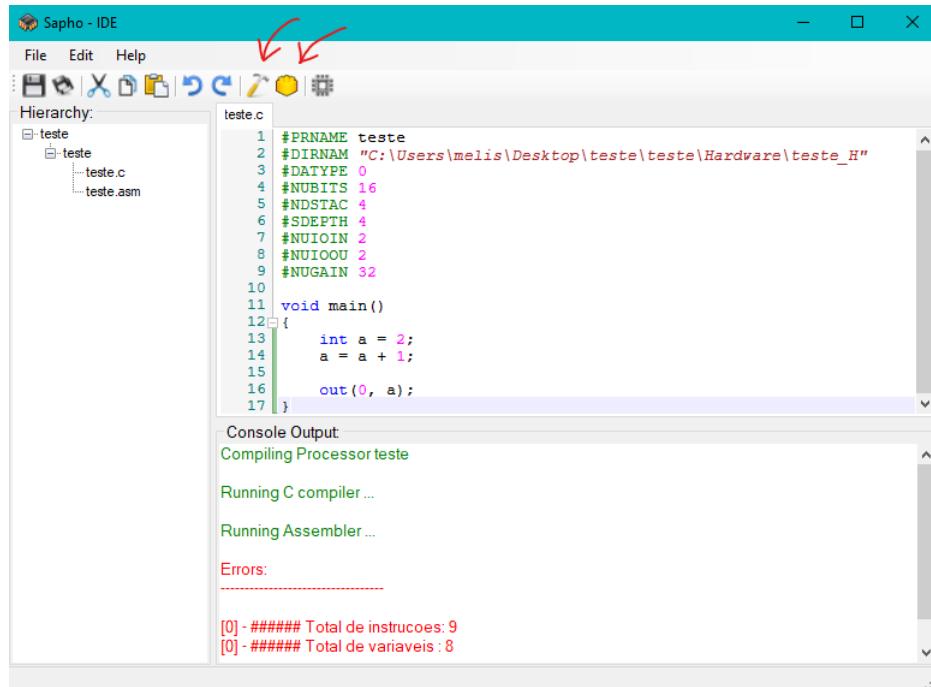


Figura 59 – Tutorial SAPHO: interface de desenvolvimento em C<sup>+</sup>.



The screenshot shows the SAPHO - IDE interface. The title bar says "Sapho - IDE". The menu bar includes "File", "Edit", and "Help". The toolbar has various icons for file operations. The "Hierarchy" panel shows a project structure with a "teste" folder containing "teste.c" and "teste.asm". The main code editor window displays the content of "teste.c". A red arrow points to the status bar at the bottom, which shows "Compiling Processor teste". Below the code editor is a "Console Output" pane with the message "Compiling Processor teste". The bottom pane shows the assembly output for "teste.asm".

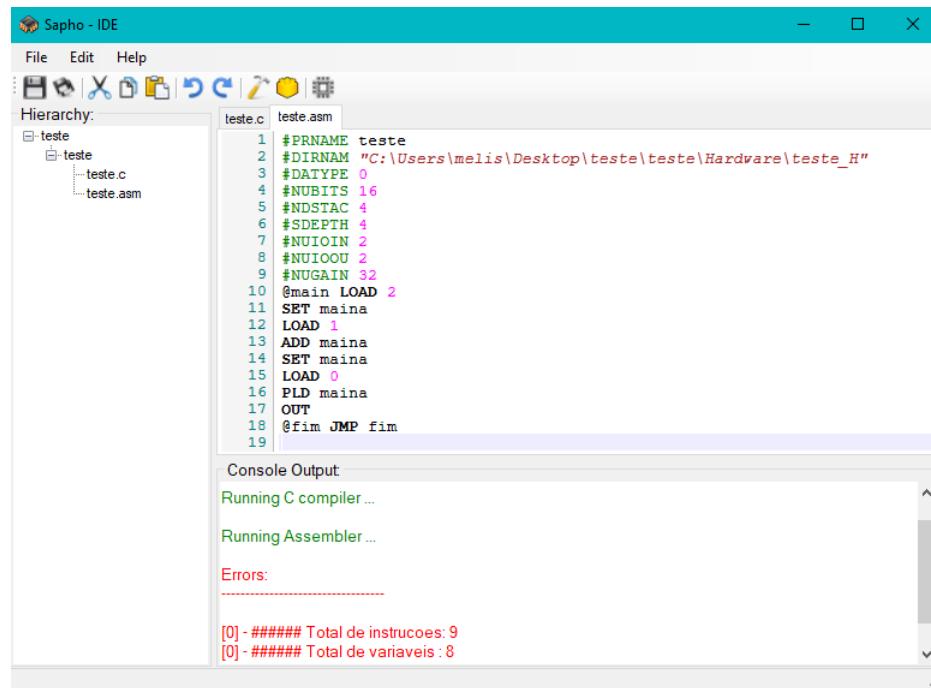
```

teste.c
1 #PRNAME teste
2 #DIRNAM "C:\Users\melis\Desktop\teste\teste\Hardware\teste_H"
3 #DATATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10
11 void main()
12 {
13     int a = 2;
14     a = a + 1;
15
16     out(0, a);
17 }

```

Console Output:  
Compiling Processor teste  
Running C compiler ...  
Running Assembler ...  
Errors:  
-----  
[0] ##### Total de instrucoes: 9  
[0] ##### Total de variaveis: 8

Figura 60 – Tutorial SAPHO: compilando o código.



The screenshot shows the SAPHO - IDE interface. The title bar says "Sapho - IDE". The menu bar includes "File", "Edit", and "Help". The toolbar has various icons for file operations. The "Hierarchy" panel shows a project structure with a "teste" folder containing "teste.c" and "teste.asm". The main code editor window displays the content of "teste.asm". The bottom pane shows the assembly output for "teste.asm".

```

teste.c teste.asm
1 #PRNAME teste
2 #DIRNAM "C:\Users\melis\Desktop\teste\teste\Hardware\teste_H"
3 #DATATYPE 0
4 #NUBITS 16
5 #NDSTAC 4
6 #SDEPTH 4
7 #NUIOIN 2
8 #NUIOOU 2
9 #NUGAIN 32
10 @main LOAD 2
11 SET maina
12 LOAD 1
13 ADD maina
14 SET maina
15 LOAD 0
16 PLD maina
17 OUT
18 @fim JMP fim
19

```

Console Output:  
Running C compiler ...  
Running Assembler ...  
Errors:  
-----  
[0] ##### Total de instrucoes: 9  
[0] ##### Total de variaveis: 8

Figura 61 – Tutorial SAPHO: código em *Assembly* gerado.

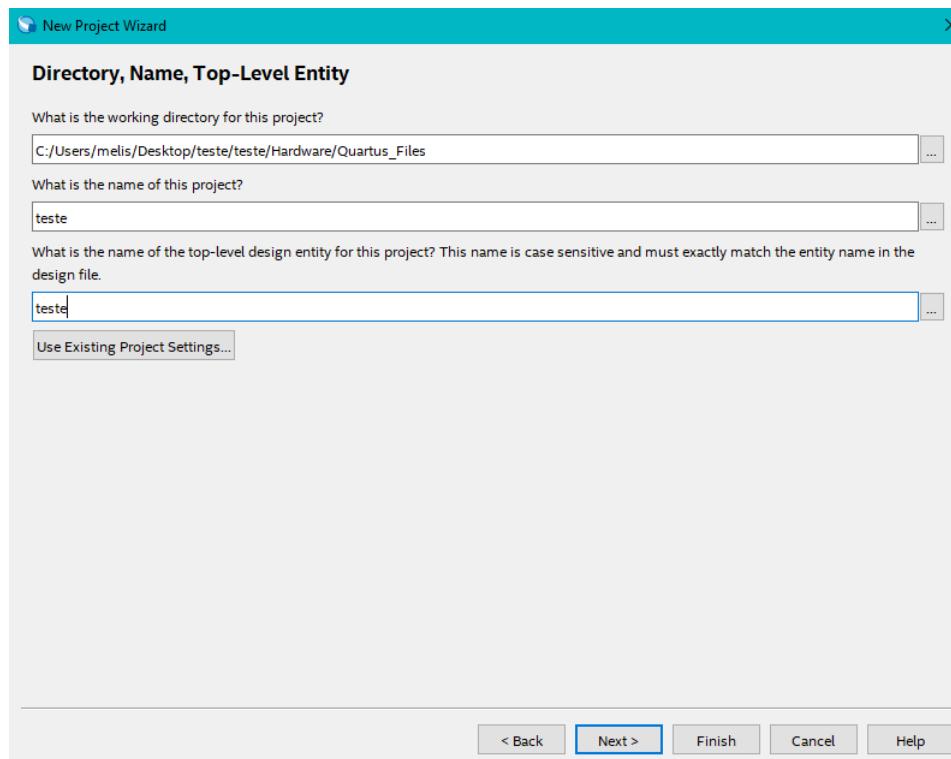


Figura 62 – Tutorial SAPHO: abrindo o projeto no Quartus.

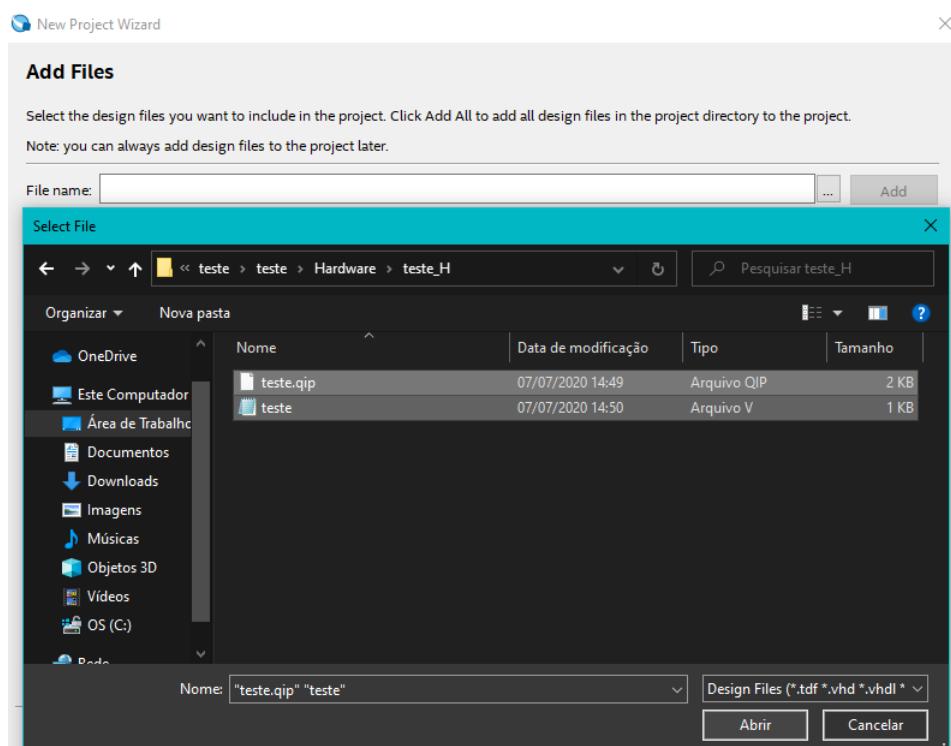


Figura 63 – Tutorial SAPHO: selecionando os arquivos de parametrização.

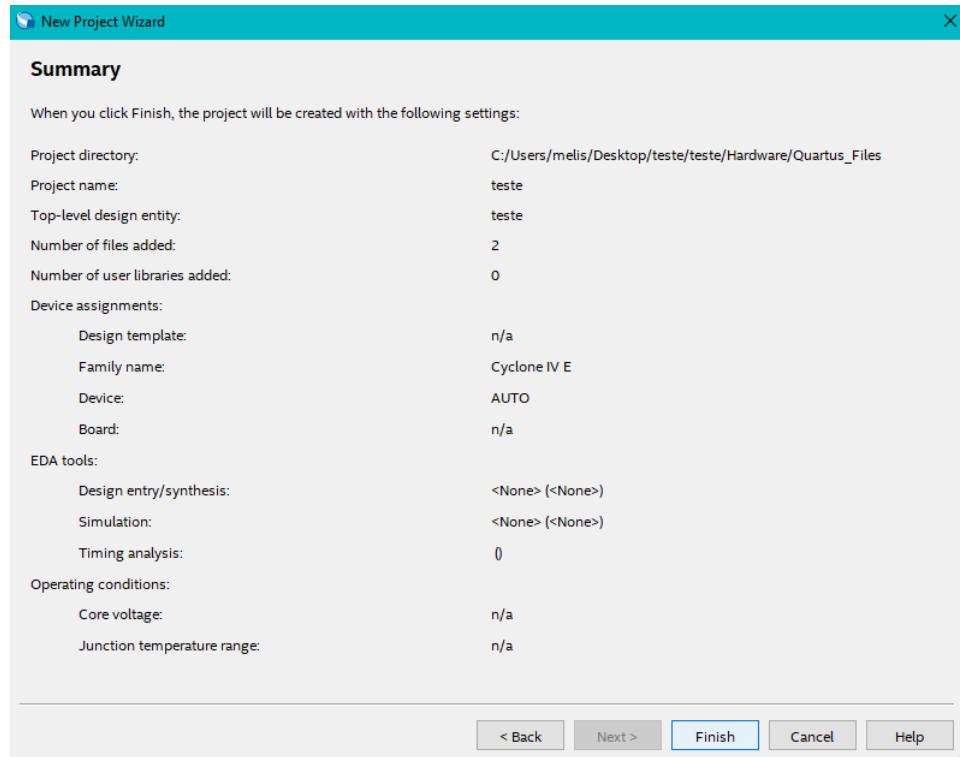


Figura 64 – Tutorial SAPHO: resumo do projeto criado no Quartus.

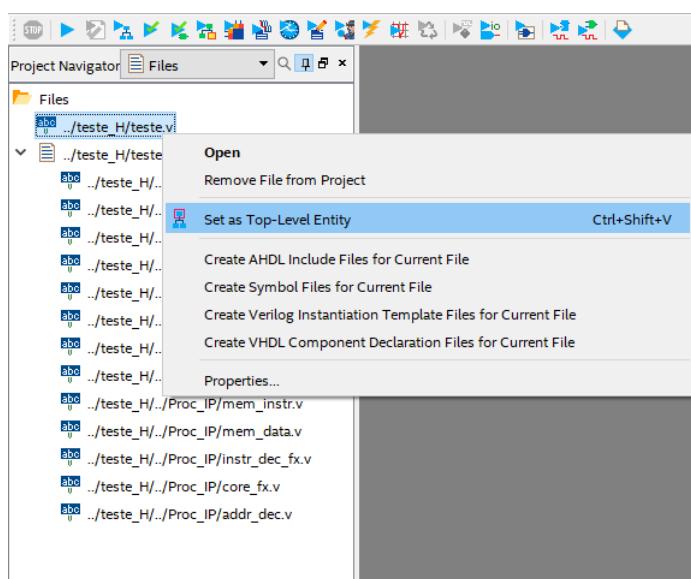
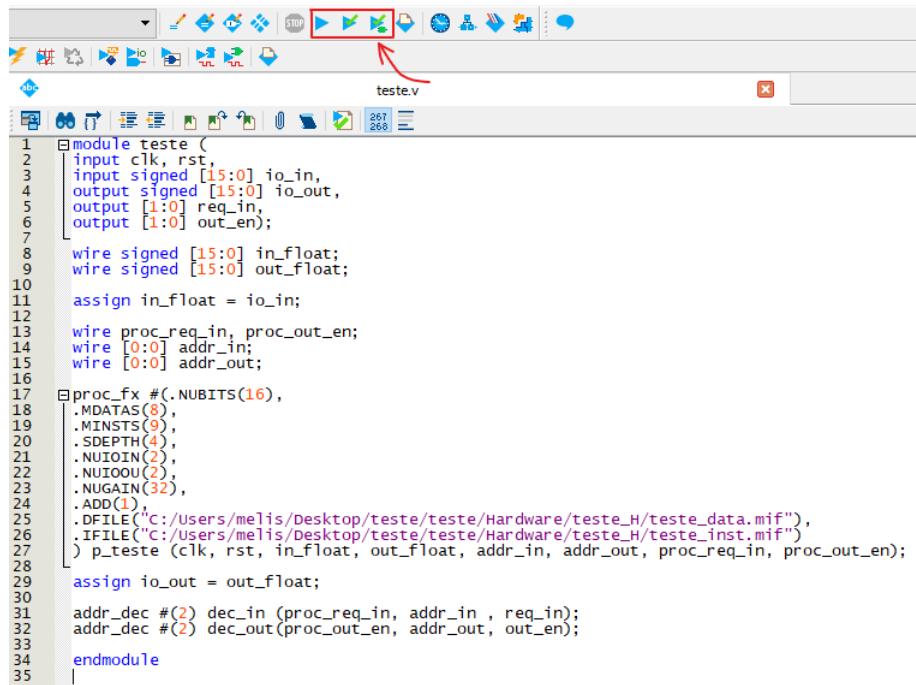


Figura 65 – Tutorial SAPHO: selecionando o arquivo principal.



```

1 module teste (
2     input clk, rst,
3     input signed [15:0] io_in,
4     output signed [15:0] io_out,
5     output [1:0] req_in,
6     output [1:0] out_en;
7
8     wire signed [15:0] in_float;
9     wire signed [15:0] out_float;
10    assign in_float = io_in;
11
12    wire proc_req_in, proc_out_en;
13    wire [0:0] addr_in;
14    wire [0:0] addr_out;
15
16    proc_fx #(NUBITS(16),
17        .MDATAS(8),
18        .MINSTS(9),
19        .SDEPTH(4),
20        .NUIOIN(2),
21        .NUIOOU(2),
22        .NUGAIN(32),
23        .ADD(1),
24        .DFILE("c:/users/melis/Desktop/teste/teste/Hardwate/teste_data.mif"),
25        .IFILE("C:/Users/melis/Desktop/teste/teste/Hardwate/teste_inst.mif")
26    ) p_teste (clk, rst, in_float, out_float, addr_in, addr_out, proc_req_in, proc_out_en);
27
28    assign io_out = out_float;
29
30    addr_dec #(2) dec_in (proc_req_in, addr_in, req_in);
31    addr_dec #(2) dec_out(proc_out_en, addr_out, out_en);
32
33 endmodule
34
35

```

Figura 66 – Tutorial SAPPHO: compilando o projeto.

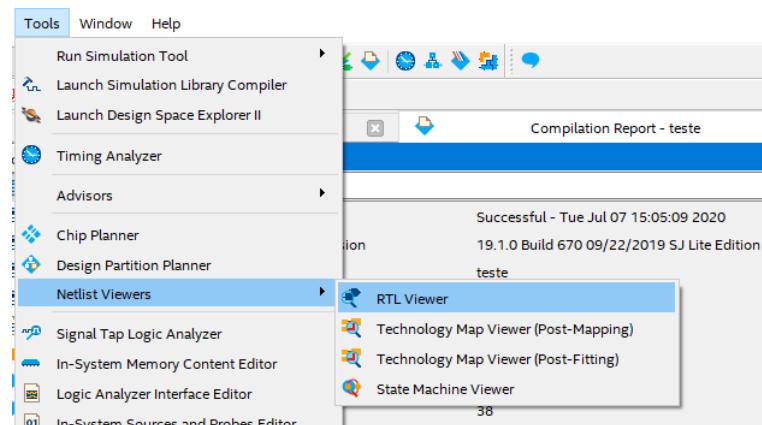


Figura 67 – Tutorial SAPPHO: visualizando o processador criado.

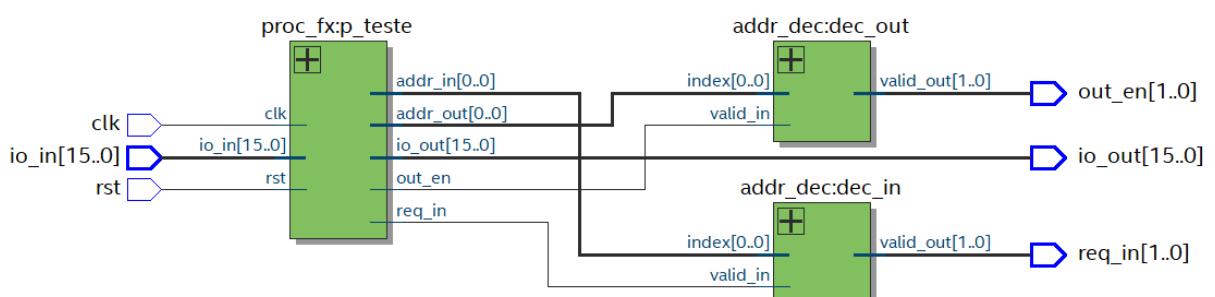


Figura 68 – Tutorial SAPPHO: diagrama do *top level* do processador desenvolvido.