UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Fábio Cardani Luna

Implementação e Avaliação de Métodos de Filtragem Digital em FPGA para Simulação de Pulsos em Calorimetria de Altas Energias

Juiz de Fora

2026

Fábio Cardani Luna

Implementação e Avaliação de Métodos de Filtragem Digital em FPGA para Simulação de Pulsos em Calorimetria de Altas Energias

Dissertação apresentada ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área: Sistemas Eletrônicos

Orientador: Prof. Dr. Luciano Manhães de Andrade Filho

Juiz de Fora

2026

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Cardani Luna, Fábio.
Implementação e Avaliação de Métodos de Filtragem Digital em FPGA para Simulação de Pulsos em Calorimetria de Altas Energias / Fábio Cardani Luna. – 2026.
121 f. : il.

Orientador: Luciano Manhães de Andrade Filho
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Pós Graduação em Engenharia Elétrica, 2026.

1. Simulador de pulsos. 2. Filtragem digital. 3. Alta taxa de eventos. I. Manhães de Andrade Filho, Luciano, orient. II. Título.

Fábio Cardani Luna

Implementação e Avaliação de Métodos de Filtragem Digital em FPGA para Simulação de Pulsos em Calorimetria de Altas Energias

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas Eletrônicos

Aprovada em 13 de março de 2026.

BANCA EXAMINADORA

Prof. Dr. Luciano Manhães de Andrade Filho - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Augusto Santiago Cerqueira
Universidade Federal de Juiz de Fora

Prof. Dr. Thiago Medeiros Carvalho
Universidade do Estado do Rio de Janeiro

Juiz de Fora, 24/02/2026.

Documento assinado eletronicamente por **Luciano Manhaes de Andrade Filho**, **Professor(a)**, em 13/03/2026, às 16:20, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Augusto Santiago Cerqueira**, **Professor(a)**, em 13/03/2026, às 16:21, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Thiago Medeiros Carvalho**, **Usuário Externo**, em 13/03/2026, às 16:29, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **2885983** e o código CRC **7D9F89AA**.

Dedico este trabalho à minha mãe.

AGRADECIMENTOS

Gostaria de agradecer, primeiramente, ao meu professor Luciano, que me deu a oportunidade de realizar um trabalho tão engrandecedor para a minha carreira profissional. Foi um professor/orientador dedicado e paciente, sempre disposto a ensinar e a caminhar comigo em cada etapa dos trabalhos que realizamos juntos. Agradeço também aos professores Bernardo Peralva e Thiago Paschoalin, por todo o suporte fornecido fornecido e pela amizade que criamos nos dias de CERN.

Agradeço à minha família por ser meu porto seguro, minha base, e por nunca soltar a minha mão nos momentos mais difíceis dessa trajetória acadêmica. Jociene, Leo, Letícia, Nathalia, Tatiana, Jardro, Norberto, Jesus, Paulo, Gustavo e todos aqueles que sempre me mostraram, na prática, o verdadeiro significado da palavra família. Agradeço também à Laura e ao Luís, que sempre me acolheram como filho, provando que, para ser família, não precisa ser de sangue.

Agradeço, em especial, à minha tia Mercês, a qual não consigo descrever em palavras o amor e a gratidão que eu sinto. Tenho certeza de que nada neste mundo que eu fizer será capaz de retribuir 1% do que um dia ela já fez por mim. Carrego isso comigo todos os dias. Gratidão eterna por tudo.

Agradeço aos meus amigos da graduação, Ariza, Victor, Álvaro, Giovana, Jhosef, e àqueles com quem tive um maior contato no mestrado, Saymon e Ricardo. Vocês foram fundamentais para fazer a primeira etapa ser concluída com sucesso e leveza, com todas as conversas sérias e as brincadeiras que vivemos juntos. Além, claro, do suporte nas disciplinas. Torço muito por vocês, de coração.

Aos meus amigos de vida, Pedro, Felipe, Matheus, Guilherme, Gabriel, Júlia, Douglas e Duda, agradeço por cada momento e cada cerveja que tomamos juntos. Essas pausas e essas risadas foram essenciais para deixar tudo mais leve quando parecia pesado demais. Um agradecimento especial ao Guarino, por ser o melhor irmão que a vida poderia me dar. Sem vocês, nada disso seria possível.

Aos meus amigos do CERN, Leandro, Catarina, Gabi, Leonardo, Dabson e Jonh, obrigado por cada conversa, apoio e risada. Estar longe de casa nem sempre é fácil, e ter vocês por perto fez toda a diferença para que essa fase fosse mais leve e cheia de bons momentos.

À Universidade Federal de Juiz de Fora (UFJF), agradeço por toda a estrutura que tornou esse projeto possível.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"The experiment is the ultimate judge of scientific truth."

Richard Feynman

RESUMO

O aumento significativo da taxa de eventos e, consequentemente, das condições de empilhamento de sinais previstas para a Fase II do *Large Hadron Collider* impõe novos desafios aos sistemas de leitura dos experimentos de física de altas energias, demandando soluções digitais capazes de oferecer maior robustez numérica, flexibilidade de projeto e compatibilidade com as restrições de recursos de hardware. Nesse contexto, o ponto central deste trabalho é o desenvolvimento de um simulador de pulsos digitais concebido considerando explicitamente as condições desafiadoras da Fase II, com o objetivo de reproduzir, de forma controlada e reprodutível, sinais representativos do regime de alta luminosidade e, assim, viabilizar uma análise sistemática e consistente do processamento digital sob efeitos de empilhamento e ruído. O estágio de conformação de sinais, parte da eletrônica de *front-end* do Calorímetro de Telhas (*TileCal*) do Experimento ATLAS, assume papel central na qualidade da reconstrução de energia e na mitigação dos efeitos do ruído eletrônico. Este trabalho também propõe a implementação e a avaliação comparativa de diferentes métodos de filtragem digital aplicados ao sinal do Calorímetro, com foco em arquiteturas realizáveis em FPGA e adequadas às exigências operacionais do cenário de alta luminosidade. Os métodos investigados incluem o filtro IIR convencional com coeficientes quantizados, o método *lattice* e uma implementação em ponto flutuante baseada na representação IEEE, cada um apresentando características distintas em termos de complexidade de hardware, sensibilidade à quantização e comportamento numérico. As arquiteturas foram descritas em nível de circuito digital e implementadas em FPGA, sendo analisadas quanto à estabilidade, ao erro introduzido pela quantização e ao consumo de recursos lógicos. Os resultados obtidos evidenciam que a implementação em ponto flutuante oferece maior precisão e robustez numérica, uma vez que elimina os efeitos associados à quantização dos coeficientes e dos estados internos, embora implique em maior complexidade e utilização de recursos. O método *lattice* mostrou-se mais tolerante à quantização quando comparado ao IIR convencional, apresentando comportamento mais previsível em termos de estabilidade, ao custo de um aumento moderado de lógica. Por sua vez, o filtro IIR convencional destacou-se pela simplicidade de implementação e menor consumo de recursos, configurando-se como uma alternativa viável em cenários com restrições severas de hardware. Conclui-se que não existe uma solução única que seja ótima para todas as aplicações, sendo a escolha do método de filtragem diretamente dependente dos requisitos de precisão, estabilidade e disponibilidade de recursos computacionais, e que os resultados apresentados fornecem subsídios relevantes para o projeto de sistemas de filtragem digital aplicados à eletrônica de leitura do TileCal em condições de alta taxa.

Palavras-chave: Processamento Digital de Sinais; Filtragem Digital; FPGA.

ABSTRACT

The significant increase in event rate and, consequently, in the signal pileup conditions expected for Phase II of the Large Hadron Collider imposes new challenges on the readout systems of high-energy physics experiments, demanding digital solutions capable of offering greater numerical robustness, design flexibility, and compatibility with hardware resource constraints. In this context, the central focus of this work is the development of a digital pulse simulator explicitly conceived for the challenging conditions of Phase II, with the aim of reproducing, in a controlled and reproducible manner, signals representative of the high-luminosity regime and thereby enabling a systematic and consistent analysis of digital processing under pileup and noise effects. The signal shaping stage, part of the front-end electronics of the Tile Calorimeter (TileCal) of the ATLAS Experiment, plays a central role in the quality of energy reconstruction and in the mitigation of electronic noise effects. This work also proposes the implementation and comparative evaluation of different digital filtering methods applied to the Calorimeter signal, with a focus on FPGA-realizable architectures suitable for the operational requirements of the high-luminosity scenario. The investigated methods include the conventional IIR filter with quantized coefficients, the lattice method, and a floating-point implementation based on the IEEE representation, each presenting distinct characteristics in terms of hardware complexity, quantization sensitivity, and numerical behavior. The architectures were described at the digital circuit level and implemented on FPGA, being analyzed with respect to stability, quantization-induced error, and logic resource consumption. The results obtained show that the floating-point implementation offers greater precision and numerical robustness, since it eliminates the effects associated with the quantization of coefficients and internal states, although it implies greater complexity and resource utilization. The lattice method proved more tolerant to quantization when compared to the conventional IIR, exhibiting more predictable behavior in terms of stability, at the cost of a moderate increase in logic usage. In turn, the conventional IIR filter stood out for its implementation simplicity and lower resource consumption, making it a viable alternative in scenarios with severe hardware constraints. It is concluded that there is no single solution that is optimal for all applications, and that the choice of filtering method is directly dependent on the requirements of precision, stability, and availability of computational resources; furthermore, the results presented provide relevant support for the design of digital filtering systems applied to the TileCal readout electronics under high-rate conditions.

Keywords: Digital Signal Processing; Digital Filtering; FPGA.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog-to-Digital Converter*
ALICE	*A Large Ion Collider Experiment*
ATCA	*Advanced Telecommunications Computing Architecture*
ATLAS	*A Toroidal LHC ApparatuS*
BIBO	*Bounded-Input Bounded-Output*
CDF	*Cumulative Distribution Function*
CERN	*Conseil Européen pour la Recherche Nucléaire*
CIS	*Charge Injection System*
CMS	*Compact Muon Solenoid*
DB	*Daughter Board*
DC	*Direct Current*
DSP	*Digital Signal Processor*
FENICS	*Front-End for the New Infrastructure with Calibration and Signal Shaping*
FIR	*Finite Impulse Response*
FPGA	*Field-Programmable Gate Array*
FWHM	*Full Width at Half Maximum*
GBT	*GigaBit Transceiver*
HDL	*Hardware Description Language*
HL-LHC	*High-Luminosity Large Hadron Collider*
HLT	*High-Level Trigger*
ID	*Inner Detector*
IEEE 754	*IEEE Standard for Floating-Point Arithmetic*
IIR	*Infinite Impulse Response*
ILANA	*Interface for the Laser system, for ATLAS New Architecture*
ISR	*Intersecting Storage Rings*
L1A	*Level-1 Accept*
LAr	*Liquid Argon*
LCG	*Linear Congruential Generator*
LED	*Light-Emitting Diode*
LEP	*Large Electron–Positron Collider*
LHC	*Large Hadron Collider*
LHCb	*Large Hadron Collider beauty*
Linac2	*Linear Accelerator 2*
LTI	*Linear Time-Invariant*
LUT	*Look-Up Table*
MB	*Minimum Bias*
MD	*Mini-Drawer*
MDT	*Monitored Drift Tubes*
MF	*Matched Filter*

MHz	*Megahertz*
MIF	*Memory Initialization File*
MS	*Muon Spectrometer*
OF	*Optimal Filtering*
OF2	*Optimal Filtering 2*
PLL	*Phase-Locked Loop*
PMT	*Photomultiplier Tube*
PPr	*PreProcessor*
PS	*Proton Synchrotron*
PSB	*Proton Synchrotron Booster*
ROD	*Read-Out Driver*
ROM	*Read-Only Memory*
ROS	*Read-Out System*
RPC	*Resistive Plate Chambers*
SC	*Synchrocyclotron*
SCT	*SemiConductor Tracker*
SOS	*Second-Order Sections*
SPS	*Super Proton Synchrotron*
TDAQ	*Trigger and Data Acquisition*
TGC	*Thin Gap Chambers*
TileCal	*Tile Calorimeter*
TRT	*Transition Radiation Tracker*
UA1	*Underground Area 1*
UA2	*Underground Area 2*
UNESCO	*United Nations Educational, Scientific and Cultural Organization*
USA15	*Underground Service Area 15*
μD	*Micro-Drawer*

LISTA DE SÍMBOLOS

$(1.F)$	Significando normalizado na representação IEEE 754
A	Amplitude do sinal digitalizado
A_k	Resíduo associado ao k-ésimo polo
a	Parâmetro multiplicativo do LCG
b_n	n-ésimo coeficiente do filtro FIR
C^{137}	Isótopo Césio-137
c	Velocidade da luz no vácuo; no LCG, constante aditiva
$\hat{c}$	Coeficiente quantizado em ponto fixo
$C_{ADC \to pC}$	Fator de conversão de ADC para pC
C_{Cs}	Constante de calibração de Césio
C_{Laser}	Constante de calibração do sistema Laser
C_{MB}	Constante de calibração do sistema Minimum Bias
$C_{pC \to GeV}$	Fator de conversão de ADC para pC
$\mathbf{C}$	Matriz de covariância do ruído
$D(z)$	Polinômio do denominador da função de transferência
d	Atraso temporal aplicado ao sistema
$\Delta\eta$	Variação em pseudorrapidez
$\Delta\phi$	Variação em ângulo azimutal
e	Número de Euler; expoente real (antes do *bias*) na representação IEEE 754
e^{-ds}	Termo de atraso no domínio contínuo
e_{energ}	Erro energético normalizado
$e[n]$	Erro amostral entre a resposta ideal e a quantizada
E	Energia; expoente armazenado na representação IEEE 754
E_e	Energia total do erro de quantização
E_y	Energia total da resposta ideal
F	Fração armazenada (parte fracionária do significando) na representação IEEE 754
$F(x)$	Função de distribuição cumulativa
$F_X(x)$	Função de distribuição cumulativa da variável X
$F^{-1}(u)$	Função inversa da CDF
f	Frequência de amostragem
G_x	Ganho aplicado à entrada do filtro
G_y	Ganho de quantização dos coeficientes de saída ($G_y = 2^{\mathtt{FB}}$)
$G_{\text{SAIDA\_LOG}}$	Fator de escala aplicado ao termo de realimentação
$g[n]$	Estado interno *backward* do filtro *lattice*
$h[n]$	Resposta ao impulso do sistema discreto; estado intermediário do filtro *lattice* de segunda ordem
$h[n-1]$	Estado intermediário h atrasado de uma amostra
h_{iir}	Resposta ao impulso obtida pelo método IIR convencional

h_{latt}	Resposta ao impulso obtida pelo método *lattice* quantizado
$H(s)$	Função de transferência no domínio de Laplace
$H(z)$	Função de transferência no domínio Z
$H_1(z)$	Função de transferência de um filtro IIR de primeira ordem
$H_i(z)$	Função de transferência do i-ésimo filtro IIR
i	Índice do filtro ($1 \leq i \leq 7$)
J	Função custo (erro quadrático médio)
k	Coeficiente de reflexão do filtro *lattice* de primeira ordem
k_1	Coeficiente de reflexão do primeiro estágio do filtro *lattice* de segunda ordem
k_2	Coeficiente de reflexão do segundo estágio do filtro *lattice* de segunda ordem
λ	Parâmetro da distribuição exponencial
$\ln(\cdot)$	Logaritmo natural
m	Módulo do LCG
M	Ordem do numerador do filtro
μ	Média da distribuição exponencial; prefixo SI micro
n	Índice discreto de amostra
N	Número de amostras; ordem do denominador do filtro
occ	Limiar de ocupação
$p(\%)$	Porcentagem de ocupação
p_k	k-ésimo polo da função de transferência contínua
$P(\cdot)$	Operador de probabilidade
$P(z)$	Polinômio do numerador da função de transferência
pp	Colisão próton-próton
pC	Picocoulomb
ϕ	Ângulo azimutal
π	Constante pi
$\mathbf{p}$	Vetor de correlação cruzada
$\mathbf{R}$	Matriz de autocorrelação
s	Bit de sinal na representação IEEE 754; variável complexa do domínio de Laplace
S	Variável complexa do domínio de Laplace (notação maiúscula)
s_i	i-ésima amostra do sinal
t_p	Instante de pico do pulso no domínio contínuo
T	Período de amostragem do sistema discreto
U_n	Versão normalizada do estado do LCG
w_i	Peso associado à i-ésima amostra
w_{N+1}	Termo de *bias*
$\mathbf{w}$	Vetor de pesos do filtro
$W(z)$	Sinal intermediário no domínio z
W^+	Bóson mediador W^+

W^-	Bóson mediador W^-
X	Variável aleatória; número de bits da parte inteira no formato QX.Y
x	Realização da variável aleatória
$x[n]$	Entrada do sistema discreto
X_n	Estado do LCG no passo n
X_{n+1}	Estado do LCG no passo n+1
$X(z)$	Transformada Z do sinal de entrada
Y	Número de bits da parte fracionária no formato QX.Y
$Y(z)$	Transformada Z do sinal de saída
$y[n]$	Saída do sistema discreto quantizado
$y_{ideal}[n]$	Saída de referência do filtro sem quantização
y_z	Termo de avanço do filtro IIR
z	Variável complexa do domínio Z
z^{-1}	Operador de atraso unitário
z^{-2}	Operador de atraso de duas amostras
z_p	Localização de um polo no plano-z
$\|z_p\|$	Módulo da posição do polo
Z^0	Bóson mediador Z^0
$Zd_i(z)$	Polinômio do denominador do i-ésimo filtro no domínio Z
$Zn_i(z)$	Polinômio do numerador do i-ésimo filtro no domínio Z
°C	Grau Celsius
ADC	Contagens do conversor analógico-digital
Gb/s	Gigabits por segundo
GeV	Gigaelétron-volt
km	Quilômetro
MeV	Megaelétron-volt
MHz	Megahertz
Msps	Mega amostras por segundo
ns	Nanosegundo
T	Tesla
TeV	Teraelétron-volt
mod	Operação módulo
var($\hat{A}$)	Variância do estimador

SUMÁRIO

1 INTRODUÇÃO

Os estudos em física de partículas desempenham um papel fundamental na compreensão das interações elementares da matéria e das leis fundamentais da natureza. A investigação de fenômenos que ocorrem em escalas subatômicas exige experimentos de grande porte, capazes de produzir, detectar e analisar um elevado número de eventos em condições extremas de energia e taxa de dados. Esses experimentos demandam sistemas de instrumentação altamente especializados, nos quais a aquisição e o processamento eficiente dos sinais gerados pelos detectores são etapas críticas para a extração de informações físicas relevantes.

Nesse cenário, o CERN destaca-se como o principal centro mundial dedicado à pesquisa em física de partículas, abrigando infraestruturas experimentais de grande complexidade e escala. Entre essas, o *Large Hadron Collider* (LHC) é o maior acelerador de partículas já construído, operando com colisões em altas energias e produzindo volumes massivos de dados a cada ciclo de funcionamento. A exploração científica possibilitada pelo LHC impõe requisitos rigorosos aos sistemas de detecção e leitura eletrônica, que devem ser capazes de registrar, processar e selecionar informações relevantes em tempo real, sob condições de elevada taxa de eventos e severas restrições de latência.

O ATLAS (*A Toroidal LHC ApparatuS*) é um experimento de física de partículas de propósito geral no *Large Hadron Collider* (LHC), concebido para investigar uma vasta gama de fenômenos físicos, abrangendo desde a busca por novas partículas até a realização de testes de precisão do Modelo Padrão. Para cumprir esses objetivos, o ATLAS é composto por diferentes subsistemas, organizados de forma a medir com alta eficiência as propriedades das partículas produzidas nas colisões, como energia, momento e trajetória. A complexidade e a dimensão do experimento resultam em um fluxo intenso de sinais provenientes de milhares de canais de leitura, o que exige uma infraestrutura eletrônica capaz de processar essas informações de maneira confiável e sincronizada com a taxa de colisões do acelerador.

Dentre os subsistemas que compõem o ATLAS, destaca-se o calorímetro hadrônico de telhas (do inglês, *TileCal*), responsável pela medição da energia de hádrons e pela contribuição à reconstrução de objetos físicos de interesse, como jatos e energia transversal ausente. O *TileCal* é um calorímetro hadrônico do tipo amostrador, baseado em camadas alternadas de material absorvedor e cintiladores, cujos sinais luminosos são convertidos em sinais elétricos por fotomultiplicadores (PMTs). Esses sinais analógicos passam por uma cadeia de leitura eletrônica que inclui estágios de amplificação, conformação de pulso e digitalização, sendo posteriormente processados por sistemas digitais baseados em FPGA. A qualidade do processamento aplicado nessa etapa é determinante para a precisão da reconstrução de energia e para o desempenho global do experimento.

No contexto das atualizações previstas para a Fase II do LHC, em que o experimento ATLAS deverá operar sob condições mais exigentes de taxa de eventos e ocupância nos detectores, torna-se necessário antecipar e caracterizar o comportamento do sistema de leitura frente a cenários mais severos de operação. Nessa perspectiva, o uso de um simulador de pulsos em tempo real constitui uma ferramenta essencial para a obtenção de dados representativos, permitindo gerar sinais com propriedades controladas e compatíveis com as condições esperadas, além de viabilizar estudos sistemáticos de validação, calibração e desenvolvimento de algoritmos de processamento digital associados à eletrônica de leitura.

A partir desses sinais gerados e analisados, um dos blocos centrais a ser avaliado é o estágio de conformação de pulso do *TileCal*. Esse estágio molda a forma do pulso analógico de tensão, buscando maximizar a relação sinal-ruído e adequar suas características às restrições do sistema de aquisição. O circuito conformador consiste em um filtro passivo de sete polos seguido por dois amplificadores com saídas diferenciais em paralelo, configurados com ganhos distintos para estender a faixa dinâmica do sistema através de duas saídas independentes. Assim, para simular em FPGA, em tempo real, o sinal de saída do *shaper* amostrado na frequência de operação do digitalizador (40 MHz), torna-se necessário projetar um filtro IIR que represente o comportamento combinado das etapas de conformação e de amostragem, permitindo reproduzir digitalmente a resposta do sistema de leitura. Nesse contexto, a implementação de filtros IIR em FPGA impõe desafios específicos, destacando-se os efeitos da quantização de coeficientes e estados internos, a propagação de erros numéricos ao longo das realimentações e o impacto dessas escolhas na estabilidade do filtro e no consumo de recursos lógicos.

1.1 MOTIVAÇÃO E OBJETIVOS

Diante do cenário descrito na seção anterior, torna-se evidente a importância de avaliar diferentes estratégias de implementação de filtros digitais em FPGA, especialmente quando aplicadas a sistemas críticos como o estágio de *shaper* do *TileCal*. Implementações convencionais de filtros IIR em aritmética de ponto fixo, embora eficientes em termos de recursos, podem apresentar degradações significativas na resposta do sistema devido à quantização dos coeficientes e à acumulação de erros numéricos ao longo das realimentações. Como alternativas, abordagens como o método *lattice* e a utilização de aritmética de ponto flutuante surgem como soluções capazes de mitigar esses efeitos, ao custo de maior complexidade de *hardware*. No entanto, a escolha entre essas arquiteturas envolve compromissos diretos entre precisão numérica, estabilidade e consumo de recursos lógicos, o que motiva a realização de um estudo comparativo sistemático, voltado especificamente para aplicações em FPGA no contexto de experimentos de física de partículas.

O objetivo principal deste trabalho é o desenvolvimento de um simulador de pulsos em FPGA destinado a reproduzir, de forma controlada e em tempo real, sinais

representativos do *TileCal* do experimento ATLAS, considerando as condições de operação previstas para a Fase II do LHC, como o aumento da taxa de eventos e a maior ocupância nos detectores. O simulador é proposto como uma ferramenta para apoiar testes, validações e estudos da cadeia de leitura em cenários de maior exigência operacional, permitindo a geração de pulsos com características ajustáveis e compatíveis com tais condições.

Além do desenvolvimento do simulador, este trabalho também contempla a implementação e a avaliação de diferentes métodos de filtragem digital aplicados ao estágio de condicionamento no contexto do próprio simulador. Nesse sentido, são desenvolvidas e validadas, em ambiente de FPGA, três abordagens distintas: o filtro IIR convencional quantizado em aritmética de ponto fixo, o filtro IIR no formato *lattice* e uma implementação baseada em aritmética de ponto flutuante. Cada método é descrito a partir de sua formulação teórica e implementado considerando as restrições de recursos e desempenho impostas pela lógica reconfigurável.

Por fim, os métodos propostos são comparados de forma sistemática quanto ao erro numérico introduzido pelo processo de quantização e pela estrutura do filtro, bem como em relação ao consumo de recursos lógicos na FPGA. A partir dessa análise, o trabalho busca identificar os compromissos envolvidos na escolha de cada arquitetura, fornecendo subsídios para a seleção de estratégias de filtragem digital mais adequadas a aplicações de física de partículas que demandam processamento em tempo real e alta confiabilidade.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho será dividido da seguinte forma:

- **Capítulo 2 - Fundamentação Teórica**: Apresenta o contexto experimental do CERN, incluindo uma visão geral do LHC, do experimento ATLAS e do *TileCal*, descrevendo a cadeia de leitura eletrônica até o estágio de conformação de pulso, de forma a contextualizar o ambiente no qual este trabalho está inserido.
- **Capítulo 3 - O Simulador de Pulsos do *TileCal***: Descreve a arquitetura e o funcionamento do simulador de pulsos desenvolvido para o *TileCal*, detalhando a geração dos sinais, os parâmetros considerados e sua adequação às condições previstas para a Fase II do LHC.
- **Capítulo 4 - Métodos de Implementação do *Shaper***: Apresenta os fundamentos teóricos e as arquiteturas de filtragem digital empregadas na implementação do *shaper*, incluindo o filtro IIR convencional quantizado, o método *lattice* e a implementação em aritmética de ponto flutuante, bem como a descrição dos circuitos desenvolvidos em FPGA.

- **Capítulo 5 - Resultados**: Apresenta e analisa os resultados obtidos a partir das implementações propostas, comparando os métodos quanto ao erro numérico, estabilidade da resposta e consumo de recursos lógicos em FPGA.
- **Capítulo 6 - Conclusão**: Resume as principais contribuições do trabalho, discute as conclusões obtidas a partir dos resultados apresentados e aponta possíveis extensões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

No contexto atual do entendimento do universo, destaca-se uma teoria que busca descrever o comportamento fundamental da matéria e das forças que a regem: o Modelo Padrão das Partículas. Essa teoria explica como as interações entre as partículas elementares dão origem à estrutura e à dinâmica da matéria, unificando o comportamento das forças fundamentais (exceto a gravitação). O Modelo Padrão fornece, até hoje, uma descrição notavelmente bem-sucedida de praticamente todos os resultados experimentais observados (1, 18).

O Modelo Padrão organiza as partículas elementares em dois grandes grupos: os férmions e os bósons. Os férmions são as partículas que constituem a matéria e se subdividem em quarks e léptons. Os quarks se combinam para formar partículas compostas, como os prótons e os nêutrons, enquanto os léptons incluem partículas leves, como o elétron e os neutrinos. Já os bósons são os mediadores das interações fundamentais da natureza, isto é, são responsáveis pela transmissão das forças entre os férmions (1, 18).

Cada força fundamental da natureza está associada a um tipo específico de bóson mediador. O fóton está relacionado à interação eletromagnética, os bósons W^+, W^- e Z^0 são responsáveis pela interação fraca (ou força nuclear fraca), enquanto os glúons transmitem a interação forte (ou força nuclear forte) (1, 18).

O Modelo Padrão, entretanto, ainda apresentava uma lacuna: a ausência de uma explicação para a origem das massas das partículas elementares. Essa questão foi abordada em 1964, quando Peter Higgs e outros físicos propuseram a existência de um campo escalar — o campo de Higgs — e de sua partícula associada, o bóson de Higgs (3). Esse campo interage com férmions e bósons de forma a gerar a propriedade que se manifesta macroscopicamente como massa, completando assim a estrutura teórica do Modelo Padrão.

A existência do bóson de Higgs foi comprovada em 2012 pelos experimentos ATLAS e CMS do LHC, o Grande Colisor de Hádrons, do CERN. (4, 5)

2.1 O CERN e o LHC

Após a Segunda Guerra Mundial e considerando todo o seu contexto, a Europa encontrava-se em um momento de reestruturação não apenas econômica e política, mas também científica. Muitos pesquisadores haviam se deslocado para países com melhor infraestrutura durante o conflito, o que acabou culminando em um acordo entre físicos europeus para a criação de um centro europeu de pesquisa nuclear. Esse centro incluiria, entre seus objetivos, o estudo de mésons — novas partículas que estavam sendo observadas em raios cósmicos (6).

Foi durante o encontro intergovernamental da UNESCO realizado em Paris, em

1951, que foi adotada a primeira resolução relativa à criação de um Conselho Europeu para Pesquisa Nuclear (*Conseil Européen pour la Recherche Nucléaire*). Em seguida, um acordo foi assinado estabelecendo o conselho provisório, do qual se originou a sigla CERN, amplamente conhecida até hoje. A convenção oficial foi assinada em 1953 e entrou em vigor em 29 de setembro de 1954, com a adesão inicial de doze países fundadores: Alemanha, Bélgica, Dinamarca, França, Grécia, Itália, Noruega, Países Baixos, Reino Unido, Suécia, Suíça e Iugoslávia (6, 7).

Com todo o contexto histórico da época, ficava claro o objetivo que ia além da pesquisa avançada em física de partículas, mas também o de consolidar a cooperação científica internacional como elemento de paz e desenvolvimento. O CERN foi instalada na fronteira franco-suíça, especificamente em Meyrin, próximo à cidade de Genebra. O laboratório tornou-se um exemplo notável de colaboração global, atraindo, ao longo das décadas, diversos países ao redor do mundo.

O CERN deixa claro que o seu principal objetivo é ajudar a entender as origens do universo, como ele é feito e como funciona. Para isso, uma gama única de aceleradores de partículas é fornecida, expandindo os limites do conhecimento humano (7).

2.1.1 Os Primeiros Aceleradores

Aceleradores de partículas são instrumentos projetados para aumentar a energia cinética de partículas subatômicas, como elétrons ou prótons, guiando-as e colidindo-as sob condições controladas. À medida que essas partículas são aceleradas, sua energia total aumenta de acordo com o princípio da equivalência entre massa e energia, formulado por Albert Einstein em 1905 na famosa relação

$$E = mc^2 \tag{2.1}$$

Quando partículas são aceleradas a velocidades próximas à da luz, sua energia cinética torna-se muito maior que sua massa de repouso, e as colisões entre elas podem converter essa energia em novas partículas, conforme previsto pela conservação da energia e pela teoria quântica de campos (8, 9).

No momento da colisão entre duas partículas, a alta energia cinética é convertida em energia de massa e de ligação, que são responsáveis pelo surgimento das novas partículas, de acordo com os princípios da física quântica para pares de partículas a antipartículas (9). Esse fenômeno permite investigar estados da matéria e interações fundamentais que existiam apenas nos primeiros instantes após o Big Bang, quando o universo tinha menos de um bilionésimo de segundo de idade (10, 11).

Os aceleradores usam campos eletromagnéticos para acelerar e direcionar as partículas, eles possuem cavidades de radiofrequência para acelerar os feixes de partículas,

enquanto os ímãs os concentram e curvam sua trajetória. Dois tipos são principais: o linear e o circular.

O primeiro acelerador construído em laboratório foi o *Synchrocyclotron* (SC), concluído em 1957. Com 15 metros de diâmetro e energia de até 600 MeV (megaelétron-volts), o SC foi utilizado principalmente em experimentos com partículas leves, como píons e mésons, e desempenhou papel fundamental na formação das primeiras gerações de físicos experimentais do CERN. O equipamento permaneceu em operação por mais de quatro décadas, até 1990, sendo posteriormente convertido em uma atração histórica do laboratório (12).

O sucesso do *Synchrocyclotron* impulsionou a construção de máquinas mais potentes, como o *Proton Synchrotron* (PS), que entrou em operação em 1959. O PS representou um marco tecnológico, acelerando prótons a energias de até 26 GeV e tornando-se, à época, o acelerador mais energético do mundo e o principal do CERN, contando com 628 metros de diâmetro (13).

Enquanto os aceleradores anteriores colidiam prótons contra alvos fixos, o *Intersecting Storage Rings* (ISR), desenvolvido em 1971, foi o primeiro a colidir dois feixes de prótons entre si, atingindo uma energia máxima de 62 GeV por feixe — o que equivale a cerca de 2000 GeV em um experimento de alvo fixo (18). O ISR foi também o primeiro colisor de hádrons do CERN e abriu caminho para os aceleradores posteriores, ao demonstrar o potencial das colisões frontais e fornecer evidências experimentais da estrutura interna dos prótons, hoje compreendida em termos de quarks e glúons (14).

o *Super Proton Synchrotron* (SPS), sucessor do ISR, foi ligado pela primeira vez em 1976, operando em até 450 GeV.

O SPS, concluído em 1976, representou um salto tecnológico significativo no desenvolvimento dos aceleradores do CERN. Com aproximadamente 7 km de circunferência, o SPS foi inicialmente projetado para acelerar prótons até 400 GeV, tornando-se o acelerador de mais alta energia do mundo na época. Em 1981, ele passou por uma modificação fundamental que o transformou em um colisor de prótons e antiprótons, permitindo colisões a uma energia no centro de massa de 540 GeV (15, 18).

Essas colisões levaram à descoberta dos bósons intermediários das interações fracas — o W^+, W^- e Z^0 — anunciadas em 1983 pelos experimentos UA1 e UA2, sob a liderança de Carlo Rubbia e Simon van der Meer, que receberam o Prêmio Nobel de Física em 1984 (19, 20, 21). Atualmente, o SPS continua em operação, atuando como um estágio intermediário na cadeia de aceleração que alimenta o *Large Hadron Collider* (LHC).

O *Large Electron–Positron Collider* (LEP), concluído em 1989, foi o sucessor direto do SPS e ocupou o mesmo túnel de 27 km de circunferência atualmente utilizado pelo *Large Hadron Collider* (LHC). O LEP foi projetado para colidir elétrons e pósitrons a energias progressivamente maiores, começando em 45 GeV por feixe (fase LEP1) e alcançando até

104 GeV por feixe (fase LEP2) (16).

O LEP encerrou suas operações em 2000, após mais de uma década de resultados experimentais fundamentais. Sua desmontagem permitiu a instalação do *Large Hadron Collider*, inaugurando uma nova era na exploração das energias de escala teraelétron-volt (TeV).

2.2 LHC - O GRANDE COLISOR DE HÁDRONS

O LHC é o maior e mais poderoso acelerador de partículas do mundo. Teve seu início de operação em setembro de 2008 e foi a última adição para o complexo de aceleradores do CERN, mostrado na Figura 1. Com um diâmetro de 27 km, ele conta com uma estrutura com vários ímãs e supercondutores que aceleram e direcionam as partículas, aumentando sua energia ao longo do caminho (17).

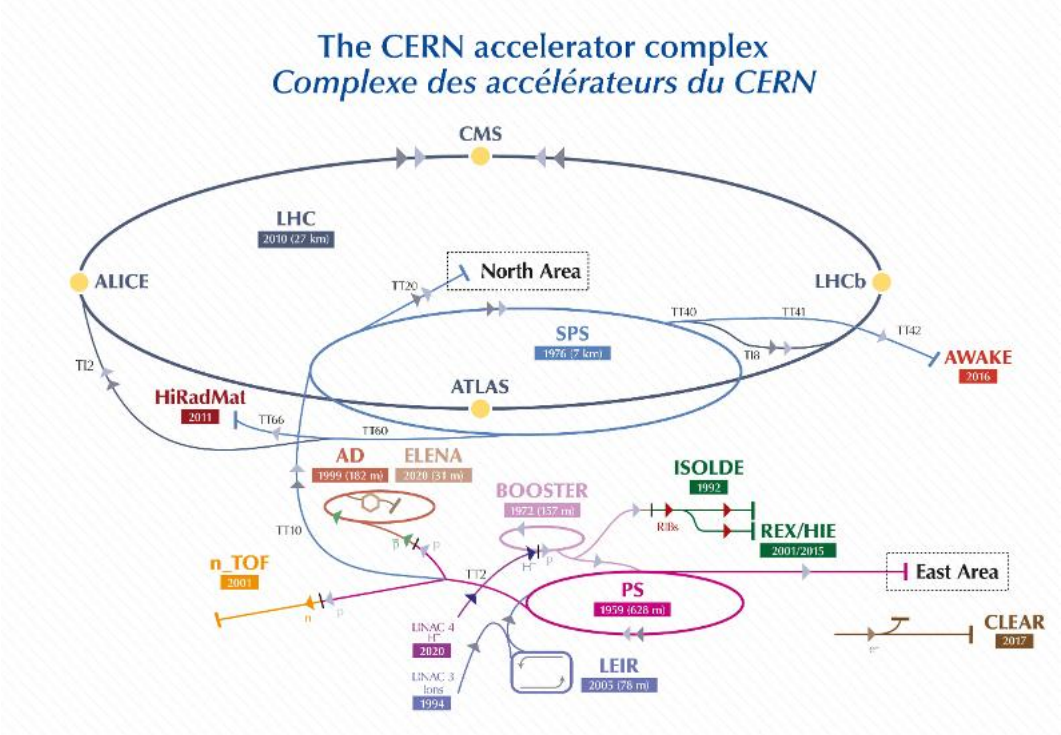

Figura 1 - Complexo de aceleradores do CERN (22)

2.2.1 Princípios de Funcionamento do LHC

Dentro do LHC, os feixes de partículas viajam em direções opostas em tubos separados, a velocidades próximas à da luz, antes de colidirem entre si. Eles são guiados e acelerados por eletroímãs supercondutores, mantidos em condições de vácuo extremo (17).

Para atingir tais condições, os ímãs do LHC são resfriados a uma temperatura de aproximadamente $-271{,}3\ ^{\circ}\mathrm{C}$ (mais fria do que o espaço sideral) permitindo que operem

em estado supercondutor, conduzindo eletricidade sem perdas de energia. Para isso, grande parte do acelerador está conectada a um sistema de distribuição de hélio líquido, responsável por manter os ímãs nessa temperatura extremamente baixa (17).

Os prótons utilizados nas colisões do LHC são extraídos de átomos de hidrogênio e acelerados gradualmente por uma cadeia de aceleradores menores (o *Linac2*, o *Proton Synchrotron Booster* (PSB), o *Proton Synchrotron* (PS) e o *Super Proton Synchrotron* (SPS)) antes de serem injetados no anel principal (23). Cada estágio aumenta progressivamente a energia das partículas, garantindo que atinjam até 6,5 TeV por feixe no LHC. Dentro do anel, cerca de 2808 pacotes de prótons circulam simultaneamente, guiados por mais de 1200 dipolos magnéticos que curvam suas trajetórias e por quadrupolos que mantêm o foco dos feixes (23).

As colisões ocorrem em quatro pontos principais do anel, onde estão instalados os grandes experimentos do LHC: ATLAS, CMS, ALICE e LHCb, conforme mostrado na Figura 2. Em cada ponto, os feixes são cruzados com precisão micrométrica, permitindo que as partículas colidam a cada 25 ns. Os experimentos associados registram e analisam os produtos dessas colisões, possibilitando a investigação das forças fundamentais e da estrutura da matéria (17).

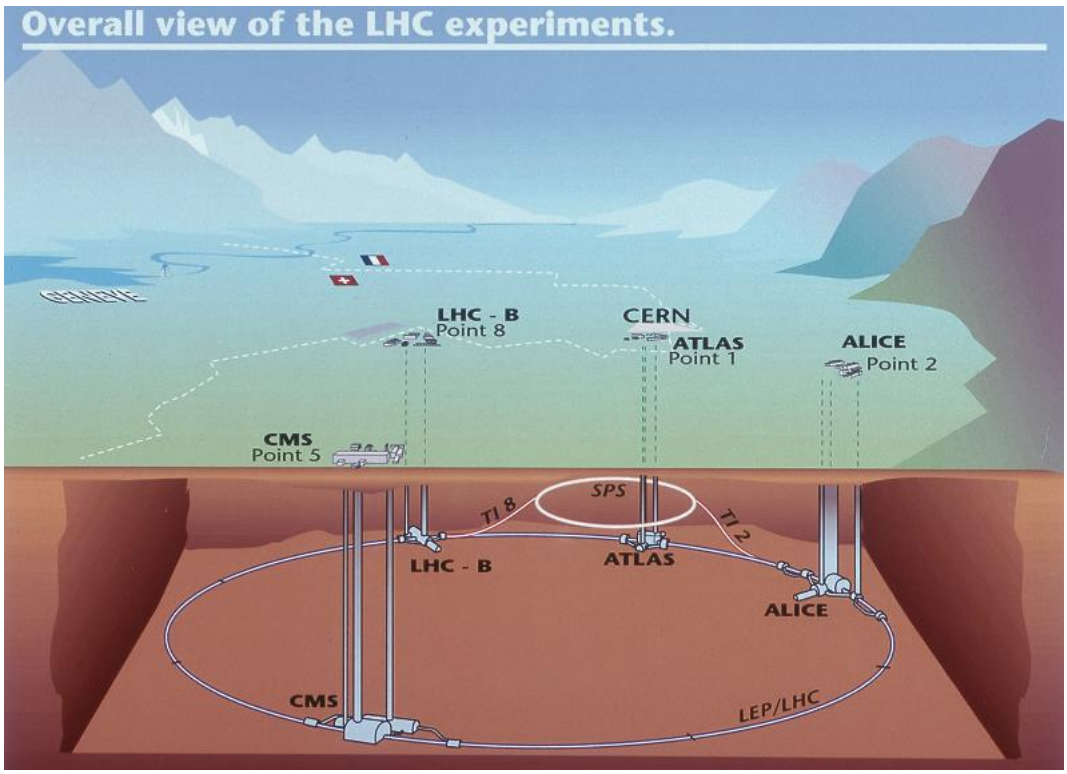

Figura 2 - Estrutura do LHC (25)

2.2.2 Os Experimentos do LHC

O LHC possui quatro grandes experimentos que são os detetores das colisões, são eles: ALICE, LHCb, CMS e ATLAS.

O ALICE (*A Large Ion Collider Experiment*) é um experimento do LHC dedicado ao estudo de colisões de íons pesados, concebido para reproduzir em laboratório as condições extremas do universo primordial. Ao colidir núcleos de chumbo a energias ultra-relativísticas, o experimento gera o Plasma de Quarks e Glúons (QGP), um estado da matéria que permite investigar propriedades fundamentais da Cromodinâmica Quântica (QCD), como a restauração da simetria quiral e o mecanismo de confinamento. Localizado em uma caverna experimental subterrânea, o complexo sistema de detecção do ALICE monitora a expansão e o resfriamento desse plasma, identificando as partículas resultantes que compõem a matéria bariônica atual (26).

O ATLAS (*A Toroidal LHC ApparatuS*) é um dos dois experimentos de uso geral do LHC, projetado para investigar uma ampla gama de fenômenos físicos — que vão desde o estudo do bóson de *Higgs* até a busca por partículas que possam compor a matéria escura. No centro do detector, os feixes de partículas do LHC colidem produzindo novas partículas que se espalham em todas as direções, sendo registradas por seis subsistemas dispostos em camadas, responsáveis por medir suas trajetórias, momento e energia. Um grande sistema de ímãs curva as partículas carregadas, permitindo determinar seu momento, enquanto um complexo sistema de aquisição e processamento seleciona e analisa os eventos mais relevantes entre o enorme volume de dados gerado (29).

O CMS (*Compact Muon Solenoid*) possui um amplo programa de investigação que vai desde o Modelo Padrão até à procura de dimensões extras e partículas que possam constituir a matéria escura. Embora tenha objetivos semelhantes ao ATLAS, ele adota soluções técnicas distintas, sendo construído em torno de um enorme solenoide supercondutor, gerando um campo magnético de 4 T (aproximadamente 100 000 vezes o campo magnético da Terra) (24, 28).

O LHCb (*Large Hadron Collider beauty*) é o experimento especializado em investigar as pequenas diferenças entre matéria e antimatéria através do quark *beauty* (B). Utiliza uma série de sub-detectores para detectar principalmente partículas frontais — as lançadas para frente em uma colisão em uma direção. Após a colisão, um grande número de quarks são criados pelo LHC antes de se decomporem em outras partículas menores. Para capturar os quarks B, o experimento usa detectores de rastreio móveis e de alta precisão próximos ao feixe (24, 27).

2.3 O EXPERIMENTO ATLAS

O ATLAS, mostrado na Figura 3, e como mencionado anteriormente, é um dos dois experimentos de propósito geral do LHC. Seu foco é na colisão próton-próton a altas energias. Seu projeto visa identificar, com elevada precisão, as partículas produzidas nessas colisões, permitindo investigar desde propriedades fundamentais do Modelo Padrão até possíveis sinais de nova física (30).

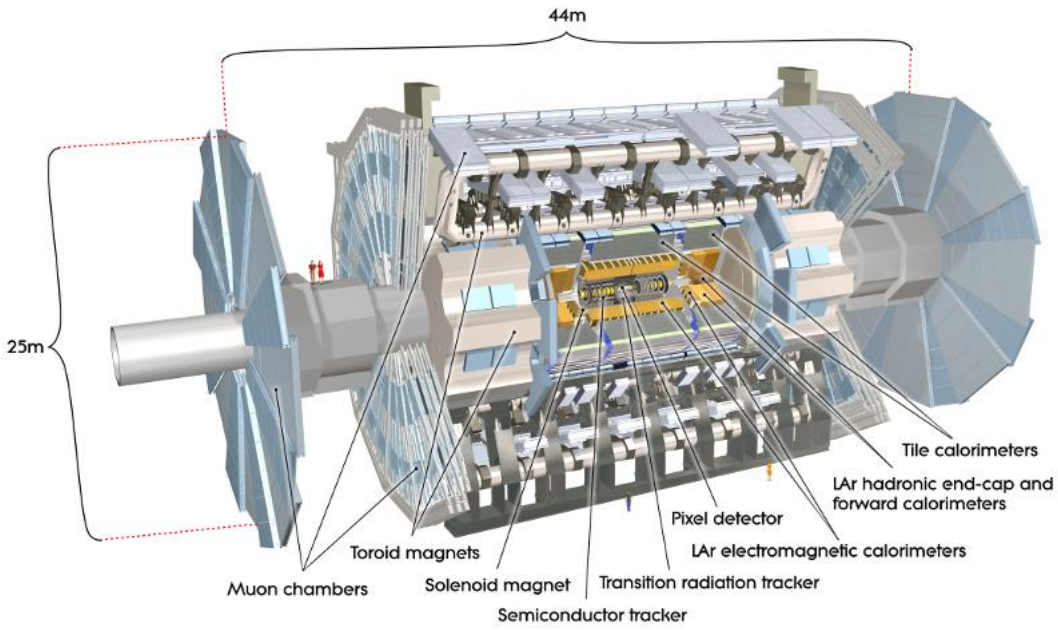

Figura 3 - Estrutura do ATLAS (31)

O experimento ATLAS possui aproximadamente 46 metros de comprimento, 25 metros de diâmetro e uma massa total de cerca de 7 mil toneladas. Sua arquitetura segue o princípio de detecção em camadas concêntricas em torno do ponto de interação, de modo que cada subsistema seja especializado em medir diferentes propriedades das partículas, como trajetória, momento, energia e tipo de carga (32).

De maneira geral, o ATLAS possui três sistemas principais, conhecidos como subsistemas, ordenados do mais interno para o mais externo, sendo eles:

- *Inner Detector* (ID).
- Sistema de Calorímetros.
- Espectrômetro de múons.

O *Inner Detector* (ID) é o subsistema mais interno do ATLAS e tem como principal função a reconstrução precisa das trajetórias das partículas carregadas e a determinação

do seu momento no campo magnético solenoidal de 2 T. Ele é composto por três subdetectores mais internos: o *Pixel Detector*, o *SemiConductor Tracker* (SCT) e o *Transition Radiation Tracker* (TRT). A combinação desses três componentes permite uma reconstrução tridimensional precisa das trajetórias e vértices primários e secundários, essencial para estudos de física de precisão e para a identificação de partículas de curta vida média (33).

O sistema de calorimetria do ATLAS, mostrado na Figura 4, é responsável pela medição da energia depositada por partículas ao interagirem com a matéria, sendo dividido em dois subsistemas principais. O Calorímetro Eletromagnético, também conhecido como *LAr (Liquid Argon)*, mede a energia de fótons e elétrons — partículas que interagem de forma eletromagnética com o meio (34). Já o Calorímetro Hadrônico, denominado *TileCal (Tile Calorimeter)*, mede a energia de partículas que interagem de forma hadrônica com o meio, ou seja, hádrons em sua maioria (35). Esses dois sistemas complementares permitem a reconstrução precisa da energia total depositada nos eventos e desempenham um papel essencial na identificação e calibração de jatos e na determinação do momento transverso ausente.

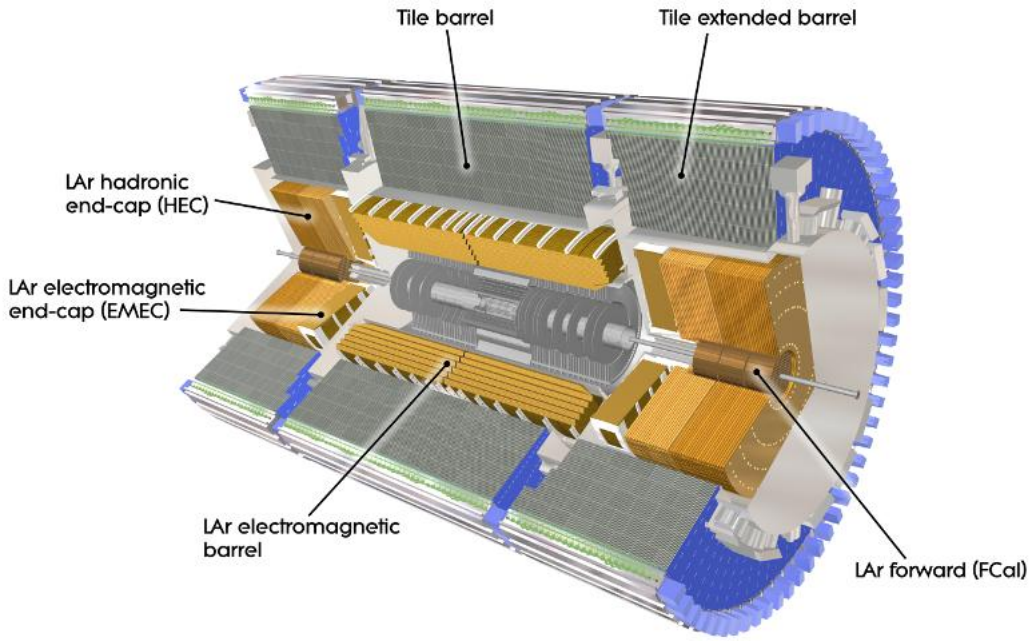

Figura 4 - Sistema de calorimetria do ATLAS (36)

O Espectrômetro de Múons (*Muon Spectrometer* — MS), apresentado na Figura 5, constitui o sistema mais externo do detector ATLAS. Ele é responsável pela identificação e medição do momento dos múons, as únicas partículas carregadas capazes de atravessar os calorímetros praticamente sem sofrer absorção. Ele é composto por cinco diferentes tecnologias de detectores, que são: *Thin Gap Chambers* (TGC), *Resistive Plate Chambers*

(RPC), *Monitored Drift Tubes* (MDT), *Small Strip Thin Gap Chambers* e *Micromegas*. A combinação desses elementos garante uma medição precisa do momento transversal dos múons e desempenha papel fundamental na identificação de eventos contendo bósons pesados ou novas partículas (37).

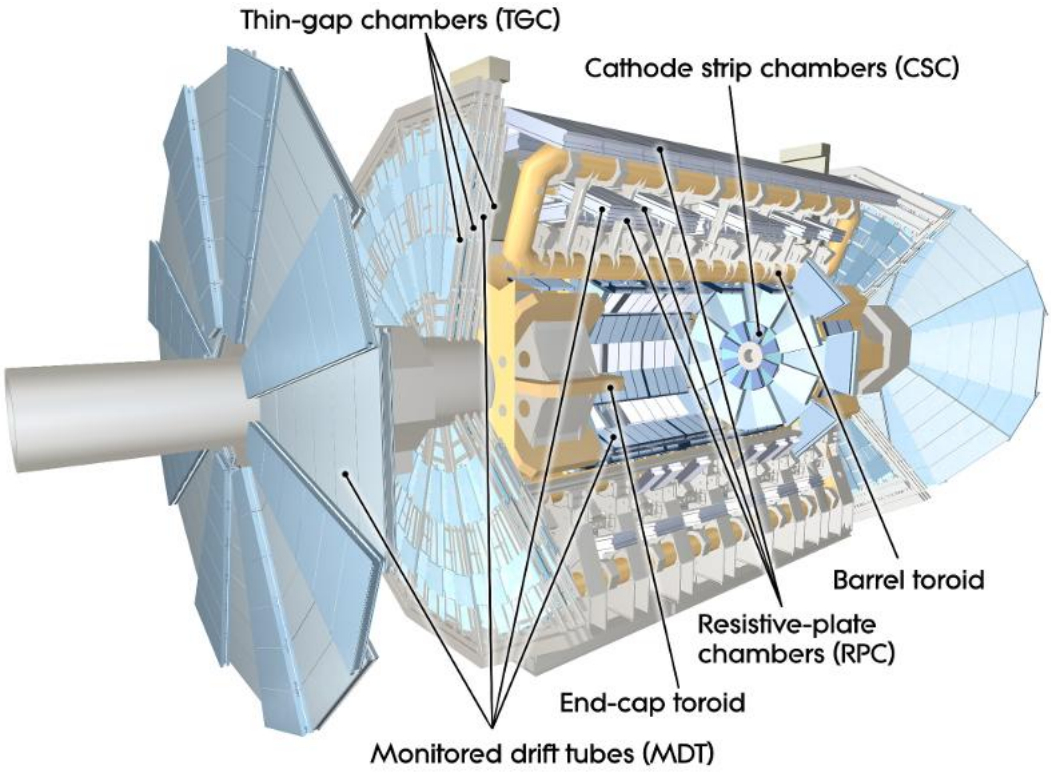

Figura 5 - Espectrômetro de Múons do ATLAS (39)

A Figura 6 ilustra as trajetórias de diferentes tipos de partículas e como elas interagem com cada subsistema do ATLAS. Nela é possível observar a deposição de energia em cada camada, conforme descrito anteriormente. Elétrons e fótons depositam sua energia predominantemente na segunda camada — o Calorímetro Eletromagnético de Argônio Líquido — enquanto prótons e nêutrons interagem de forma hadrônica, depositando energia no Calorímetro Hadrônico (*TileCal*). Já os múons, por sua vez, atravessam quase todas as camadas e são detectados apenas no Espectrômetro de Múons, uma vez que sua energia não é suficientemente absorvida pelos detectores mais internos. Também é ilustrada a trajetória dos neutrinos, que interagem muito fracamente com a matéria e, portanto, não são detectados pelos sub-detectores do ATLAS.

2.4 TILECAL - O CALORÍMETRO HADRÔNICO DE TELHAS

O *TileCal* é o calorímetro responsável pela identificação de hádrons, medindo a energia por eles depositada. Este calorímetro de amostragem é composto por camadas

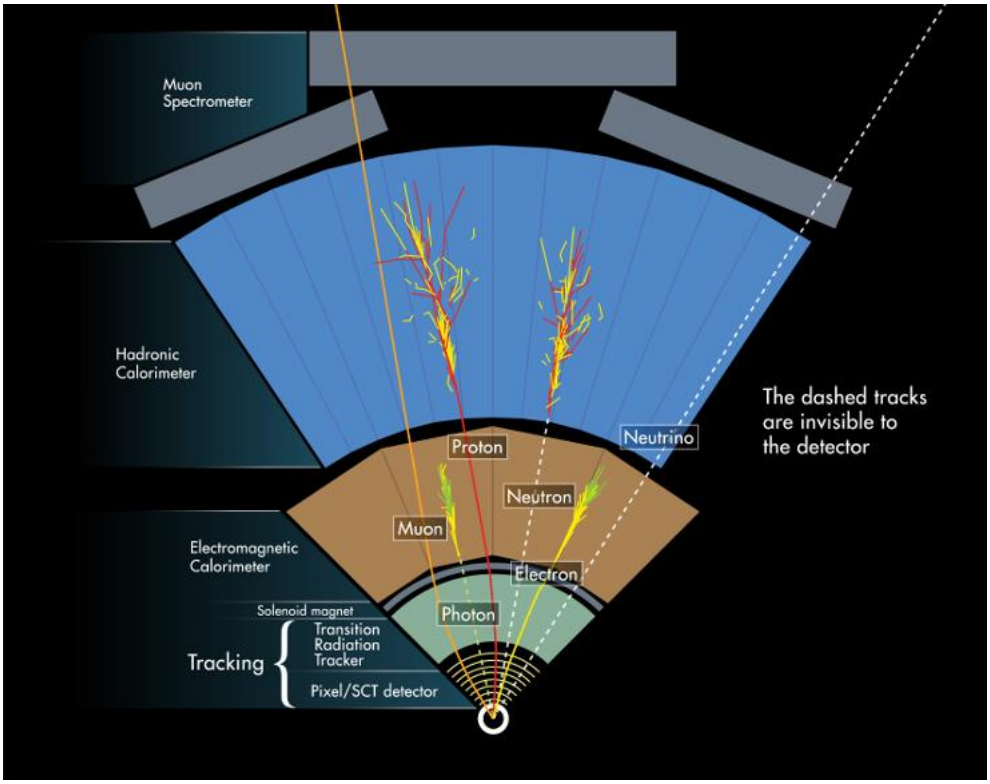

Figura 6 - Interação das partículas com os sub-detectores do ATLAS (39)

alternadas de aço, que atuam como material absorvedor, e cintiladores plásticos, que geram luz de cintilação a partir das interações hadrônicas para a subsequente medição da energia. Essa luz é coletada por fibras ópticas de deslocamento de comprimento de onda (*Wavelength Shifting fibers*) e convertida em sinais elétricos por fotomultiplicadoras (*Photomultiplier Tubes*, PMTs). Tais sinais apresentam uma amplitude de corrente proporcional à energia depositada pela partícula incidente, permitindo a reconstrução precisa do evento.

Em termos estruturais, o *TileCal* possui formato cilíndrico e é dividido em três grandes seções: um barril central (*Long Barrel*) e dois barris estendidos (*Extended Barrels*), localizados em ambas as extremidades do detector. Cada barril é segmentado azimutalmente em 64 módulos, dispostos ao redor do eixo do feixe, cobrindo 2π em ϕ. Os módulos do barril central contêm até 45 canais de leitura óptica independentes, cada um associado a uma PMT, enquanto os módulos dos barris estendidos contêm 32 canais, resultando em um total de aproximadamente 10 000 canais de leitura em todo o sistema. Essa geometria modular garante uma cobertura uniforme em η e ϕ, além de facilitar a manutenção e o acesso aos componentes eletrônicos (35).

A Figura 7 mostra a estrutura de um módulo do *TileCal*, em que é possível observar as telhas, que são dispostas perpendicularmente ao feixe de partículas.

O *TileCal* é estruturado em três camadas radiais principais: A, BC e D, no barril

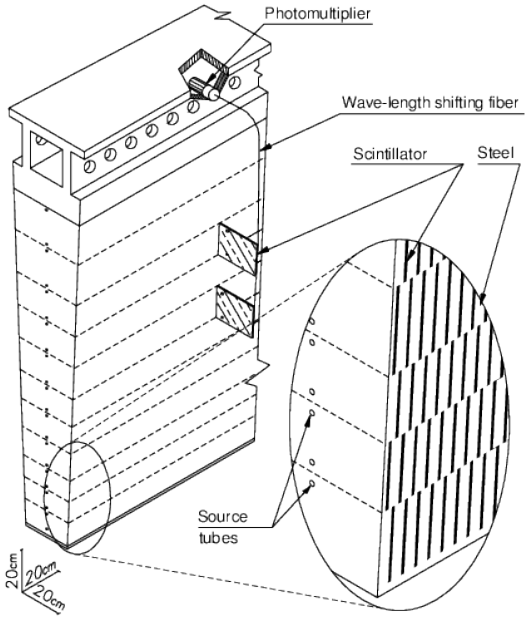

Figura 7 - Módulo do *TileCal* (35)

central (*Long Barrel*); e A, B e D, nos barris estendidos (*Extended Barrels*). As células dessas camadas cobrem regiões do espaço definidas em termos de pseudorapidez e ângulo azimutal, e podem ser agrupadas logicamente (seja por eletrônica analógica, no cenário atual, seja digitalmente, no contexto do *HL-LHC*) para formar torres projetivas em relação ao ponto de interação, permitindo que a energia depositada pelas partículas seja associada de forma coerente à sua direção original. Cada torre cobre uma região típica de $(\Delta\eta, \Delta\phi) \approx (0.1, 0.1)$. O barril central cobre a região $|\eta| < 1.0$, enquanto os barris estendidos estendem essa cobertura até aproximadamente $|\eta| < 1.7$.

A Figura 8 apresenta um recorte esquemático das células do *TileCal*, evidenciando a disposição das camadas e a correspondência entre as regiões do barril central e dos barris estendidos.

2.4.1 Processamento dos Sinais no *TileCal*

A saída da PMT é conectada à placa de *front-end* do *TileCal*, denominada *3-in-1 card*, representada esquematicamente na Figura 9. Essa placa é responsável pelo condicionamento do pulso proveniente da PMT e pela geração de duas saídas lineares com razão de ganho de 64, cobrindo uma faixa dinâmica equivalente a 16 bits (41).

O condicionamento do sinal é realizado por um circuito interno da placa, denominado *shaper*, que produz um pulso com largura total a meia altura (*Full Width at Half Maximum, FWHM*) de aproximadamente 50 ns. Dessa forma, o formato do pulso do *TileCal* pode ser considerado praticamente invariante entre os diferentes canais, e a

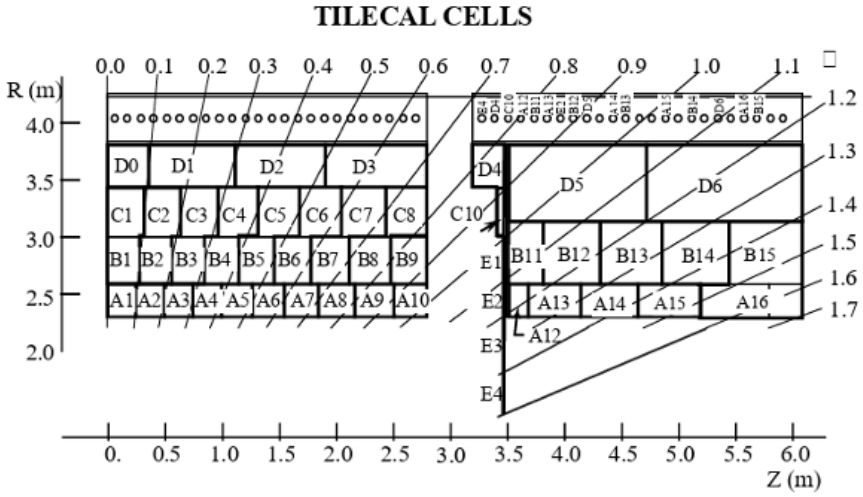

Figura 8 - Células do *TileCal*

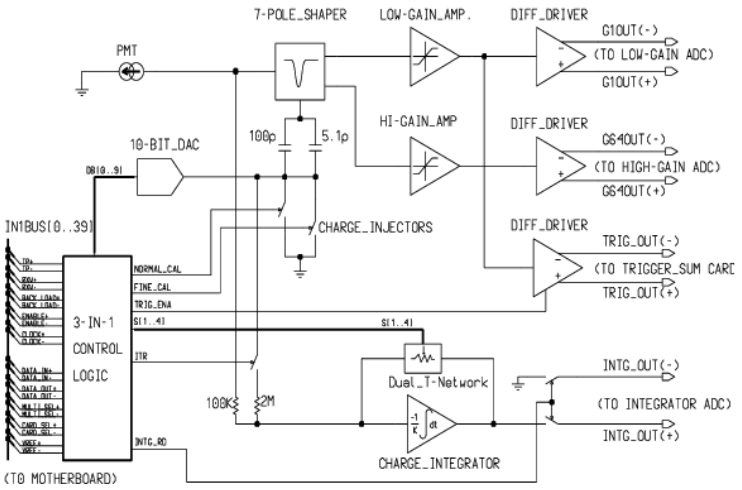

Figura 9 - Diagrama de blocos da *3-in-1 card* (41)

energia depositada pela partícula em uma determinada célula pode ser obtida a partir da estimativa corrigida da amplitude do pulso. Para abranger toda a faixa de energia do calorímetro (de aproximadamente 220 MeV a 1.3 TeV), o sinal modelado é amplificado por dois amplificadores operacionais com razão de ganho 64, produzindo uma saída de *High Gain* e outra de *Low Gain* (40).

O circuito do *shaper* é implementado como um filtro passivo de 7 polos, constituído por uma rede RLC, conforme ilustrado na Figura 10. Sua resposta temporal apresenta uma cauda longa, característica que torna adequada a sua modelagem no simulador de pulsos, apresentado no próximo capítulo, por meio de um filtro IIR capaz de reproduzir a dinâmica

do circuito de forma eficiente. Detalhes adicionais sobre essa rede e sua modelagem são discutidos na seção dedicada ao *shaper*.

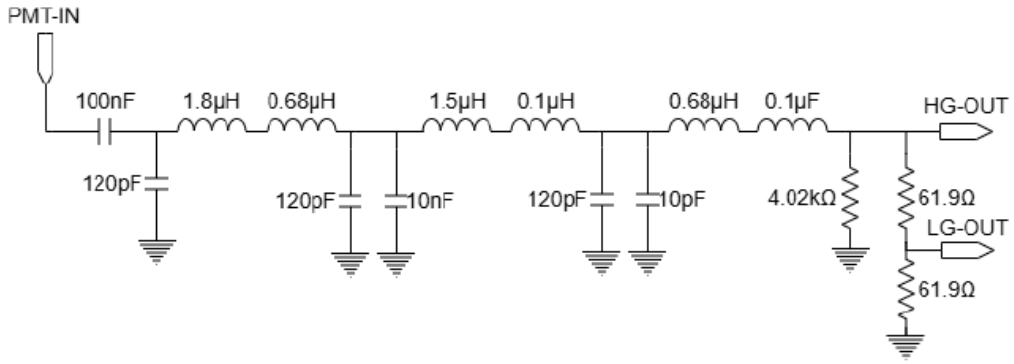

Figura 10 - Circuito de condicionamento do pulso da PMT (59)

As duas saídas analógicas da *3-in-1* são enviadas para a placa digitalizadora (*Digitizer Board*), onde são convertidas em amostras digitais a uma frequência de 40 MHz. Uma janela de sete amostras (150 ns) engloba praticamente todo o pulso e é lida a cada evento. O sinal analógico característico do *TileCal*, bem como os parâmetros de amostragem, podem ser observados na Figura 11, na qual as sete amostras estão representadas por pontos espaçados de 25 ns entre si (40, 41).

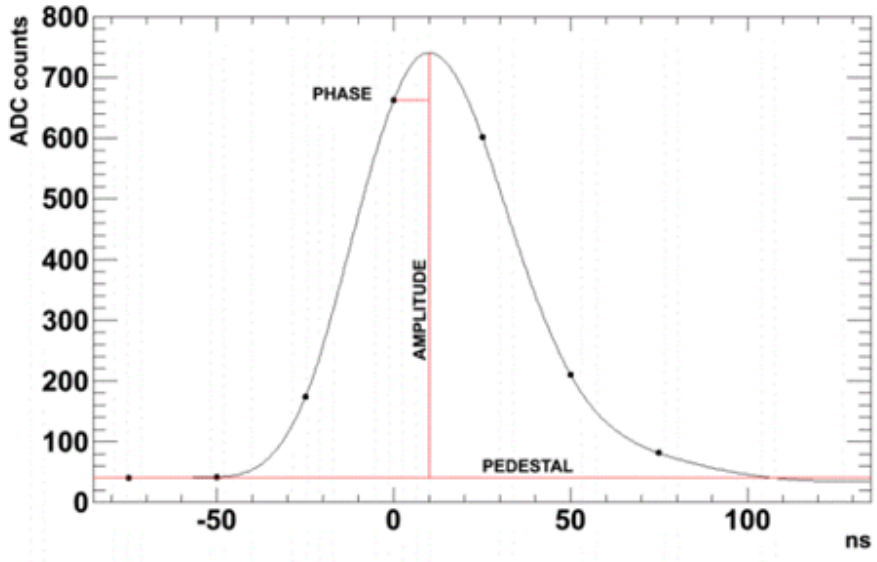

Figura 11 - Pulso característico do *TileCal* (40)

A Figura 11 apresenta apenas a porção positiva do pulso. A componente negativa, por sua vez, possui uma cauda mais longa e pode estender-se por alguns microssegundos

até retornar completamente à linha de base. Quando um pulso é analisado de forma isolada, essa contribuição pode parecer pouco relevante. No entanto, nas condições de maior empilhamento de sinais esperadas para a Fase II, essa cauda negativa pode induzir variações na linha de base devido à sobreposição entre pulsos sucessivos, influenciando a resposta observada e a reconstrução do sinal. Por esse motivo, é necessário que o modelo utilizado no simulador represente adequadamente essa dinâmica, o que é obtido pela modelagem do *shaper* por meio do filtro IIR apresentada nos capítulos posteriores.

A partir da *Main Board*, os dados digitalizados de cada canal são transmitidos por enlaces ópticos aos módulos *Read-Out Driver* (ROD), localizados na sala de serviço do ATLAS (USA15). O ROD agrega as amostras correspondentes aos canais e módulos associados a cada sinal de *Level-1 Accept* (L1A), aplica constantes de calibração (pedestais, ganho e fase) e executa em *DSPs* a reconstrução rápida de amplitude e tempo — tipicamente via *Optimal Filtering* (OF), podendo empregar *Matched Filter* (MF) em cenários específicos de estudo (40). Em seguida, o ROD formata os fragmentos de evento e os envia ao *Read-Out System* (ROS), que os disponibiliza ao sistema global de aquisição e seleção (*TDAQ/HLT*) para a formação do evento completo e posterior processamento (42, 43).

A Figura 12 ilustra o esquema eletrônico do *TileCal* no atual cenário.

Figura 12 - Fluxo eletrônico do *TileCal*

2.4.2 Sistemas de Calibração do *TileCal*

O TileCal possui quatro sistemas de calibração, que garantem uma medição precisa da energia. Eles são divididos em Laser, Césio, CIS (*Charge Injection System*) e *Minimum Bias* (MB). São usados para definir a escala das medições e corrigir as variações temporais das respostas dos componentes eletrônicos e ópticos. A energia depositada em cada canal do *TileCal* é dada por (44):

$$E[GeV] = A[ADC] \times C_{ADC \to pC} \times C_{pC \to GeV} \times C_{Cs} \times C_{Laser} \times C_{MB} \tag{2.2}$$

- $A[ADC]$: Amplitude do sinal, determinada utilizando as sete amostras do sinal na leitura digital.
- $C_{pC \to GeV}$: Fator que converte a energia de pC para GeV, obtido em feixes de teste utilizando elétrons e múons (44).

- $C_{ADC \rightarrow pC}$: Fator de conversão ADC→pC, obtido usando o sistema CIS (*Charge Injection System*), que injeta uma carga predefinida na cadeia de leitura, permitindo corrigir instabilidades da eletrônica. As calibrações de CIS são realizadas em média duas vezes por semana. Esse fator de conversão também é conhecido como constante de calibração CIS (44).

- C_{Cs}: Constante de calibração de Césio, determinada utilizando uma fonte radioativa removível de Césio 137 (C^{137}). Esse sistema monitora toda a cadeia óptica do *TileCal* (tiles cintiladores, fibras *wavelength-shifting* e PMTs), medindo as derivas acumuladas desses componentes e sendo essencial para estabelecer e manter a escala correta de energia eletromagnética (44).

- C_{Laser}: Constante de calibração de *Laser*, determinada pelo sistema de *Laser*. Ele opera injetando luz monocromática diretamente no fotocátodo de cada PMT, permitindo medir variações de ganho e *timing* (44).

- C_{MB}: Obtida a partir do sistema de *Minimum Bias* que mede o valor das correntes nos PMTs durante as colisões próton-próton. O sistema compartilha a saída com o de Césio e é usado especificamente para calibrar as células que não são alcançada com o sistema de Césio (44).

A Figura 13 mostra o esquema para os quatro sistemas de calibração do *TileCal*.

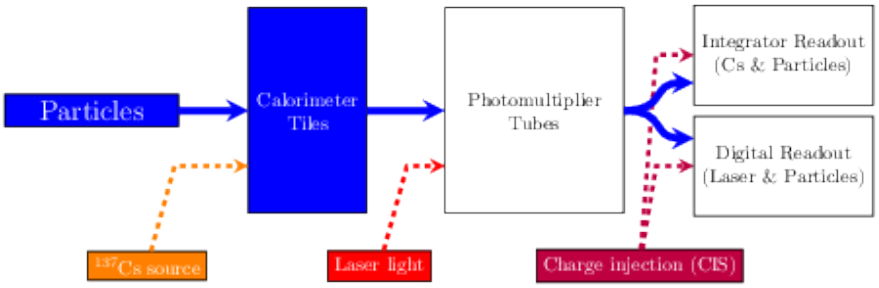

Figura 13 - Sistemas de calibração do *TileCal* (44)

2.4.3 O *TileCal* no HL - LHC

O *High-Luminosity Large Hadron Collider* (HL-LHC) foi concebido para elevar o desempenho do LHC e ampliar o potencial de descobertas científicas após 2030. O objetivo central é aumentar a luminosidade integrada em um fator de dez além do valor de projeto do LHC. Como a luminosidade é proporcional ao número de colisões por unidade de tempo, sua elevação permite acumular muito mais dados experimentais, condição indispensável

para observar processos raros e reduzir incertezas sistemáticas em medições de precisão (45).

Com a operação prevista para meados da década de 2030, o HL-LHC permitirá estudar com maior detalhe mecanismos já estabelecidos — como as propriedades do bóson de Higgs — e, ao mesmo tempo, abrir janelas para fenômenos novos de baixíssima probabilidade. Para ilustrar a mudança de escala, estima-se a produção de pelo menos 15 milhões de bósons de Higgs por ano no HL-LHC, em contraste com os cerca de três milhões registrados no LHC em 2017, o que potencializa buscas em canais exóticos e análises diferenciais antes limitadas por estatística (45).

O *TileCal* passará por um upgrade abrangente de Fase II para a operação no HL-LHC, com substituição integral da eletrônica *on/off-detector* — esquematizada na Figura 14 — para compatibilização com um sistema *TDAQ* totalmente digital. O novo arranjo proverá maior precisão na digitalização e na medição de energia, maior tolerância à radiação, redundância e melhorias em potência, calibração e sistemas mecânicos. Com isso, busca-se preservar desempenho e estabilidade sob condições extremas de *pile-up* e radiação — com até ~ 200 interações *pp* por *bunch crossing* (47).

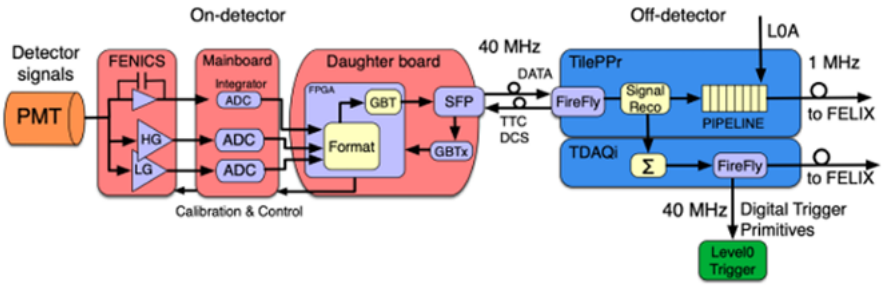

Figura 14 - Eletrônica do *TileCal* para a fase II (46)

O *TileCal* substituirá os atuais *super-drawers* — duas gavetas longas que compartilham potência e serviços e dificultam a manutenção — por um arranjo modular. No *long barrel*, serão utilizados quatro *mini-drawers* (MDs); no *extended barrel*, três MDs acompanhados de dois *micro-drawers* (μDs). Cada MD pode acomodar até 12 PMTs e possui alimentação e leitura independentes, enquanto os μDs (com 2 PMTs cada) são lidos pelos MDs adjacentes (47).

As eletrônicas *on-detector* do TileCal para o HL-LHC foram projetadas com tolerância à radiação e redundância, reduzindo perdas de dados em alta ocupância. Cada *mini-drawer* integra três blocos (47):

- ***FENICS*** (*Front-End for the New Infrastructure with Calibration and Signal Shaping*), que amplifica e modela os sinais dos PMTs com duplo ganho (1:40), cobrindo

$\sim$ 200–1000 pC, e incorpora injeção de carga para calibração, além de leitura rápida (física) e leitura lenta por integrador para calibração com césio. Tal bloco, seguido da digitalização à 40 MHz é a parte que será desenvolvida no presente trabalho.

- ***Main Board*** (MB), que digitaliza os sinais moldados com ADCs de 12 bits a 40 Msps, gerencia temporização/controle, regula baixas tensões e faz a interface com o *back-end*.
- ***Daughter Board*** (DB), montada sobre a MB, que agrega os dados digitalizados e os envia por links ópticos GBT a 9,6 Gb/s, além de distribuir *clock*, comandos de controle e configuração.

O sistema de leitura *off-detector* recebe, via links ópticos de alta velocidade, os sinais digitalizados enviados pelas *Daughter Boards* e os processa nos módulos *PreProcessor* (PPr) instalados em *crates* ATCA. Cada PPr, baseado em FPGAs de alto desempenho, realiza o tratamento em tempo real: reconstrução e formatação dos dados calorimétricos, além de distribuir relógio do LHC e sinais de controle para sincronizar a eletrônica *on-detector* durante a amostragem dos PMTs (47).

Para garantir uma medição estável e precisa dos pulsos na fase II, considerando o aumento na luminosidade, o sistema de calibração do *TileCal* também passará por uma fase de atualização.

O sistema de *Laser* contará com uma nova fonte e uma nova interface de controle (ILANA). Além disso, uma matriz adicional de LEDs foi integrada para emular as condições de alto *pile-up* esperadas no HL-LHC, superpondo um componente DC aos pulsos de laser. O sistema de Césio O foi atualizado com novas eletrônicas *on-* e *off-detector* utilizando links ópticos, além de um sistema hidráulico modernizado para melhorar desempenho e confiabilidade (47).

2.4.4 Efeitos de *pile-up* no TileCal em Condições de Alta Luminosidade

O aumento da luminosidade prevista na fase II do LHC vem acompanhada com o aumento de um efeito indesejado conhecido como *pile-up*. Este corresponde à ocorrência de múltiplas interações em uma mesma passagem dos feixes pelo ponto de interação ou em cruzamentos adjacentes. Em cada *bunch-crossing* ocorrem dezenas de colisões, além da colisão de interesse, depositando energia nos subdetectores e gerando sinais que se sobrepõem ao evento principal (48).

A Figura 15 mostra o efeito do *pileup* para uma janela de pulso do *TileCal*. Nela, a linha preta é o pulso de interesse, sendo a forma esperada do sinal, enquanto a linha vermelha é um pulso deslocado no tempo, resultante de uma outra colisão. O pulso resultante, em roxo, mostra o efeito do *pile-up*, distorcendo o sinal. É esperado que esse efeito seja acentuado na fase II do LHC, com o aumento da luminosidade.

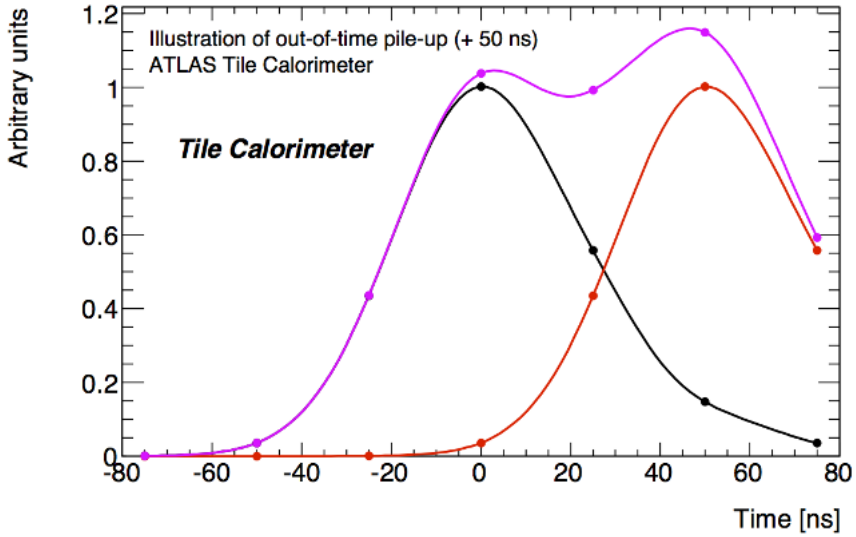

Figura 15 - Efeito do *pileup* (49)

Para exemplificar os efeitos do *pile-up* associados ao aumento da luminosidade, a Figura 16 apresenta dois gráficos. No primeiro, é mostrado o vetor com os valores de energia das colisões e suas respectivas posições no tempo. No segundo, é apresentada a saída do *shaper* correspondente a esses dados. Os sinais foram gerados pelo simulador de pulsos implementado em *Matlab*, cuja metodologia será descrita no capítulo posterior.

Os dados consideram dois regimes de ocupação, isto é, frações distintas de colisões presentes no intervalo analisado, de 95% e 20%. O maior nível de ocupação implica maior *pile-up*, e seu principal efeito é o aumento da flutuação e do deslocamento da linha de base na resposta do *shaper*. Assim, no segundo gráfico observa-se que, no início, a linha de base assume valores mais negativos. Após a transição da ocupação de 95% para 20%, o valor mínimo da resposta aumenta, evidenciando a influência do *pile-up* sobre a linha de base.

As Figuras 17 e 18 apresentam o pulso de saída do *shaper* para ocupações de 20% e 95%, respectivamente. Em ambos os casos, é possível observar o comportamento da linha de base, que se torna mais deslocada e com maior variação no regime de maior ocupação, evidenciando a influência do *pile-up*. Esse efeito reforça a necessidade de modelar adequadamente a resposta do *shaper* no simulador, o que motiva o uso de um filtro IIR para sua implementação.

O método de reconstrução de energia utilizado atualmente no *TileCal*, em sua maioria, é o OF2 (*Optimal Filtering 2*), que consiste em utilizar a função de autocorrelação do ruído das amostras digitais para determinar, de forma ótima, os pesos que minimizam a variância do estimador de energia. A técnica permite obter simultaneamente a origem temporal do pulso e sua amplitude, mesmo na presença de ruído correlacionado (50).

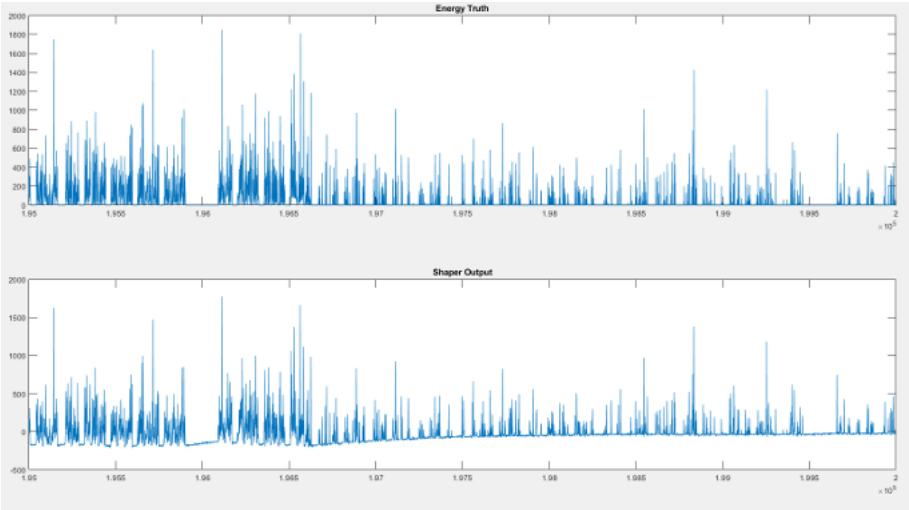

Figura 16 - Pulso de saída do *shaper* no simulador do *Matlab* para diferentes ocupações.

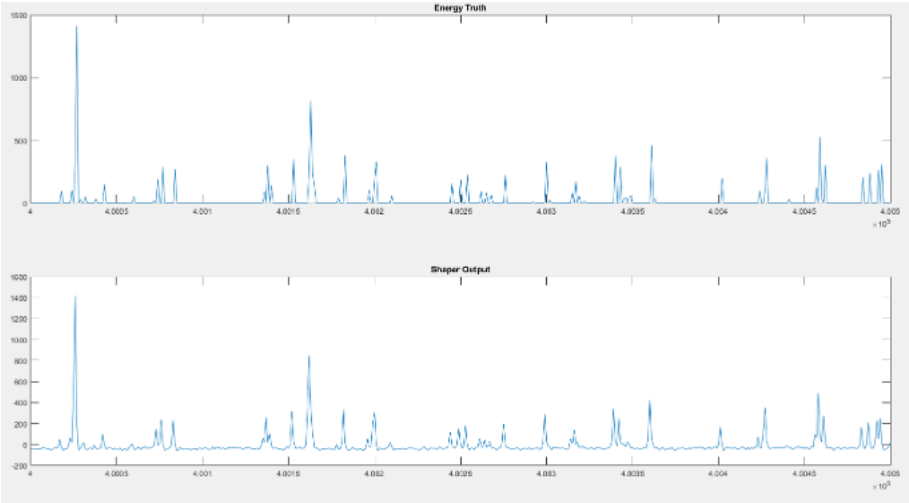

Figura 17 - Pulso de saída do *shaper* no simulador do *Matlab* para ocupação de 20%.

O OF2 utiliza um conjunto fixo de pesos aplicados a uma janela de sete amostras digitalizadas, que baseia-se, portanto, na combinação linear das amostras s_i do sinal digitizado, resultando em uma estimativa da amplitude A dada por:

$$A = \sum_{i=1}^{N} s_i \, w_i \tag{2.3}$$

Em que w_i são os pesos ótimos obtidos pela solução do sistema associado às condições de minimização da variância, que pode ser obtida a partir do matrix de covariância do sinal de ruído do *TileCal*, dada por:

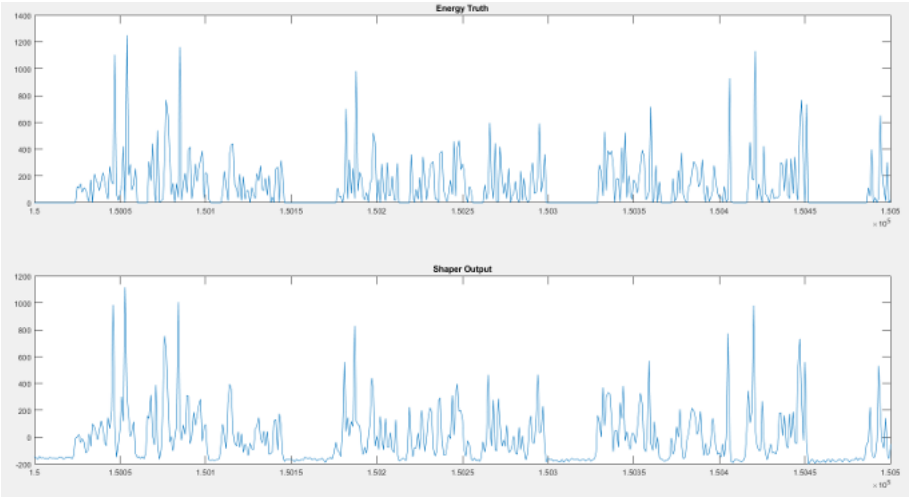

Figura 18 - Pulso de saída do *shaper* no simulador do *Matlab* para ocupação de 95%.

$$\text{var}(\hat{A}) = \mathbf{w}^T \, \mathbf{C} \, \mathbf{w} \tag{2.4}$$

O uso desses pesos permite reconstruir a energia depositada na célula com precisão superior à obtida por métodos tradicionais, ao mesmo tempo em que reduz o impacto do ruído eletrônico e do ruído induzido por *pile-up*.

Para a fase II, entretanto, vários métodos mais efetivos estão sendo testados, considerando o aumento do efeito do *pile-up*. Um deles é o método baseado no filtro *Wiener-Hopf*, que utiliza as propriedades estatísticas do sinal e do ruído para determinar um conjunto de pesos ótimos que estimam o sinal a partir de um conjunto de dados ruidosos (48).

Os pesos são calculados com o objetivo de minimizar o valor esperado do erro quadrático, dado por:

$$J = E[(\hat{A} - A)^2] \tag{2.5}$$

Com isso, o erro seria minimizado quando a derivada de J for igual a zero. A solução algébrica leva à seguinte equação:

$$\mathbf{wR} = \mathbf{p} \tag{2.6}$$

Ou,

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{p} \tag{2.7}$$

Em que **w** corresponde aos pesos do filtro, a matriz **R** é a matriz de autocorrelação para os sinais de entrada e o vetor **p** corresponde à matriz de correlação cruzada entre os sinais de entrada e o vetor dos valores de amplitude desejados.

A amplitude estimada pelo algoritmo proposto, no entanto, é dada por:

$$\hat{A} = \sum_{i=1}^{N} s_i w_i + w_{N+1} \tag{2.8}$$

Em que o vetor **s** corresponde às amostras recebidas do ADC e w_{N+1} é o *bias* que é subtraído de cada estimativa (48).

No entanto, os dados atuais do LHC ainda não reproduzem integralmente o regime de *pile-up* esperado no HL-LHC, o que motiva o uso intensivo de simulações dedicadas. O tema central desse trabalho é justamente o aprimoramento de um simulador de pulsos do *TileCal*, considerando os dados no formato da fase II, com todas as condições do *pile-up* para uma alta luminosidade. A atuação será especificamente na parte do *shaper*, que modela os pulsos no formato conhecido, com testes de diferentes tipos de filtros que tenham o melhor desempenho, visando balancear entre erro e eficiência computacional.

3 SIMULADOR DE PULSOS DO TILECAL

Nesta seção é apresentado o simulador de pulsos previamente desenvolvido e adaptado para este trabalho, cujo objetivo é testar técnicas de estimação de energia e detecção de partículas para o *TileCal* (51), incorporando o modelo temporal dos pulsos gerados nas colisões e a frequência nominal de operação de 40 MHz. O simulador permite gerar formas de onda sintéticas controladas, possibilitando avaliar o desempenho dos algoritmos de filtragem e reconstrução em condições equivalentes às observadas no detector.

Para alcançar esse objetivo, o simulador foi implementado em FPGA, uma tecnologia que permite processamento em altas frequências e execução determinística, características essenciais para aplicações associadas ao ambiente do LHC. Essa implementação possibilita treinar e validar novas técnicas de processamento digital em condições equivalentes às do detector, além de facilitar sua posterior integração ao sistema do colisor (51).

Entre essas tecnologias, destaca-se a implementação de diversos métodos de filtragem aplicados ao *shaper* do simulador, cuja arquitetura será apresentada nas seções seguintes. Foram considerados filtros convencionais (FIR e IIR), filtros na configuração *lattice* e abordagens baseadas em aritmética de ponto flutuante. Cada um desses métodos será discutido em detalhe ao longo do presente trabalho.

A primeira etapa do simulador é a atribuição das posições das colisões, por meio de um gerador de números pseudo-aleatórios; o próximo passo é determinar a energia para essas colisões, por meio de uma distribuição exponencial que respeita o padrão do *TileCal*; por fim, o simulador do *shaper* adiciona ruído e o efeito de *pile-up* para as colisões, como ocorre atualmente no *TileCal* (51). Tais etapas são mostradas na Figura 19.

Figura 19 - Esquema do Simulador

3.1 GERADOR DE NÚMEROS PSEUDO-ALEATÓRIOS

A atribuição da posição das colisões possui um passo muito importante por trás, que é a geração de números aleatórios, ou no caso do presente trabalho: pseudo-aleatórios. O método utilizado para isso é conhecido como LCG (*Linear Congruential Generator*).

O método destaca-se por sua combinação de baixo custo computacional, facilidade de implementação e boa adequação para sistemas digitais em FPGA (52). Por utilizar apenas operações aritméticas básicas — multiplicação, soma e módulo — o LCG pode

operar a altas frequências, acompanhando as exigências do ambiente de leitura do TileCal, que trabalha a uma taxa de amostragem de 40 MHz.

O LCG pertence à família dos geradores baseados em recorrências lineares modulares, cuja proposta é produzir uma sequência determinística com propriedades estatísticas suficientemente adequadas para aplicações de simulação, modelagem estocástica e teste de sistemas digitais (52).

A sequência pseudo-aleatório é definida por:

$$X_{n+1} = (aX_n + c) \bmod m \tag{3.1}$$

Em que:

- X_0: valor inicial.
- a: multiplicador.
- c: incremento.
- m: módulo.

X_n pertence ao intervalo $[0, m-1]$, podendo ser convertido a uma forma normalizada através de $U_n = X_n/m$. A escolha adequada dos parâmetros é essencial para garantir um bom período e evitar correlações indesejadas.

Apesar das limitações conhecidas em termos de correlação e qualidade estatística (já destacadas nos estudos clássicos de Park e Miller (53)), o LCG permanece uma escolha robusta para aplicações em que simplicidade, velocidade e reprodutibilidade são fatores determinantes — como no simulador apresentado neste trabalho.

Para o simulador desenvolvido, os parâmetros escolhidos foram (51):

- $a = 75$
- $c = 74$
- $m = 2^{16} + 1$

No código mostrado abaixo, um exemplo de LCG foi desenvolvido em *Python*, com os parâmetros do presente trabalho. A Figura 20 apresenta dois gráficos: (i) a sequência temporal dos primeiros 100 valores gerados pelo LCG e (ii) o histograma correspondente. A partir desses resultados é possível discutir aspectos fundamentais sobre a aleatoriedade produzida pelo método.

```python
import matplotlib.pyplot as plt
import numpy as np

def lcg(seed, a, c, m):
    x = seed
    while True:
        x = (a * x + c) % m
        yield x

a=75, c=74, m=2**16+1, seed=42

gen = lcg(seed, a, c, m)
x_vals = np.array([next(gen) for _ in range(100)], dtype=float)

fig, axes = plt.subplots(2, 1, constrained_layout=True)

axes[0].plot(range(100),x_vals,marker="o",linestyle="-",linewidth=1,markersize=3,)
axes[0].set_xlabel(r"Índice $n$")
axes[0].set_ylabel(r"$X_n$")
axes[0].set_title(r"Sequência dos 100 primeiros valores do LCG")

axes[1].hist(x_vals,bins=10,edgecolor="black",)
axes[1].set_xlabel(r"Valor $X_n$")
axes[1].set_ylabel("Contagem")
axes[1].set_title(r"Distribuição dos valores gerados")
```

No primeiro gráfico da Figura 15 é possível observar uma ausência de periodicidade e uma variação irregular entre as amostras consecutivas, o que indica que, no domínio temporal, o gerador não apresenta correlações triviais perceptíveis por inspeção visual, o que é desejável para o simulador. O histograma mostra que a distribuição é praticamente uniforme, o que indica um bom gerador pseudoaleatório.

Os geradores lineares congruenciais (LCGs) têm sido historicamente utilizados em uma ampla variedade de aplicações, incluindo simulações físicas e estatísticas (53), métodos de Monte Carlo (52), modelagem estocástica (54) e sistemas computacionais embarcados (55). Sua simplicidade e eficiência computacional justificaram seu uso extensivo em diversas áreas da ciência e engenharia.

3.1.1 Aplicação em *FPGA*

A aplicação e o teste desse método em FPGA foram realizados utilizando o *Quartus*, *software* desenvolvido pela *Intel*. Ele permite a programação em linguagens do tipo HDL (*Hardware Description Language*). Para o desenvolvimento do simulador e das demais

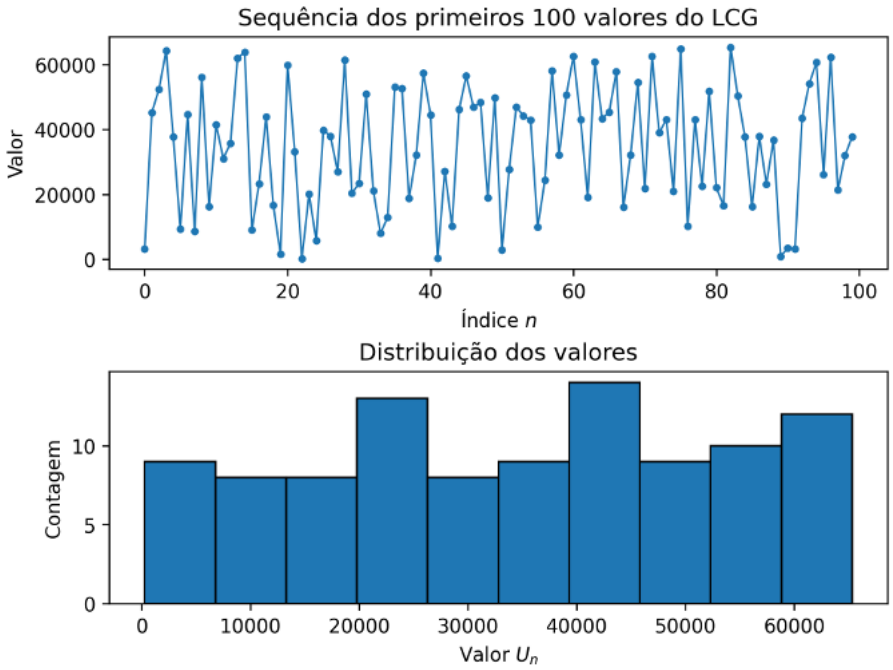

Figura 20 - Exemplo de implementação do LCG

etapas deste trabalho, foi escolhida a linguagem *Verilog* (*Verilog Hardware Description Language*). O *Quartus* será utilizado não apenas nesta etapa, mas também em todas as demais fases do projeto.

O circuito desenvolvido no *Quartus* pode ser interpretado como um bloco, mostrado na Figura 21. Ele recebe como entrada um sinal de *clock* — responsável por sincronizar todo o sistema e elemento fundamental em circuitos digitais — e um sinal de *reset*, que permite restaurar o circuito aos seus valores iniciais. Sua saída, de 7 bits, corresponde ao número aleatório desejado, resultante da Equação 3.1 (51).

Dentro do bloco foi implementado um registrador de 32 bits, utilizado para armazenar o deslocamento dos dados mostrado na Equação 3.1. A saída desse registrador possui 32 bits; no entanto, a saída do bloco, como mencionado anteriormente, possui apenas 7 bits, que correspondem ao valor de interesse para obter os 128 estados aleatórios, valor arbitrário escolhido para equilibrar a resolução do sistema e o desempenho computacional (51). Esse processo é realizado por meio do truncamento, isto é, do descarte dos bits menos significativos.

O código mostrado abaixo ilustra a implementação desse circuito em *Verilog*, no *Quartus*.

```
1 module rand128
2 #(
3     parameter a = 75,
```

```
4       parameter c = 74,
5       parameter m = 2**16 + 1
6   )
7   (
8       input clk, rst,
9       output [6:0] rand_out
10  );
11
12  reg [31:0] x;
13
14  always @(posedge clk or posedge rst)
15  begin
16  if (rst) begin
17          x <= 32'b0;
18      end
19  else begin
20          x <= (a*x + c);
21      end
22  end
23
24  assign rand_out = x[31:25];
25
26  endmodule
```

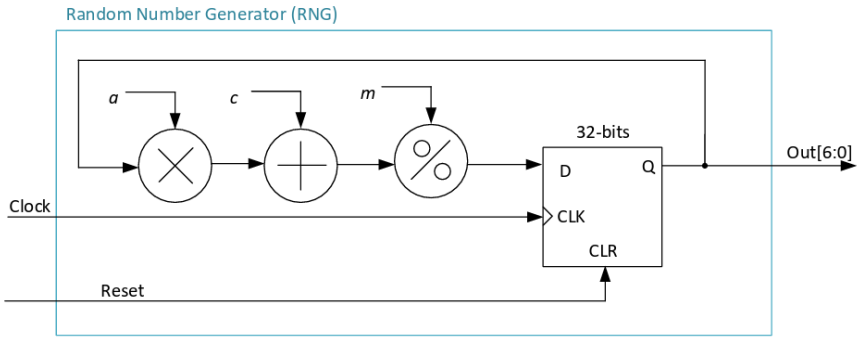

Figura 21 - Gerador de números aleatórios (51)

3.2 SIMULAÇÃO DOS EVENTOS DA COLISÃO

O segundo passo desenvolvido no simulador é determinar a posição das colisões e a energia delas, e ambas as etapas utilizam o método de geração de números pseudo-aleatórios

descritos na seção anterior.

3.2.1 Determinação da Posição das Colisões

Como mencionado no capítulo anterior, as colisões do LHC ocorrem a uma frequência de 40 MHz, isto é, a cada 25 ns. Elas são aleatórias e cada colisão é completamente descorrelacionada da anterior, o que implica a necessidade de reproduzir essa característica no simulador. Essa descorrelação é assegurada pelo método de geração de números pseudo-aleatórios, conforme explicado na seção anterior.

O circuito do LCG desenvolvido é utilizado para atribuir a posição das colisões, juntamente com outro fator importante: o número da ocupação. Este é basicamente o valor da quantidade de colisões em um período determinado, por exemplo, considerando um vetor com 100 posições e com uma taxa de ocupação de 50%, a quantidade de colisões atribuídas deve ser de aproximadamente 50 (51). Como o número aleatório gerado é limitado em 128 (7 bits), o valor da ocupação é dado por:

$$occ = p(\%) \cdot \frac{128}{100} \tag{3.2}$$

Em que p é justamente a porcentagem de ocupação desejada.

A lógica por trás da determinação das posições é simples: gera-se um número aleatório entre 1 e 128. Se a ocupação for de 50%, o valor do limiar de comparação será 64, conforme a Equação 3.2. Assim, caso o número aleatório seja menor que o limiar, considera-se que ocorreu uma colisão naquele ponto; caso contrário, não (51).

O código abaixo exemplifica esse método em *Python*, de uma função que determina se houve colisão ou não, ponto a ponto.

```
1     def hits_position(p, x):
2         occ = p*128/100
3         return 1 if (x < occ) else 0
```

A Figura 22 apresenta dois gráficos: o primeiro mostra os números aleatórios gerados em um vetor de 100 posições, juntamente com o limiar (em vermelho) definido para uma ocupação de 50% ($occ = 64$). Dessa forma, é possível visualizar quais valores estão abaixo do limiar, indicando que houve uma colisão nesses pontos. O segundo gráfico evidencia isso ao exibir um vetor com valores unitários exatamente nas posições onde as colisões ocorreram.

O código abaixo em *Verilog* ilustra o circuito desenvolvido.

```
1     module simul
2     (
```

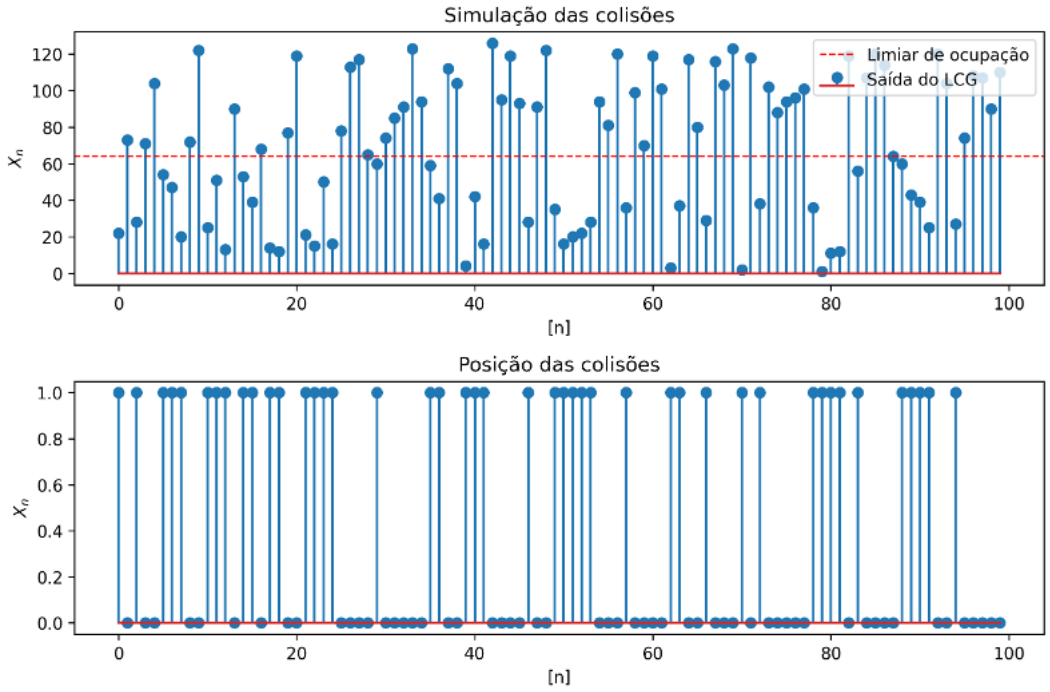

Figura 22 - Determinação da posição das colisões

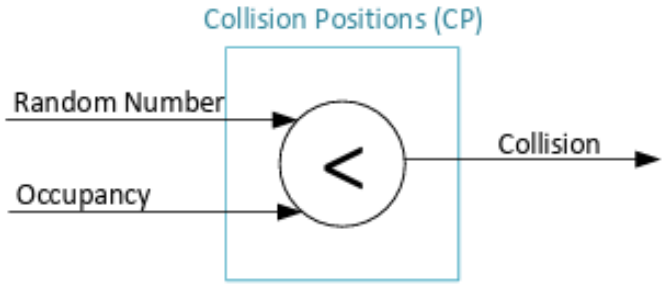

Figura 23 - Bloco em FPGA para a posição das colisões (51)

```
3               input clk, rst,
4               input [6:0] occ,
5               output reg [9:0] hit
6       );
7       rand128 rand128(clk, rst, rand_out);
8
9       always @ (*) begin
10          if (rand_out < occ)
11                  hit <= 10'd1;
```

```
12          else
13                  hit <= 10'd0;
14      end
15      endmodule
```

3.2.2 Determinação da Energia das Colisões

A determinação da energia das colisões foi feita a partir de um método probabilístico baseado na função de distribuição cumulativa (CDF, do inglês *Cumulative Distribution Function*) e na sua inversa. O objetivo desse método é garantir que os dados simulados sigam uma distribuição a mesma estatística dos dados reais observados no *TileCal*, que seguem uma distribuição exponencial (56).

A CDF é o objeto matemático que liga diretamente a variável aleatória X às probabilidades associadas a seus valores. Formalmente, para uma variável aleatória X, a CDF é definida por (57):

$$F_X(x) = P(X \leq x), \tag{3.3}$$

ou seja, $F_X(x)$ fornece a probabilidade acumulada que o valor observado de X seja menor ou igual a um dado limiar x. Em muitos problemas de simulação, a CDF e sua inversa são ferramentas centrais para gerar amostras que seguem uma distribuição desejada a partir de números pseudoaleatórios uniformes (58).

A distribuição exponencial com média μ possui função de distribuição cumulativa dada por (57):

$$F(x) = 1 - e^{-x/\mu},\ x \geq 0 \tag{3.4}$$

Tal expressão decorre da definição clássica da distribuição exponencial com parâmetro $\lambda = 1/\mu$ (57).

Como o objetivo é determinar o valor da energia com base em sua probabilidade, deve-se usar a função inversa da CDF, que para a exponencial é:

$$x = F^{-1}(u) = -\mu \ln(1 - u) \tag{3.5}$$

Logo, a Equação 3.5 fornece a base para determinar a energia das colisões para o simulador do *TileCal*.

Como exemplo, foi desenvolvido um programa em *Python*. Nele, um número aleatório u é gerado e, a partir da Equação 3.5, obtém-se um valor de energia que segue uma distribuição exponencial com média μ, de acordo com sua função de probabilidade. A Figura 24 apresenta o histograma da variável x gerada pelo programa, evidenciando

que a distribuição resultante corresponde à distribuição exponencial obtida por meio da inversa da CDF.

```
MEDIA = 100
N = 10000
MAX_VALUE = 1023

#Variável aleatória
u = np.random.rand(N)

#Valor da energia
x = -MEDIA*np.log(1 - u)
```

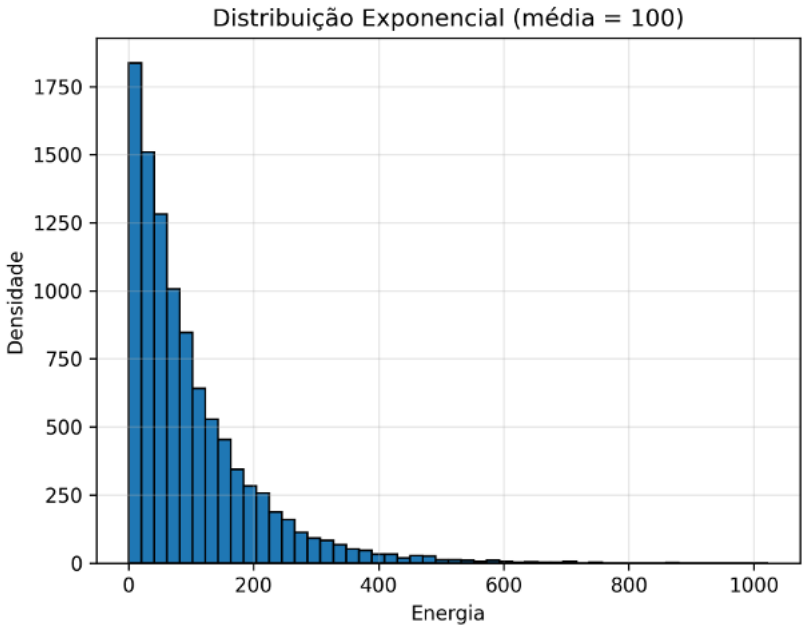

Figura 24 - Distribuição exponencial com média $\mu = 100$

A implementação desse método em FPGA torna-se inviável, visto que um circuito capaz de calcular logaritmos a cada borda de *clock* exigiria um esforço computacional muito elevado. Para contornar esse problema, foi utilizada uma LUT (*Look-Up Table*), que armazena valores de energia que obedecem a uma distribuição exponencial (51).

A LUT funciona como uma memória, em cada endereço ela armazena o valor da energia x, no intervalo:

$$x \in [0, 1023] \tag{3.6}$$

Como o circuito de número aleatório do LCG gera números de 7 bits, a LUT deve conter 128 posições, e é justamente ele que vai determinar qual energia será sorteada. Os valores de energia armazenados na LUT possuem 10 bits pois este é um valor suficiente para realizar operações de ponto fixo sem perder informações de bits (51).

Portanto, o circuito desenvolvido em FPGA possui um entrada de 7 bits para receber o número aleatório do LCG, um pino de *clock* e outro de *reset*, e uma saída com o valor da energia, de 10 bits. Esse processo é mostrado na Figura 25, como um bloco de FPGA.

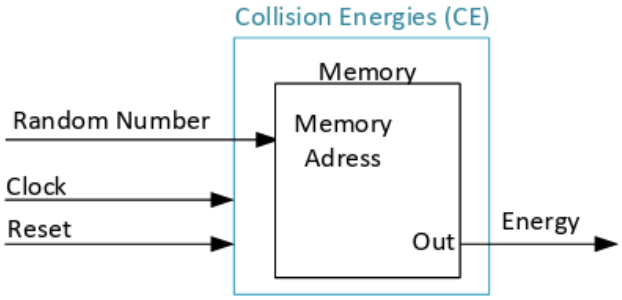

Figura 25 - Bloco para determinar a energia das colisões (51)

O código abaixo, em *Verilog*, implementado no simulador, mostra como foi feito esse passo, ele é uma versão aprimorada do código para determina a posição das colisões.

```
1   module simul
2   (
3           input clk, rst,
4           input [6:0] occ,
5           output reg [9:0] hit
6   );
7
8   wire [6:0] rand_out, addr_out;
9
10  reg [9:0] mem [0:127];
11
12  reg [9:0] exp = 0;
13
14  initial begin
15          $readmemb("exponencial.mif", mem);
16  end
```

```
17
18      rand128 rand128(clk, rst, rand_out);
19
20      rand128 #(32'ha92fe137) mem_addr(clk, rst, addr_out);
21
22      always @ (posedge clk) exp <= mem[addr_out];
23
24      always @ (*) begin
25              if (rand_out < occ)
26                      hit <= exp;
27              else
28                      hit <= 10'd0;
29      end
30      endmodule
```

A LUT contendo os valores da distribuição exponencial é previamente gerada de forma *offline* e armazenada no arquivo `exponencial.mif`. No bloco `initial`, o comando `$readmemb` carrega esse arquivo para a memória interna `mem`, populando todas as 128 posições da *ROM*.

O módulo `rand128` é então instanciado, produzindo a cada ciclo um endereço pseudoaleatório entre 0 e 127. Esse endereço é utilizado para acessar a LUT: na borda de subida do *clock*, o valor correspondente é lido da memória e armazenado no registrador `exp`, que representa a energia simulada para aquela colisão.

O bloco lógico responsável por determinar as posições das colisões opera de maneira análoga ao descrito na seção anterior. A diferença fundamental é que, em vez de atribuir um valor unitário quando uma colisão é detectada, o sistema utiliza o valor armazenado no registrador `exp`, proveniente da LUT exponencial. Dessa forma, cada colisão é associada a uma energia simulada, e não apenas a um indicador binário de ocorrência.

A Figura 26, gerada pelo autor, mostra a forma de onda gerada com auxílio do *sofware Questa*, simulador do *Quartus*, mostrando as posições em que houveram colisões, juntamente com a energia atribuída, resultante do código mostrado acima.

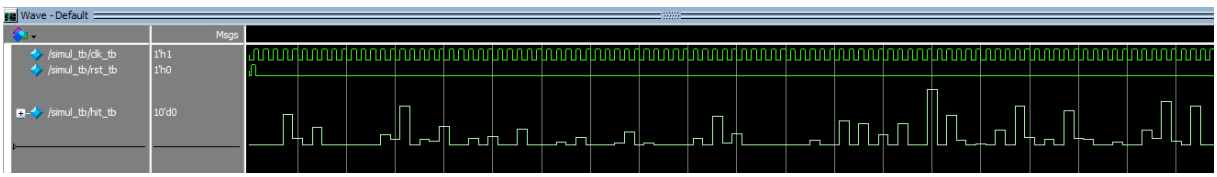

Figura 26 - Colisões do simulador com energia

3.3 O *SHAPER* DO *TILECAL*

3.3.1 A Função de Transferência do *shaper*

Como mencionado na Seção 2.5.2, o *shaper* do *TileCal* é responsável pelo condicionamento do sinal analógico, de acordo com o circuito de conformação conhecido como *7-Poles Shaper*. Esse circuito é composto por duas cadeias de amplificação que operam em paralelo, denominadas *High Gain* e *Low Gain*. Em condições normais de operação, a cadeia de alto ganho é utilizada, pois permite melhor resolução para sinais de baixa amplitude. No entanto, caso a amplitude do sinal ultrapasse o valor máximo suportado pelo conversor analógico-digital (ADC), a cadeia de baixo ganho passa a ser utilizada, evitando a saturação do sistema (59).

A Figura 27 mostra o circuito de conformação do sinal, e pode ser interpretado como um filtro de Bessel (60), que tem a função de adequar a banda de frequência da resposta da resposta em frequência do pulso de entrada (sinal da PMT). O pulso é alongado, permitindo sua operação em 40 MHz, frequência do LHC (60).

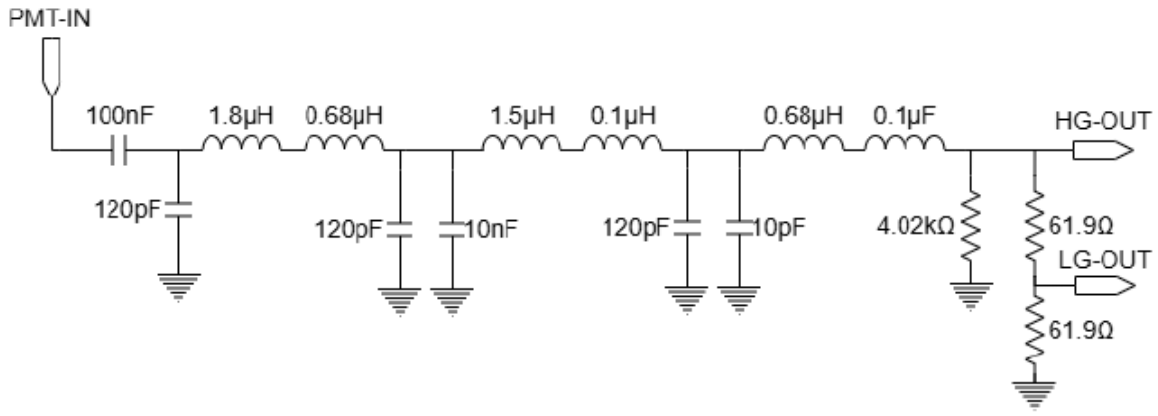

Figura 27 - Circuito de condicionamento do pulso da PMT (59).

A função de transferência do *shaper* pode ser representada de acordo com a Equação 3.7, e sua resposta ao impulso está mostrada na Figura 28 (59).

$$H(s) = \frac{1.80 \cdot 10^7 s - 5.92 \cdot 10^{14}}{s^2 + 5.04 \cdot 10^7 s + 1.63 \cdot 10^{16}} - \frac{7.21 \cdot 10^4}{s + 7.20 \cdot 10^4} - \frac{7.98 \cdot 10^7 s - 2.11 \cdot 10^{14}}{s^2 + 6.48 \cdot 10^7 s + 4.05 \cdot 10^{15}} + \frac{6.19 \cdot 10^7}{s + 6.28 \cdot 10^7} \tag{3.7}$$

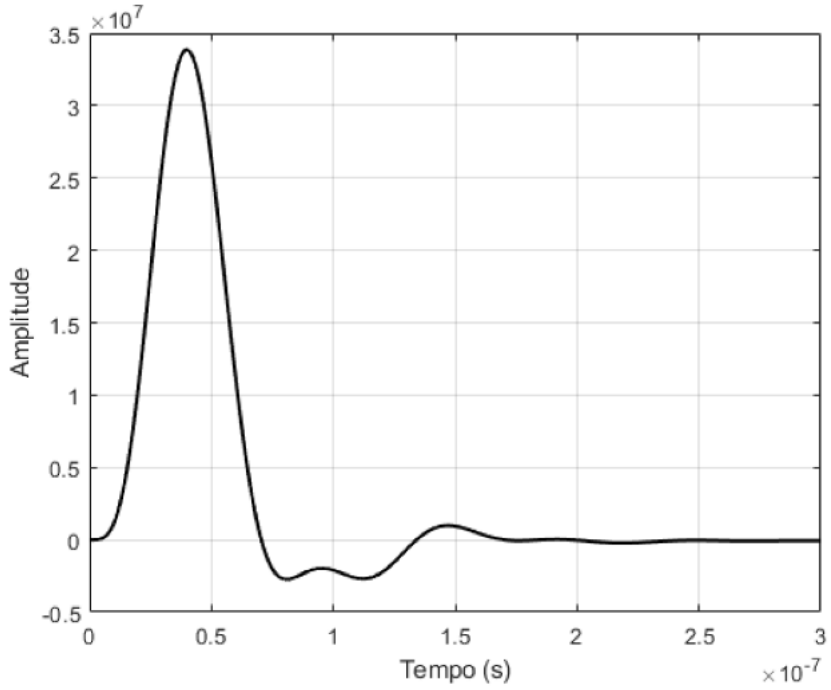

Figura 28 - Resposta ao impulso da função de transferência do *shaper* (59)

3.3.2 Digitalização da Função de Transferência

A função de transferência digitalizada do *shaper* permite a implementação do circuito analógico da Figura 23 em sistemas digitais, tais como FPGA, que é o objetivo do presente trabalho. Tal função de transferência é mais precisamente sintetizada por um filtro IIR (*Infinite Impulse Response*), não sendo necessária a implementação de filtros FIR (*Finite Impulse Response*) (59).

No simulador desenvolvido por F. C. Luna *et al.* (51), foi empregado um filtro FIR para essa finalidade. Entretanto, a função de transferência apresentada na Figura 24 possui uma componente exponencial de amplitude muito baixa e longa duração, a qual influencia significativamente as análises que envolvem altas taxas de ocupação das células.

A implementação de um filtro FIR capaz de representar adequadamente essa componente, como realizado anteriormente (51), implicaria em um número elevado de coeficientes, aumentando substancialmente a complexidade do projeto e tornando impraticável sua implementação em FPGA. Nesse contexto, o uso de um filtro IIR mostra-se uma alternativa mais viável para o simulador, pois permite um melhor compromisso entre a fidelidade da resposta ao impulso e a economia de recursos computacionais do sistema no qual é aplicado (59).

Um dos métodos amplamente utilizados para a digitalização de funções de trans-

ferência analógicas é o *Impulse Invariant Method* (62, 63). Esse método baseia-se na correspondência direta entre as respostas ao impulso de sistemas contínuos e digitais, garantindo a amostragem do sistema analógico seja igual à sequência de amostras do sistema digital (59). Tal transformação é obtida de acordo com a Equação 3.8 (64)

$$H(z) = \sum_{k=1}^{N} \frac{A_k(1 - e^{p_k T})}{1 - e^{p_k T} z^{-1}} \tag{3.8}$$

em que T é o período de amostragem do sistema discreto e p_k representam os polos da função de transferência contínua. Essa relação garante que a resposta ao impulso do sistema digital corresponda a uma versão amostrada da resposta ao impulso do sistema analógico (59).

Considerando a frequência de operação do LHC de $f = 40\ MHz$, o período de amostragem é dado por:

$$T = \frac{1}{f} = \frac{1}{40\ MHz} = 25\ ns \tag{3.9}$$

Logo, a função de transferência discreta é obtida através dos polinômios complexos do circuito de conformação e está representada na Equação 3.10, com o período de amostragem de 25 ns (59).

$$\begin{aligned} H(z) = \frac{5.32 \cdot 10^{-1} + 2.82 \cdot 10^{-1} z^{-1}}{1 + 1.07 \cdot z^{-1} + 2.84 \cdot 10^{-1} z^{-2}} - \frac{2.36 + 8.65 \cdot 10^{-1} z^{-1}}{1 - 1.77 \cdot 10^{-1} z^{-1} + 1.98 \cdot 10^{-1} z^{-2}} - \\ - \frac{5.32 \cdot 10^{-1}}{1 - 9.98 \cdot 10^{-1} z^{-1}} + \frac{1.83 \cdot 10^{0}}{1 - 2.08 \cdot 10^{-1} z^{-1}} \end{aligned} \tag{3.10}$$

A Figura 29 apresenta a resposta ao impulso digitalizada em comparação com a resposta original. Observa-se que, ao considerar a frequência de amostragem de 40 MHz, houve sincronismo na digitalização; contudo, não foi possível amostrar o pico máximo da resposta ao impulso original. Em calorimetria, esse sincronismo com o instante de pico do sinal é de extrema importância, pois permite a correta determinação do tempo de voo das partículas (59).

Uma forma de contornar esse problema, feita por Paschoalin *et al.* (59), consiste na aplicação de um atraso no domínio do tempo à função de transferência contínua, de modo que, posteriormente, seja realizada sua transformação para um sistema discreto. Esse atraso pode ser determinado a partir da diferença entre o instante em que ocorre o pico do pulso (t_p), no domínio contínuo, e o instante correspondente à amostra $n = 2$, que equivale a 50 ns. O atraso no domínio do tempo pode ser implementado por meio da concatenação da função de transferência original com o termo exponencial e^{-ds}.

Entretanto, a aplicação direta da exponencial no domínio S resulta em uma função de transferência não linear, o que a torna impraticável para implementação direta em

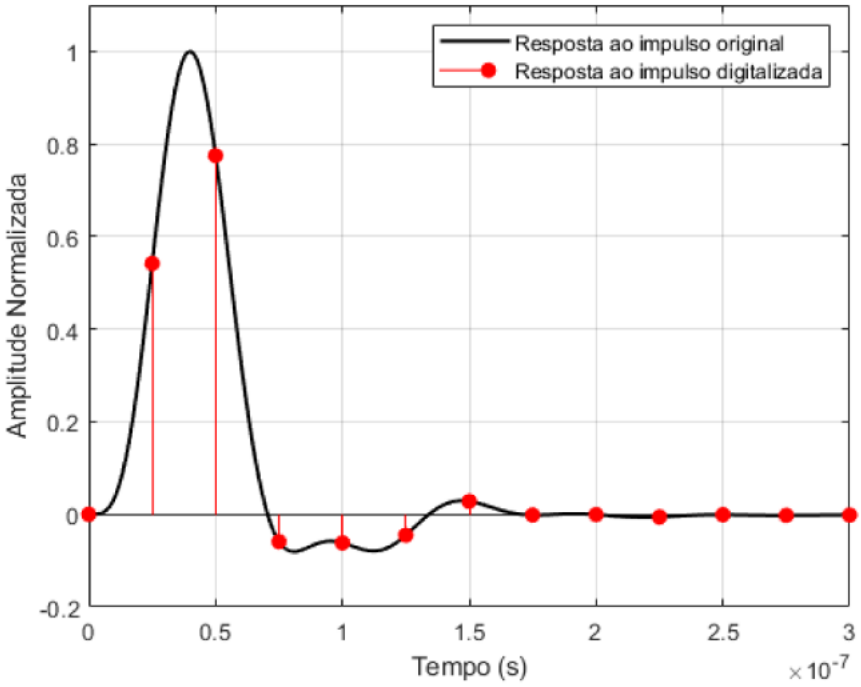

Figura 29 - Comparação da resposta ao impulso digitalizada com a original (59)

hardware. Dessa forma, propõe-se o uso da aproximação de Padé de primeira ordem, de modo a preservar a característica polinomial no domínio Z e a linearidade da implementação em hardware, conforme apresentado na Equação 3.11.

$$e^{-d \cdot s} \approx \frac{2 - d \cdot s}{2 + d \cdot s} \tag{3.11}$$

Conforme foi feito por Paschoalin *et al* (59), o atraso proposto foi aplicado na função de transferência completa do sistema, adicionando apenas um polo no sistema antes da digitalização. Outra opção seria aplicar o atraso em cada componente da função de transferência contínua, porém resultaria na inclusão de mais 12 polos. A primeira opção foi a mais viável e utilizada.

Ao aplicar o atraso e realizar a transformação, a função de transferência digital mostrada na Equação 3.12 é obtida, e sua resposta ao impulso é mostrada na Figura 30, dessa vez, o pico máximo da função original é amostrado corretamente.

$$\begin{aligned} H(z) = \frac{5.37 \cdot 10^{-1} + 2.88 \cdot 10^{-1} z^{-1}}{1 + 1.07 \cdot z^{-1} + 2.84 \cdot 10^{-1} z^{-2}} - \frac{3.81 + 3.64 \cdot 10^{-1} z^{-1}}{1 - 1.77 \cdot 10^{-1} z^{-1} + 1.98 \cdot 10^{-1} z^{-2}} - \\ - \frac{2.10 \cdot 10^{-2}}{1 - 9.98 \cdot 10^{-1} z^{-1}} + \frac{3.53 \cdot 10^{0}}{1 - 2.08 \cdot 10^{-1} z^{-1}} - \frac{2.59 \cdot 10^{-1}}{1 - 7.10 \cdot 10^{-2} z^{-1}} \end{aligned} \tag{3.12}$$

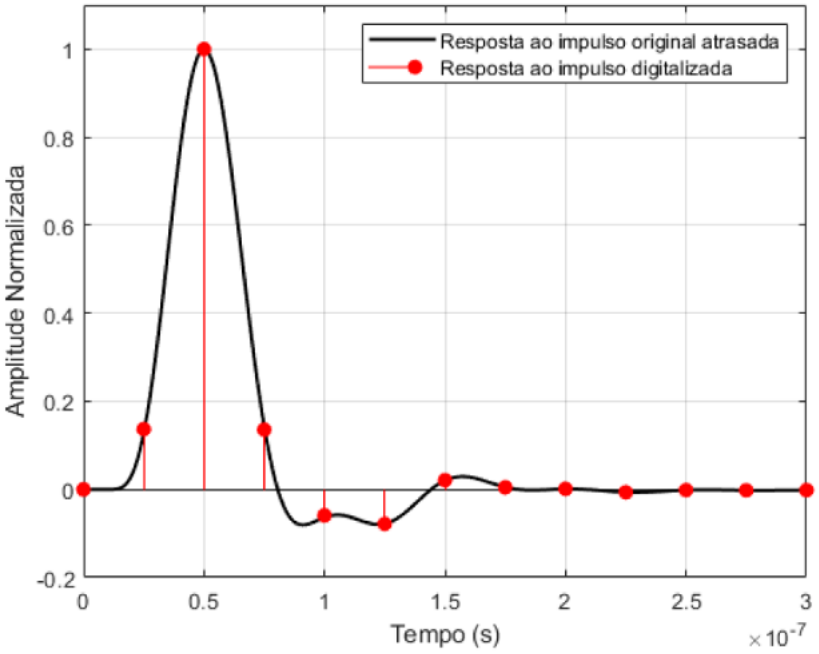

Figura 30 - Comparação da resposta ao impulso digitalizada com a original atrasada (59)

A função de transferência digital, mostrada na Equação 3.7, possui 5 componentes, cada um pode ser visto como um filtro IIR a ser implementado. Com isso, o simulador de pulsos pode ser descrito como a soma dos 5 filtros IIR de primeira ou segunda ordem (59). Os filtros são representados de acordo com as equações a seguir.

$$\begin{aligned} y_1[n] = 5.37 \cdot 10^{-1} \cdot x[n] + 2.88 \cdot 10^{-1} \cdot x[n-1] - \\ -1.07 \cdot y_1[n-1] - 2.84 \cdot 10^{-1} \cdot y_1[n-2] \end{aligned} \tag{3.13}$$

$$\begin{aligned} y_2[n] = -3.81 \cdot 10^{0}\, x[n] - 3.64 \cdot 10^{-1}\, x[n-1] \\ + 1.77 \cdot 10^{-1}\, y_2[n-1] - 1.98 \cdot 10^{-1}\, y_2[n-2] \end{aligned} \tag{3.14}$$

$$y_3[n] = -2.10 \cdot 10^{-2}\, x[n] + 9.98 \cdot 10^{-1}\, y_3[n-1] \tag{3.15}$$

$$y_4[n] = 3.53 \cdot 10^{0}\, x[n] + 2.08 \cdot 10^{-1}\, y_4[n-1] \tag{3.16}$$

$$y_5[n] = -2.59 \cdot 10^{-1}\, x[n] + 7.10 \cdot 10^{-2}\, y_5[n-1] \tag{3.17}$$

Logo, a saída do filtro IIR resultado é dado pela Equação (3.18).

$$y[n] = y_1[n] + y_2[n] + y_3[n] + y_4[n] + y_5[n] \tag{3.18}$$

Considerando uma sequência de 3 impulsos, a saída é mostrada de acordo com a Figura 31, em que é possível observar a formação do pulso característico do *TileCal* e o efeito de *pileup*.

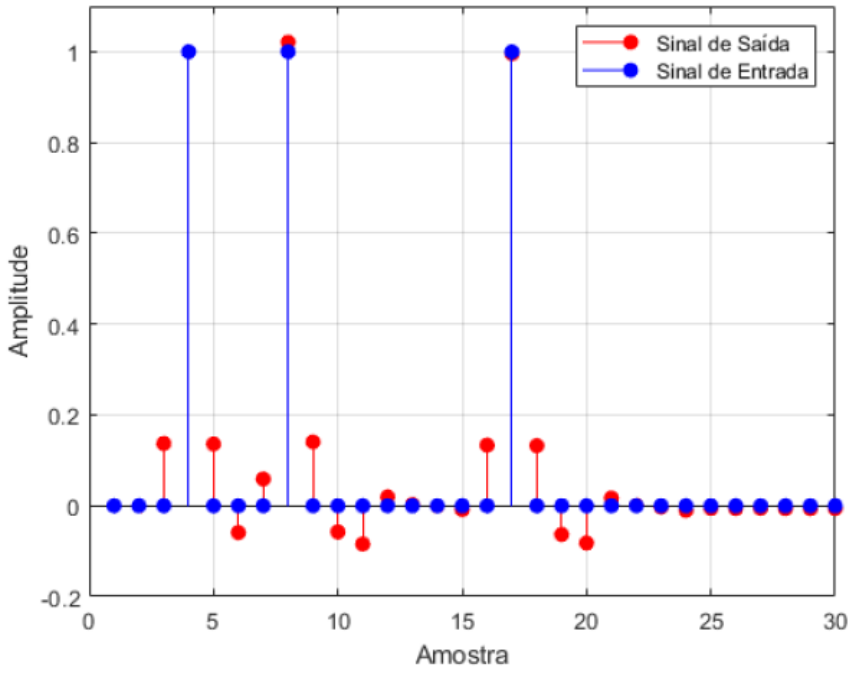

Figura 31 - Sinais de entrada $x[n]$ e de saída $y[n]$ do filtro IIR aplicado com a função digitalizada (59)

A implementação dos filtros em FPGA, conforme mostrada por Paschoalin *et al.* (59), requer uma atenção maior, visto que as FPGAs operam nativamente com aritmética de números inteiros. Uma forma amplamente utilizada para representar os coeficientes de filtros digitais é a aritmética de ponto fixo no formato $QX.Y$, em que X corresponde ao número de bits da parte inteira e Y ao número de bits da parte fracionária. Dessa forma, torna-se necessário analisar cuidadosamente os coeficientes dos filtros IIR, bem como as operações aritméticas envolvidas, a fim de determinar corretamente a quantidade de bits alocada para cada parte do número. Uma escolha inadequada dessa representação pode resultar em divergências na resposta obtida, além de comprometer a estabilidade do filtro IIR implementado (59).

Conforme escolhido por Paschoalin *et al.* (59), o ganho de entrada foi de 2^{32} e o ganho dos pesos foi de 2^{10} utilizando os formatos Q1.31 na entrada e Q1.48 na saída; os

mesmos valores foram adotados para o presente trabalho. As Figuras 32 e 33 mostram o bloco de operação em FPGA para filtros de primeira e segunda ordem, respectivamente.

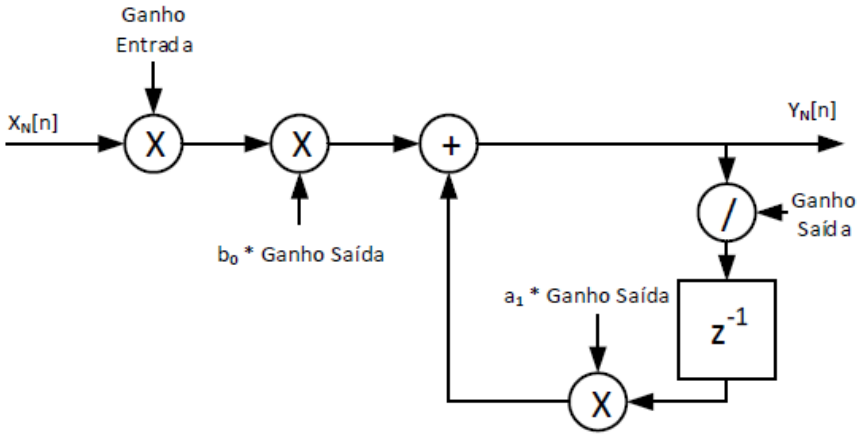

Figura 32 - Diagrama de blocos para implementação do filtro IIR de primeira ordem em FPGA (59)

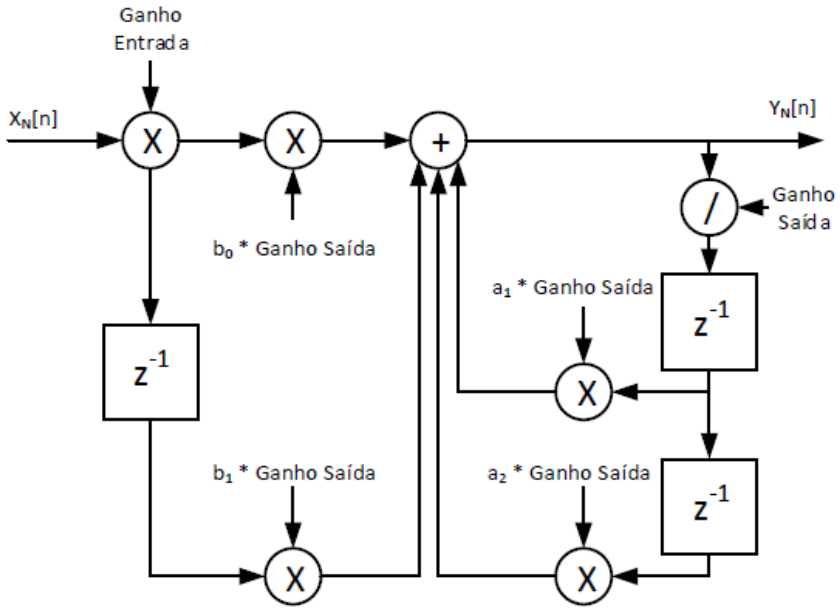

Figura 33 - Diagrama de blocos para implementação do filtro IIR de segunda ordem em FPGA (59)

A Figura 34 mostra a resposta ao impulso implementada em FPGA (59), o que indica que mesmo com a operação de ponto fixo, aproximando a resposta do circuito

através de uma função de transferência digitalizada, o resultado da simulação teve um valor muito bom, sendo muito próximo do sistema original.

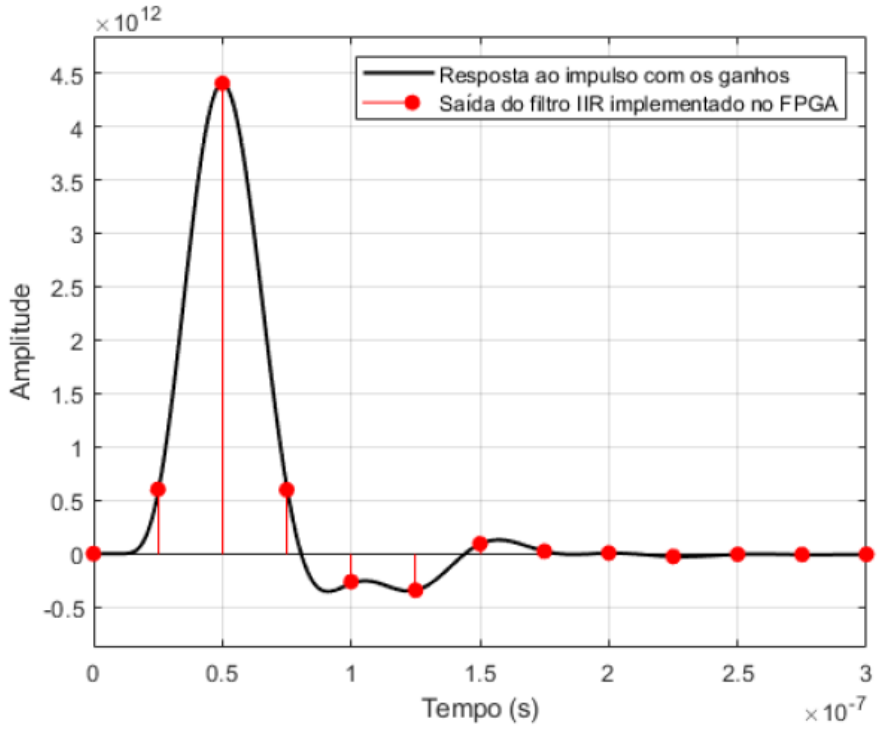

Figura 34 - Saída do filtro IIR implementada em FPGA comparada com a resposta ao impulso atrasada (59)

Com o *shaper* implementado, o simulador está completo, e sua estrutura em diagrama de blocos está representada na Figura 35 (51), que mostra uma versão simplificada dos blocos que foram implementados e previamente explicados. Nela é possível observar o bloco PLL para a geração do *clock*, dois blocos para geração de números aleatórios, que são atribuídos à geração das posições das colisões e suas energias, e por fim, o *shaper*.

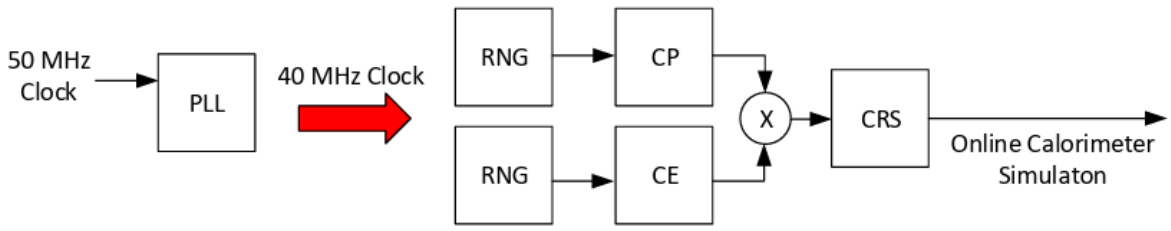

Figura 35 - Diagrama de blocos do simulador do calorímetro (51)

4 MÉTODOS DE IMPLEMENTAÇÃO DO *SHAPER*

O presente capítulo apresenta o objetivo central deste trabalho, que consiste na aplicação e avaliação de diferentes métodos de filtragem digital empregados no estágio de *shaper* do sistema de leitura do *TileCal*. O foco está na fundamentação teórica dos métodos considerados, bem como na descrição detalhada das arquiteturas digitais e das estratégias de implementação adotadas em FPGA.

Todos os métodos avaliados têm como finalidade a realização da mesma função de filtragem, diferindo entre si principalmente na forma de implementação digital e no tratamento numérico adotado. Dessa forma, busca-se investigar como diferentes arquiteturas, incluindo uma implementação IIR convencional quantizada, uma realização baseada no método *lattice* e uma abordagem utilizando aritmética de ponto flutuante, impactam aspectos como estabilidade numérica, complexidade arquitetural e viabilidade para operação em tempo real.

Ressalta-se que este capítulo tem caráter essencialmente metodológico e descritivo, não sendo priorizada a análise quantitativa dos resultados obtidos, a qual será apresentada e discutida em capítulo posterior.

4.1 IMPLEMENTAÇÕES RECENTES DE FILTROS IIR EM FPGA

Uma prática recorrente em implementações modernas de filtros IIR em FPGA é a decomposição do filtro em uma cascata de seções de segunda ordem, conhecidas como biquads ou SOS. Essa estratégia permite que cada seção seja implementada como uma pequena estrutura recursiva, com poucos estados internos, facilitando o controle da faixa dinâmica e a validação numérica. Além disso, essas estruturas favorecem o mapeamento em blocos DSP dedicados e contribuem para a redução do caminho crítico — sequência de operações lógicas consecutivas que determina o menor período de clock viável e, portanto, a frequência máxima de operação do circuito. Um exemplo clássico dessa abordagem é apresentado no *white paper* da Xilinx sobre estruturas IIR em FPGA, que discute diferentes formas de realização e destaca o uso de SOS como alternativa robusta para filtros de ordem elevada (67).

Outra linha bastante explorada na literatura busca aumentar o *throughput* do sistema, contornando as limitações de *pipelining* impostas pela realimentação dos filtros IIR. Para isso, são empregadas arquiteturas paralelas e técnicas como *look-ahead* e *two-level pipeline*. Datta e Dutta propõem uma implementação reconfigurável em FPGA na qual o comportamento IIR é obtido a partir de uma estrutura baseada em FIR, combinando paralelismo e *pipeline*. Os autores comparam essa solução com implementações IIR mais diretas e reportam ganhos em frequência de operação e eficiência de recursos em uma plataforma Xilinx Virtex-5 (68).

Há também trabalhos que concentram esforços na otimização do núcleo aritmético do filtro, uma vez que multiplicações e somas dominam o consumo de área, atraso e potência. Nessa abordagem, a estrutura do filtro permanece próxima à forma direta, porém com unidades aritméticas escolhidas ou customizadas para reduzir custo e latência. Eddla e Pappu apresentam a arquitetura AM-CSA-IIR, que combina um multiplicador do tipo *array multiplier* com um somador *carry-skip*, além da inserção de registradores para encurtar o caminho crítico. O trabalho avalia o desempenho da implementação em termos de LUT, FF, atraso e potência para diferentes dispositivos FPGA (69).

Uma alternativa frequentemente associada à robustez numérica é o uso de estruturas *lattice* e suas variações *wave digital*. Essas realizações são atrativas em implementações com precisão finita por apresentarem bom comportamento numérico e uma organização estrutural adequada ao hardware. Sharma, Rawat e Agrawal apresentam a implementação em FPGA de um filtro *lattice wave digital* do tipo *notch*, no qual o fator de qualidade é ajustado ao longo do tempo para reduzir a duração do regime transitório. Os resultados mostram que essa abordagem contribui para um comportamento transitório mais controlado (70).

Por fim, apesar do maior custo em recursos, existem implementações que utilizam aritmética em ponto flutuante com o objetivo de ampliar a faixa dinâmica e reduzir a sensibilidade à quantização, aspectos especialmente relevantes em filtros recursivos. Bharade, Joshi e Manthalkar apresentam uma implementação em FPGA de filtros em estrutura *lattice* utilizando aritmética em ponto flutuante compatível com o padrão IEEE 754. O trabalho descreve a construção dos blocos de soma e multiplicação em ponto flutuante e discute o impacto dessa escolha em termos de precisão e custo de hardware, sendo frequentemente usado como referência para comparações entre ponto fixo e ponto flutuante. (71).

De forma geral, a literatura recente mostra que diferentes estratégias de implementação de filtros IIR em FPGA priorizam compromissos distintos entre eficiência de hardware, taxa de processamento, robustez numérica e faixa dinâmica. Estruturas convencionais em ponto fixo permanecem atrativas pela simplicidade e baixo custo, enquanto arquiteturas *lattice* se destacam pela maior robustez frente à quantização. Implementações em ponto flutuante, por sua vez, são adotadas quando a precisão numérica e a estabilidade são requisitos centrais, mesmo à custa de maior utilização de recursos. Nesse contexto, o presente trabalho propõe uma avaliação comparativa dessas três abordagens, IIR convencional quantizado, estrutura *lattice* e implementação em ponto flutuante, aplicadas à filtragem digital em FPGA no estágio de *shaper*, analisando os compromissos entre erro numérico, estabilidade e consumo de recursos lógicos.

4.2 IMPLEMENTAÇÃO DE UM FILTRO IIR QUANTIZADO

O primeiro método para implementação do circuito conformado a ser realizado é de um filtro IIR convencional com os coeficientes quantizados, o que será discutido na presente seção.

A implementação em FPGA exige a representação de grandezas reais em aritmética de precisão finita, já que os circuitos digitais operam exclusivamente com números inteiros. Porém, os coeficientes dos filtros e os sinais processamentos assumem, em geral, valores reais, o que torna necessário a necessidade de adotar estratégias de quantização, que consistem na aproximação desses valores contínuos por representações discretas com numero limitado de bits (61, 66).

Em implementações convencionais, e conforme foi feito por Paschoalin *et al.* (59) e adaptado para o presente trabalho, a quantização é realizada por meio de representações em ponto fixo, descritas pelo formato $QX.Y$, em que X representa o número de bits da parte inteira e Y da parte fracionária. Um coeficiente real c pode ser quantizado de acordo com

$$\hat{c} = round(c \cdot 2^{Y}) \cdot 2^{-Y} \tag{4.1}$$

Em que $\hat{c}$ representa o valor quantizado e $round(\cdot)$ denota a operação de arredondamento (65). Esse processo induz um erro de quantização que depende diretamente do número de bits utilizados e da distribuição dos coeficientes do filtro.

Para avaliar o erro de quantização no presente trabalho, o resultado da filtragem implementada com coeficientes quantizados é comparado com a resposta obtida a partir de uma filtragem realizada sem quantização. Para fins de padronização de nomenclatura, denota-se por $y[n]$ a saída da filtragem quantizada e por $y_{ideal}[n]$ a resposta de referência adotada para comparação.

O erro introduzido pelo processo de quantização é avaliado por meio de uma métrica energética, que considera a energia total da diferença entre a resposta quantizada e a resposta ideal do filtro, o erro amostral é dado por:

$$e[n] = y_{ideal}[n] - y[n] \tag{4.2}$$

O erro energético normalizado, amplamente utilizado na análise de quantização em filtros digitais, é então calculado como a razão entre a energia do erro e a energia da resposta ideal, fornecendo uma medida global do impacto da quantização ao longo de todo o sinal analisado (65, 66).

Logo, a energia do erro é dada por:

$$E_e = \sum_{n=0}^{N-1} e^2[n] \tag{4.3}$$

Enquanto a energia do sinal de referência é dada por:

$$E_y = \sum_{n=0}^{N-1} y_{ideal}^2[n] \tag{4.4}$$

Assim, o erro energético normalizado pode ser definido como (66):

$$e_{energ} = \frac{\sum_{n=0}^{N-1} e^2[n]}{\sum_{n=0}^{N-1} y_{ideal}^2[n]} = \frac{\sum_{n=0}^{N-1} (y_{ideal}[n] - y[n])^2}{\sum_{n=0}^{N-1} y_{ideal}^2[n]} \tag{4.5}$$

Tal equação será amplamente utilizada em todos os métodos analisados no presente trabalho.

4.2.1 Implementação em *Matlab*

O primeiro passo da análise consiste na implementação do filtro em *Matlab*, com o objetivo de avaliar a forma de onda de saída a partir da aplicação de uma resposta ao impulso na entrada, conforme ilustrado na Figura 30. O código apresentado a seguir descreve todo o processo de implementação, abrangendo desde a extração dos coeficientes do filtro até a realização da filtragem individual para cada filtro IIR implementado.

No código, os coeficientes dos filtros são carregados já no domínio-z, correspondendo a cada um dos sete filtros IIR a serem implementados. Cada conjunto de coeficientes é representado pela seguinte função de transferência:

$$H_i(z) = \frac{Zn_i(z)}{Zd_i(z)}, \tag{4.6}$$

em que

$$1 \leq i \leq 7. \tag{4.7}$$

A saída de cada filtro individual é descrita pela Equação 4.14. Em virtude das propriedades de linearidade dos filtros IIR, a resposta global do sistema pode ser interpretada como a soma das saídas dos filtros individuais, conforme discutido em (66).

Dessa forma, o código gera duas saídas distintas: o resultado da filtragem considerando a quantização dos coeficientes e o resultado obtido sem a aplicação desse efeito. Em ambos os casos, foi aplicado um ganho de entrada $G_x = 2^{32}$, conforme adotado e detalhado por Paschoalin *et al.* (59).

A implementação do filtro é realizada por meio de uma função denominada `iir_generico`, responsável pela execução de um filtro IIR genérico, ou seja, capaz de

operar com filtros de qualquer ordem. Essa função é disponibilizada no anexo do presente trabalho. De forma resumida, a função recebe como entradas os coeficientes Z_n e Z_d, juntamente com os fatores de quantização, e retorna tanto a saída do filtro quanto os coeficientes quantizados correspondentes.

```
1     N = 50000;
2
3     y       = zeros(N,length(Zn));
4     y_quant = zeros(N,length(Zn));
5
6     Gx = 2^32;
7     Gy = 2^10;
8
9     for i=1:length(Zn)
10        [y_quant(:,i), Zn_q, Zd_q] = iir_generico(Zn{i}, Zd{i}, N, Gx, Gy);
11        [y_temp, B] = impz (Zn{i}, Zd{i}, N);
12
13        Zn_quant{i} = Zn_q;
14        Zd_quant{i} = Zd_q;
15
16        y(:,i) = y_temp;
17    end
18
19    h       = sum(y, 2);
20    h_quant = sum(y_quant, 2);
```

As matrizes y e y_{quant} armazenam, amostra a amostra, as saídas dos filtros aplicados, razão pela qual possuem dimensão $N \times p$. Nesse contexto, N representa o tamanho do vetor de entrada, assumido, neste trabalho, como um valor arbitrário de 50 000 amostras, e $p = 7$ corresponde ao número de filtros implementados.

A partir dessas matrizes, são então calculados dois vetores finais, h e h_{quant}, que representam, respectivamente, as saídas globais dos filtros sem e com a aplicação da quantização dos coeficientes. A Figura 36 apresenta o *plot* desses dois vetores de forma sobreposta, evidenciando a eficácia do método implementado tanto na ausência quanto na presença da quantização.

Observa-se que o comportamento obtido é consistente com a resposta ao impulso da função de transferência do *shaper*, previamente apresentada nas Figuras 24 e 30, o que valida a coerência da implementação proposta.

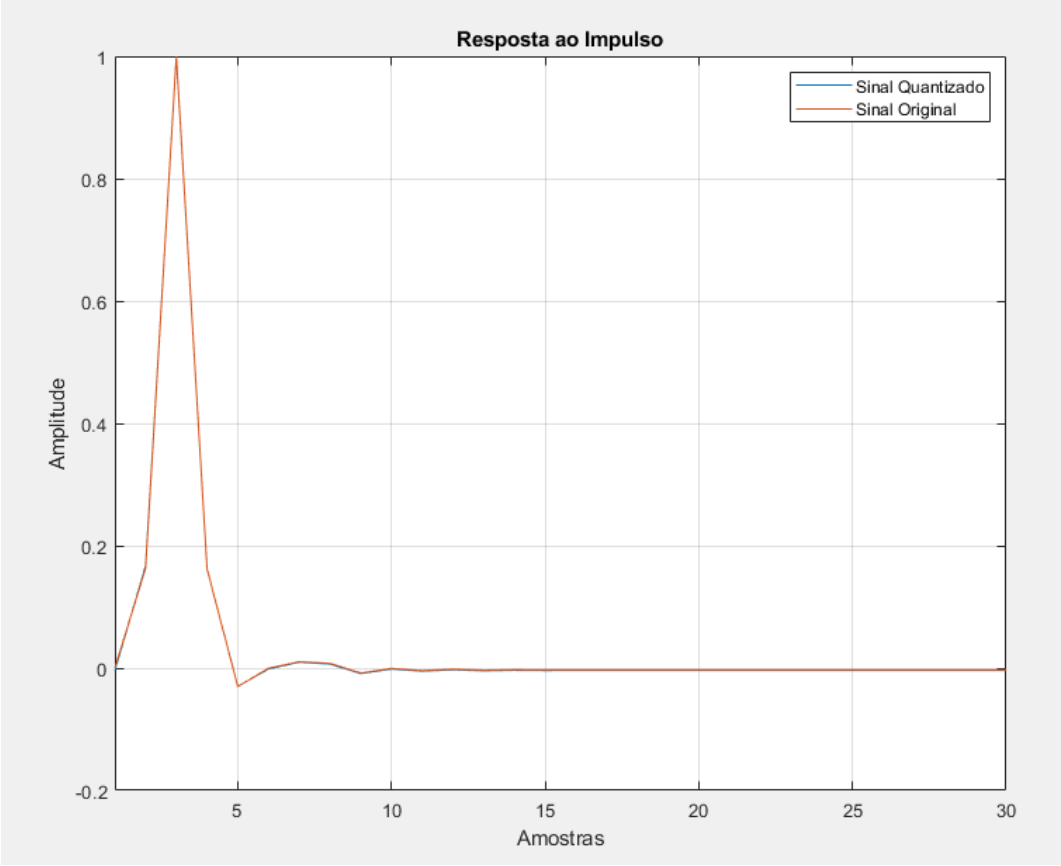

Figura 36 - Resposta ao Impulso do Filtro IIR

4.2.2 Implementação em FPGA

A implementação de um filtro digital IIR em FPGA requer a adaptação do modelo matemático do filtro às restrições impostas pelo hardware. No ambiente *Quartus*, essa implementação é realizada a partir da descrição da equação de diferenças do filtro em uma linguagem de descrição de hardware, na qual as operações aritméticas são explicitamente mapeadas em blocos digitais, como multiplicadores, somadores e registradores de atraso.

Como os FPGAs operam nativamente com aritmética discreta, os coeficientes e sinais do filtro devem ser previamente quantizados, permitindo sua representação em ponto fixo ou ponto flutuante. Dessa forma, o filtro IIR quantizado é implementado por meio de uma estrutura digital que reproduz a realimentação característica desse tipo de filtro, assegurando compatibilidade com os recursos disponíveis no dispositivo e com os requisitos de processamento em tempo real.

O primeiro passo é a criação de blocos que implementam o filtro IIR, tanto de primeira quanto de segunda ordem, que é a ordem máxima dos filtros que são utilizados, conforme a Equação 3.12. Para isso, dois módulos foram implementados em FPGA: `iir_ordem1` e `iir_ordem2`.

O código abaixo mostra a implementação do filtro de primeira ordem, o módulo `iir_ordem1`.

```
1     module iir_ordem1
2
3     #(
4             parameter BITS_IN = 33,
5             parameter G_ENTRADA = 2**32,
6             parameter G_SAIDA_LOG = 10,
7             parameter signed b0 =   785,
8             parameter signed a1 =  -1366
9     )
10
11    (
12            input  clock,
13            input  signed [BITS_IN-1:0] in,
14            output signed [BITS_IN+14:0] out
15            );
16
17            reg signed  [BITS_IN+14:0] ry = 0;
18            wire signed [BITS_IN+14:0] yz;
19            wire signed [BITS_IN+G_SAIDA_LOG+14:0] yp;
20
21            assign yz =   b0*in;
22            assign yp = - a1*ry;
23            assign out =   yz + (yp >>> G_SAIDA_LOG);
24
25            always @(posedge clock)
26            begin
27                    ry <= out;
28            end
29
30
31    endmodule
```

Esse trecho descreve, em *Verilog*, um filtro IIR de 1ª ordem implementado em hardware síncrono. A lógica segue a ideia clássica de um IIR com realimentação: a saída atual é calculada a partir da entrada atual e de uma fração (quantizada) da saída anterior armazenada em um registrador.

O módulo recebe uma amostra de entrada `in` a cada ciclo de *clock* e produz a saída `out`. O registrador `ry` armazena a saída anterior do filtro, de modo que a implementação reproduz uma equação diferenças do tipo:

$$y[n] = b_0 x[n] - a_1 y[n-1] \tag{4.8}$$

Como a implementação é quantizada, o termo de realimentação é escalonado por um fator definido por `G_SAIDA_LOG` (equivalente a uma divisão por $2^{G\_SAIDA\_LOG}$), o que resulta na forma prática em:

$$y[n] = b_0 x[n] + \frac{-a_1 y[n-1]}{2^{G\_SAIDA\_LOG}} \tag{4.9}$$

Em que $x[n]$ é `in`, $y[n]$ é `out`, e $y[n-1]$ é o valor armazenado em `ry`.

Com relação aos parâmetros e quantização:

- `BITS_IN`: define a largura do sinal de entrada `in` (33 bits, por padrão).
- `b_0` e `a_1`: os coeficientes do filtro já quantizados (inteiros com sinal).
- `G_SAIDA_LOG`: controla o ganho/escala do termo de realimentação. Na prática, o código usa um *shift* aritmético à direita para implementar uma divisão por potência de dois, o que é eficiente em FPGA.

Nas linhas 21 e 22, as operações do filtro são implementadas para o termo de avanço y_z e de realimentação y_p, por fim, na linha 23, a saída é atribuída à `out`, já com o escalonamento da parte quantizada.

A saída `out` tem largura maior (`BITS_IN+14:0`) para acomodar o crescimento de bits causado pelas multiplicações (`b0*in` e `a1*ry`) e reduzir risco de *overflow*. Esse aumento de largura é uma técnica comum em implementações quantizadas para preservar precisão e evitar saturação/estouro durante os cálculos.

A Figura 37 apresenta o circuito sintetizado no *Quartus* a partir do código descrito anteriormente. Nela, podem ser identificadas as entradas `in` e `clock`, o registrador de atraso implementado (`ry`), a saída `out`, bem como as principais operações aritméticas empregadas na implementação do filtro.

O outro código implementado em FPGA foi o IIR de segunda ordem, a lógica é parecida com a implementação de um bloco IIR de primeira ordem, mas dessa vez com mais registradores de atraso na entrada e na saída. Tal bloco corresponde, em essência, a uma equação de diferenças do formato:

$$y[n] = b_0 x[n] + b_1 x[n-1] - a_1 y[n-1] - a_2 y[n-2] \tag{4.10}$$

```
1    module iir_ordem2
2
```

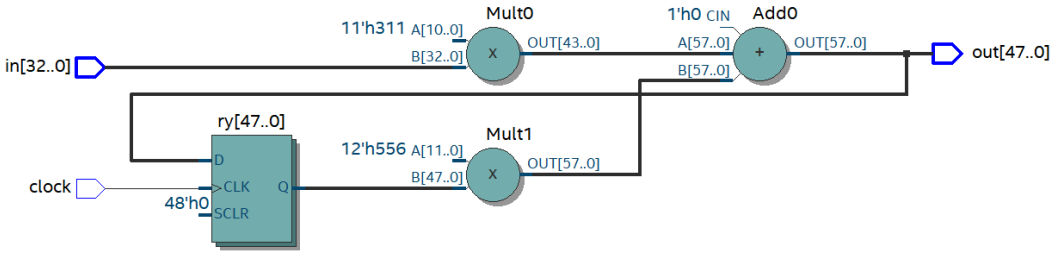

Figura 37 - Circuito digital sintetizado no *Quartus* para o filtro de primeira ordem

```
3       #(
4               parameter BITS_IN = 33,
5               parameter G_SAIDA_LOG = 10,
6               parameter signed b0 =  -788,
7               parameter signed b1 =  -399,
8               parameter signed a1 =  -618,
9               parameter signed a2 =   1362
10      )
11
12      (
13              input  clock,
14              input  signed [BITS_IN-1:0] in,
15              output signed [BITS_IN+14:0] out
16              );
17
18              reg signed  [BITS_IN:0] rx1 = 0;
19              reg signed  [BITS_IN+14:0] ry1 = 0, ry2 = 0;
20              wire signed [BITS_IN+14:0] yz;
21              wire signed [BITS_IN+G_SAIDA_LOG+14:0] yp;
22
23              assign yz =   b0*in + b1*rx1;
24              assign yp = - a1*ry1 - a2*ry2;
25              assign out =   yz + (yp >>> G_SAIDA_LOG);
26
27              always @(posedge clock)
28              begin
29                  rx1 <= in;
30                  ry2 <= ry1;
31              ry1 <= out;
```

```
32              end
33
34      endmodule
```

Os novos registradores implementados referem-se aos novos atrasos necessário para o cálculo do saída. São eles o `rx1`, que armazena o valor de $x[n-1]$, o `ry1` e o `ry2`, que armazenam os valores de $y[n-1]$ e $y[n-2]$, respectivamente.

O termo de avanço y_z passa a incorporar duas contribuições: a amostra de entrada atual e a amostra de entrada atrasada de uma unidade de tempo, conforme definido na linha 23. De forma análoga, o termo de realimentação é constituído pela soma ponderada de duas saídas anteriores do filtro, como apresentado na linha 24, caracterizando o comportamento de um filtro IIR de segunda ordem. A atribuição do valor de saída mantém a mesma estratégia adotada no filtro de primeira ordem, utilizando um fator de quantização que é implementado por meio de um deslocamento aritmético à direita, garantindo compatibilidade com a aritmética inteira do FPGA.

A lógica sequencial descrita pela estrutura `always` implementa um registrador de deslocamento de dois estágios, responsável por armazenar as amostras anteriores da entrada e da saída. A cada borda de subida do sinal de *clock*, os valores são atualizados e propagados para os respectivos registradores, assegurando que as informações necessárias para o cálculo das iterações seguintes estejam disponíveis de forma síncrona.

Vale ressaltar que os valores dos parâmetros presentes nos código não necessariamente refletem os valores verdadeiros que podem ser utilizados na implementação.

A Figura 38 mostra o circuito digital sintetizado no *Quartus*, com os três registradores de deslocamento, as entradas e a saída no circuito, semelhante ao filtro de primeira ordem.

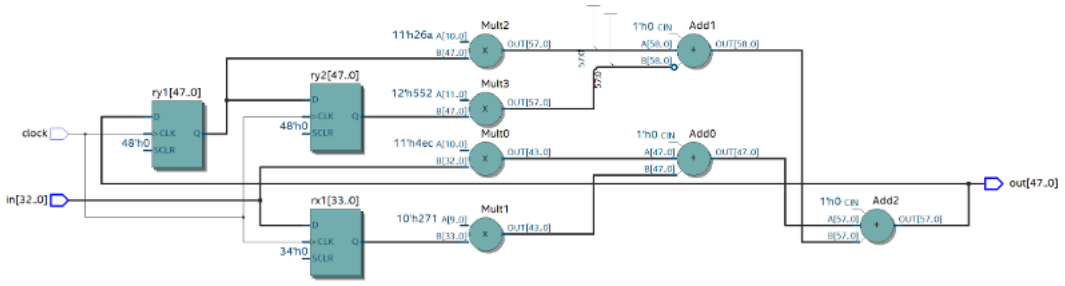

Figura 38 - Circuito digital sintetizado no *Quartus* para o filtro de segunda ordem

Por fim, a implementação da filtragem completa em FPGA baseia-se principalmente na instância dos módulos mostrados acima, em um arquivo principal do projeto, aplicando

corretamente os pesos utilizados. A Figura 39 mostra o circuito final sintetizado no *Quartus*, com cada bloco verde representando um filtro separado que foi implementado, totalizando 5. O código implementado está disponível no apêndice do trabalho.

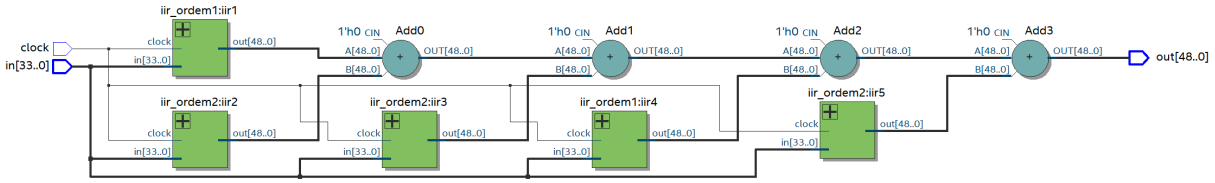

Figura 39 - Circuito digital sintetizado no *Quartus* para o circuito completo quantizado

Uma simulação no ambiente *Quartus* foi desenvolvida com o objetivo de avaliar a resposta ao impulso do filtro implementado. A simulação foi realizada com o auxílio do software *Questa*. A resposta ao impulso obtida é apresentada na Figura 40.

Nela é possível observar duas formas de onda: a primeira correspondente ao impulso aplicado na entrada, com ganho $G_x = 2^{10}$, e a segunda referente à saída do filtro, implementado de acordo com o circuito apresentado na Figura 39. Observa-se o comportamento da forma de onda semelhante ao que foi teorizado.

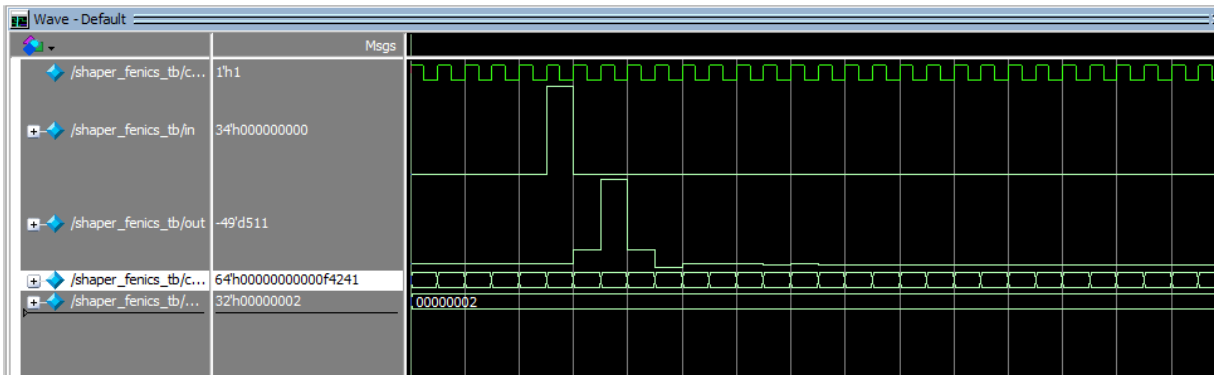

Figura 40 - Simulação da resposta ao impulso do filtro IIR em FPGA

4.3 IMPLEMENTAÇÃO DE UM FILTRO IIR QUANTIZADO NA ESTRUTURA *LATTICE*

A estrutura *lattice* (ou *lattice filter*) é uma realização alternativa para filtros digitais que substitui a implementação direta em termos dos coeficientes polinomiais a_i e b_i por uma decomposição em estágios recursivos e uniformes, parametrizados por coeficientes de

reflexão. Nessa abordagem, o filtro é construído como uma cascata de células idênticas, em que cada estágio combina sinais internos associados a erros de predição *forward* e *backward* (61, 66).

A implementação de um filtro IIR em uma estrutura *lattice* possui duas formas principais, de acordo com Mitra (66). A realização de uma função de transferência (*all-pole*) e o método de Gray-Markel. Para o presente trabalho foi escolhida a segunda implementação, devido à complexidade da função de transferência adotada.

Considere a função de transferência de ordem M,

$$H(z) \;=\; \frac{P_M(z)}{D_M(z)}, \qquad P_M(z) = \sum_{i=0}^{M} p_i z^{-i}, \qquad D_M(z) = 1 + \sum_{i=1}^{M} d_i z^{-i}. \tag{4.11}$$

Na realização IIR em estrutura *lattice* pelo método de Gray–Markel (*tapped cascaded lattice*), o filtro é implementado como uma cascata de seções *lattice* (parametrizadas pelos coeficientes de reflexão $\{k_m\}$) que gera sinais internos, e a saída é formada por uma combinação linear desses sinais. A equação final de saída é dada por

$$y[n] \;=\; \alpha_1\, y_1[n] \;+\; \sum_{m=1}^{M} \alpha_{m+1}\, s_{M-m+1}[n], \tag{4.12}$$

em que $y_1[n]$ e $s_m[n]$ são variáveis internas da própria estrutura *lattice* e $\{\alpha_i\}$ são coeficientes de ponderação escolhidos de forma a tornar a realização equivalente a $H(z) = P_M(z)/D_M(z)$ (66).

A estrutura *lattice* é adotada como alternativa para a implementação da função de transferência do *shaper* por apresentar uma realização mais robusta em aritmética de precisão finita. Em comparação com formas diretas, a arquitetura em estágios tende a reduzir a sensibilidade a quantização de coeficientes e aos efeitos de arredondamento acumulados na realimentação, contribuindo para preservar a estabilidade e o comportamento dinâmico do filtro após a discretização em ponto fixo.

Sua natureza modular e regular também facilita a implementação em FPGA, permitindo o controle sistemático de largura de palavra ao longo dos estágios e uma relação mais favorável entre erro numérico e recursos de hardware empregados (66).

A Figura 41 mostra a estrutura de filtro *lattice* no formato discutido, nela é possível observar que o filtro pode ser interpretado como vários estágios idênticos em cascata, o que facilita sua implementação em softwares como *Matlab* e *Quartus*.

A estrutura *lattice* pode ser vista como uma cascata de M estágios, em que cada estágio m atualiza recursivamente os sinais internos *forward* e *backward*. Em uma forma compacta, as equações de diferença por estágio podem ser escritas como

$$f_m[n] = f_{m-1}[n] - k_m\, g_{m-1}[n-1], \tag{4.13}$$

$$g_m[n] = g_{m-1}[n-1] + k_m\, f_m[n], \tag{4.14}$$

$$y_m[n] = \alpha \cdot g_m[n] \tag{4.15}$$

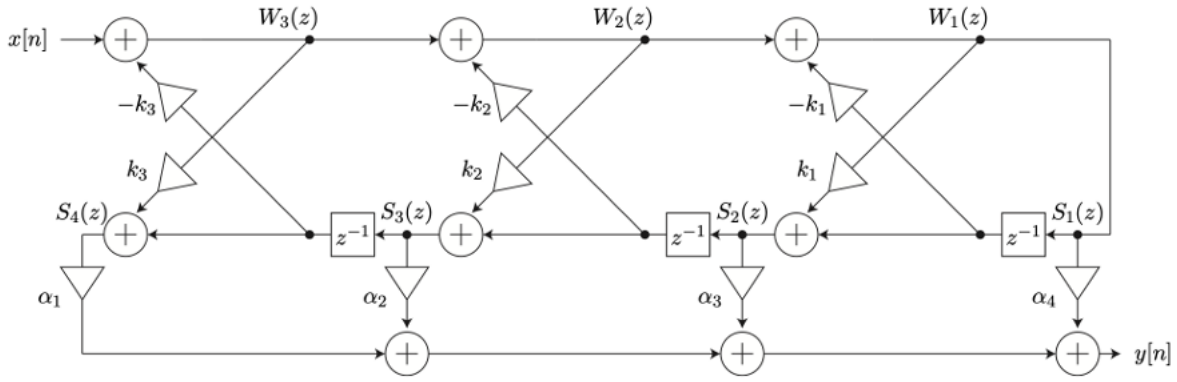

Figura 41 - Estrutra de um filtro *lattice* para filtro IIR de acordo com o método de Gray-Markel (76)

para $m = 1, 2, \ldots, M$, com inicialização usual $f_0[n] = x[n]$ e $g_0[n] = x[n]$, em que k_m é o coeficiente de reflexão do m-ésimo estágio (66).

4.3.1 Implementação em *Matlab*

A implementação de um filtro em estrutura *lattice* no *Matlab* pode ser realizada diretamente a partir do diagrama de blocos apresentado anteriormente, em que os estágios são conectados em cascata: cada etapa utiliza os sinais internos produzidos pela etapa anterior, e a concatenação de M estágios resulta no filtro completo.

No *Matlab*, a função `tf2latc` permite converter os coeficientes da função de transferência no domínio z para a parametrização em *lattice*, retornando os coeficientes de reflexão k_m e os coeficientes de ponderação α_m associados à realização do tipo Gray–Markel, conforme ilustrado na Figura 40 para cada estágio do filtro.

De maneira análoga ao caso do IIR convencional, os estágios podem ser organizados em seções implementadas separadamente e, ao final, suas contribuições podem ser combinadas, explorando a linearidade do sistema, desde que a decomposição adotada preserve a equivalência global com a função de transferência original.

A função `lattice_filter` foi desenvolvida pelo autor e é apresentada a seguir. Ela recebe como parâmetros o sinal de entrada $x[n]$, os coeficientes da estrutura *lattice* k e v, e o ganho de quantização dos coeficientes G. Por convenção de nomenclatura adotada nesta implementação, os coeficientes α_m do método de Gray–Markel são representados por v_m, notação também encontrada em parte da literatura. De forma resumida, a função executa a filtragem em estrutura *lattice* conforme o modelo de Gray–Markel, sendo aplicável a filtros de ordem arbitrária.

```
1   function y = lattice_filter(x, k, v, G)
2
3       k = floor(k*G);
4       v = floor(v*G);
5
6       N = length(x);
7       M = length(k);
8
9
10      f = zeros(N, M+1);
11      g = zeros(N, M+1);
12      rg = zeros(M, 1);
13      y = zeros(N,1);
14
15      for n = 1:N
16
17          f(n, 1) = x(n)*G;
18
19          for m = 1:M
20              f(n, m+1) = f(n,m) - floor(k(M-m+1) * rg(m)/G);
21
22              g(n,m) = floor(f(n, m+1) * k(M-m+1)/G) + rg(m);
23
24              y(n) = y(n) + floor(v(M-m+2) * g(n, m)/G);
25          end
26
27          g(n, M+1) = f(n, M+1);
28
29          y(n) = y(n) + floor(v(1) * g(n, M+1)/G);
30
31          for m = 1:M
32              rg(m) = g(n, m+1);
33          end
34      end
35  end
36
```

Para utilizar a função, foi desenvolvido o código a seguir, também em *Matlab*. Inicialmente, os coeficientes do filtro no formato convencional são convertidos para a parametrização em *lattice* por meio da função `tf2latc`, conforme indicado na linha 11. Em seguida, a filtragem é executada tanto pela estrutura *lattice* quanto pela implementação IIR convencional, com o objetivo de permitir uma comparação direta entre as duas realizações

sob as mesmas condições de quantização. Ao final, são obtidos os vetores h_{latt} e h_{iir}, que representam, respectivamente, a resposta resultante do método *lattice* quantizado (composta pela soma das contribuições dos filtros implementados separadamente) e a resposta do método IIR convencional.

```
    Gx = 2^32;
    Gy = 2^10;

    k = Zn;
    v = Zd;

    y_latt = zeros(N, length(Zn));
    y_iir  = zeros(N, length(Zn));

    for i = 1:length(Zn)
        [k{i}, v{i}] = tf2latc(Zn{i}, Zd{i});

        y_latt(:,i) = lattice_filter(x, k{i}, v{i}, Gy);

        y_iir(:,i) = filter(Zn{i}, Zd{i}, x);
    end

    h_latt = sum(y_latt, 2);
    h_iir  = sum(y_iir, 2);
```

Como resultado, observa-se na Figura 42 a resposta ao impulso do filtro implementado em estrutura *lattice* em comparação com a resposta ao impulso obtida pela realização IIR convencional. A forte sobreposição entre os sinais indica que a implementação em *lattice* reproduz adequadamente o comportamento esperado do filtro sob as mesmas condições de quantização. A análise quantitativa dos erros e das diferenças residuais entre as duas realizações é apresentada no capítulo seguinte.

4.3.2 Implementação em FPGA

A implementação do filtro *lattice* em FPGA segue a mesma abordagem adotada para o filtro IIR convencional, sendo baseada diretamente nas equações de diferenças correspondentes aos filtros de primeira e segunda ordens. De forma análoga, os coeficientes são previamente quantizados antes de sua aplicação na arquitetura digital.

Como etapa inicial, foi desenvolvido um bloco digital destinado à implementação de um filtro IIR *lattice* de primeira ordem. Esse bloco, denominado `lattice_1ordem`, é apresentado a seguir por meio do código correspondente em *Verilog*.

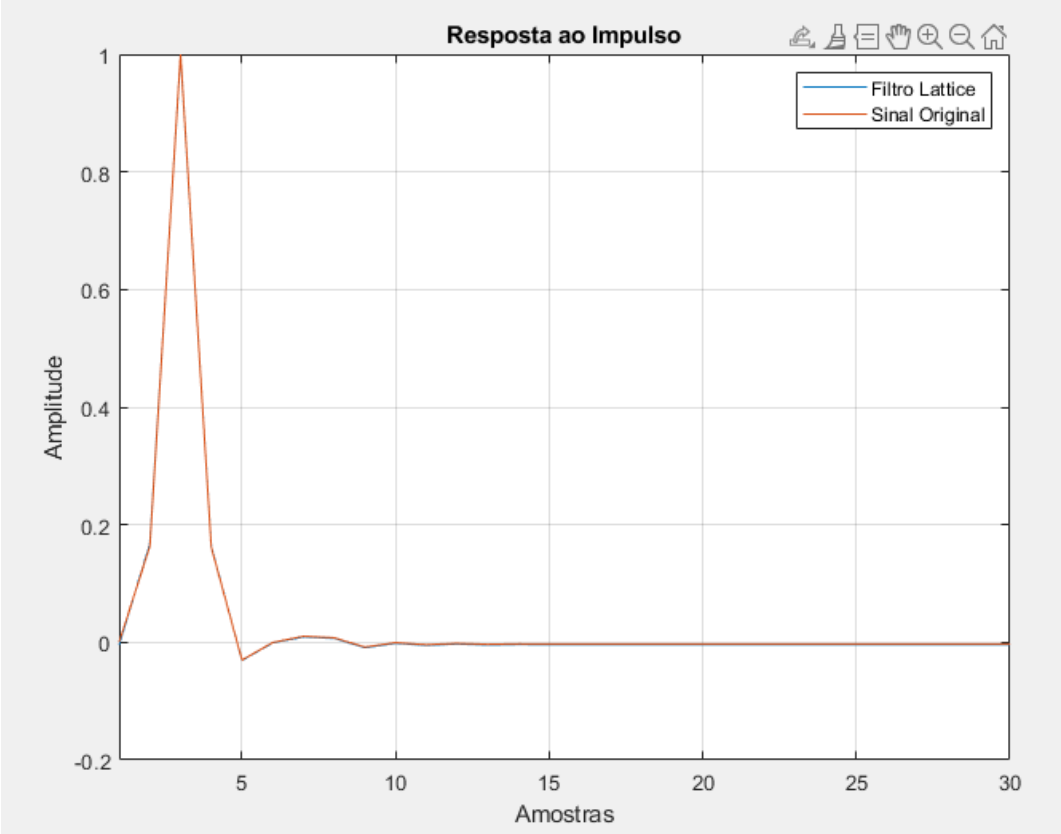

Figura 42 - Resposta ao impulso com a estrutura *lattice* em comparação com o IIR convencional

A Figura 43, desenvolvida pelo autor, mostra o diagrama de um filtro *lattice* de primeira ordem.

A partir dela, é possível extrair as seguintes equações diferenças, que serão implementadas em FPGA.

$$f[n] = x[n] - k \cdot f[n-1] \tag{4.16}$$

$$g[n] = k \cdot f[n] + f[n-1] \tag{4.17}$$

$$y[n] = v_2 \cdot g[n] + v_1 \cdot f[n] \tag{4.18}$$

No código desenvolvido, os parâmetros utilizados são:

- `NBITS_IN`: determina o número de bits do sinal de entrada `in`, sendo adotado `NBITS_IN=33` por padrão. Como o filtro envolve multiplicações por coeficientes e somas internas, a largura de palavra dos sinais intermediários e da saída é aumentada para reduzir o risco de saturação e acomodar o crescimento dinâmico, motivo pelo qual a saída `out` é declarada com `NBITS_IN` + 15 bits.

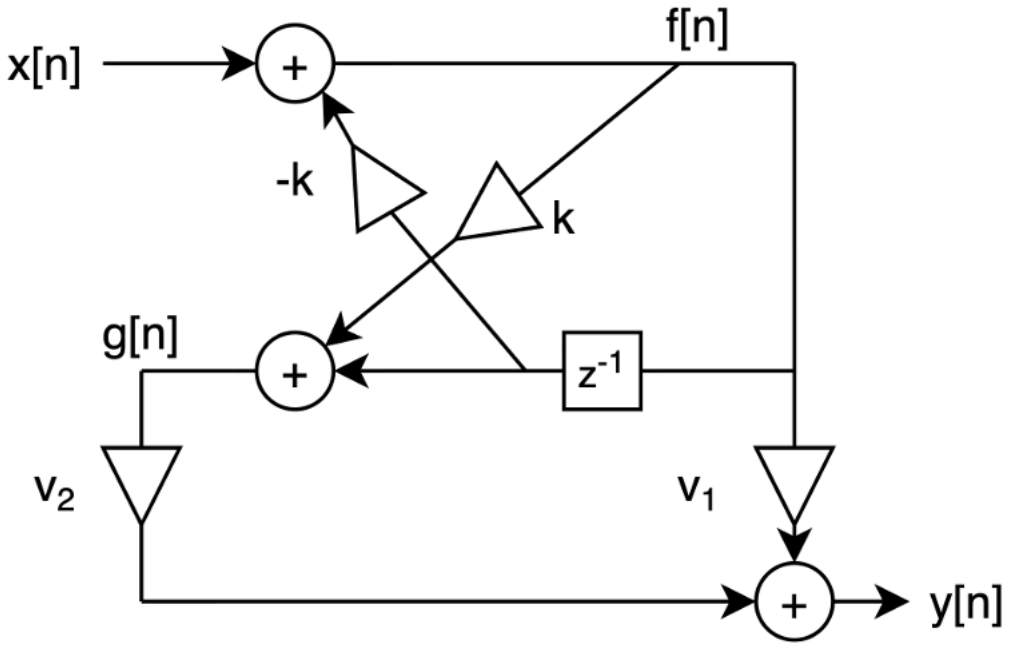

Figura 43 - Diagrama de blocos de um filtro *lattice* de primeira ordem

- `FB`: número de bits fracionários associado à quantização dos coeficientes, isto é, o ganho $G = 2^{\texttt{FB}}$. Neste trabalho, adota-se $\texttt{FB} = 10$, correspondendo a $G_y = 2^{10}$.
- k, v_1 e v_2: coeficientes do filtro *lattice* de primeira ordem, representados em ponto fixo com `FB` bits fracionários.

A partir disso, o próximo passo é a implementação das Equações (4.33), (4.34) e (4.35), que descrevem o fluxo do sinal. O código segue uma estratégia padrão de ponto fixo: toda multiplicação por um coeficiente quantizado é seguida pela reescala por $2^{\texttt{FB}}$, implementada pelo deslocamento aritmético à direita (`>>> FB`).

Assim, os termos `kf1` e `kf` correspondem a $k \cdot f[n-1]$ e $k \cdot f[n]$ reescalados, enquanto `v1g` e `v2f` implementam $v_1 \cdot g[n]$ e $v_2 \cdot f[n]$ no mesmo formato numérico. Em seguida, as atribuições contínuas realizam diretamente as equações do filtro, e o registrador `f1` armazena o estado $f[n]$ para ser utilizado como $f[n-1]$ no próximo ciclo de *clock*.

```
1     module lattice_1ordem
2     #(
3             parameter NBITS_IN = 33,
4             parameter integer FB = 10,
5             parameter signed k = 1,
6             parameter signed v1 = 1,
7             parameter signed v2 = 1
```

```
8
9       )
10      (
11              input clock,
12              input  signed [NBITS_IN - 1:0] in,
13              output signed [NBITS_IN + 14:0] out
14              );
15
16              reg signed [NBITS_IN:0] f1 = 0;
17
18              wire signed [NBITS_IN+14:0] f;
19              wire signed [NBITS_IN+14:0] g;
20
21              //reescala
22              wire signed [NBITS_IN+14:0] kf1  = (k * f1)  >>> FB;
23              wire signed [NBITS_IN+14:0] kf   = (k * f)   >>> FB;
24
25              wire signed [NBITS_IN+14:0] v1g  = (v1 * g)  >>> FB;
26              wire signed [NBITS_IN+14:0] v2f  = (v2 * f)  >>> FB;
27
28
29              assign f = in - kf1;
30              assign g = kf + f1;
31              assign out = v1g + v2f;
32
33              always @(posedge clock)
34              begin
35                      f1 <= f;
36              end
37
38
39      endmodule
```

A Figura 44 mostra o circuito digital resultante do código implementado.

A segunda etapa do projeto foi desenvolver o bloco digital para implementação de um filtro IIR de segunda ordem, denominado de `lattice_2ordem`, mostrado abaixo, também em *Verilog*.

A Figura 45, desenvolvida, pelo autor mostra o diagrama de um filtro *lattice* de segunda ordem. As equações diferenças relacionadas aos sinais destacados são:

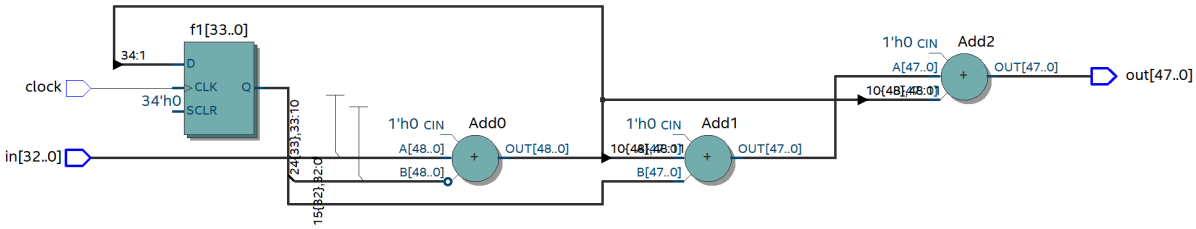

Figura 44 - Circuito digital implementado para um filtro IIR *lattice* de primeira ordem

$$f[n] = x[n] - k_2 \cdot i[n-1] \tag{4.19}$$
$$h[n] = f[n] - k_1 \cdot h[n-1] \tag{4.20}$$
$$i[n] = k_1 \cdot h[n] + h[n-1] \tag{4.21}$$
$$g[n] = k_2 \cdot f[n] + i[n-1] \tag{4.22}$$
$$y[n] = v_3 \cdot g[n] + v_2 \cdot i[n] + v_1 \cdot h[n] \tag{4.23}$$

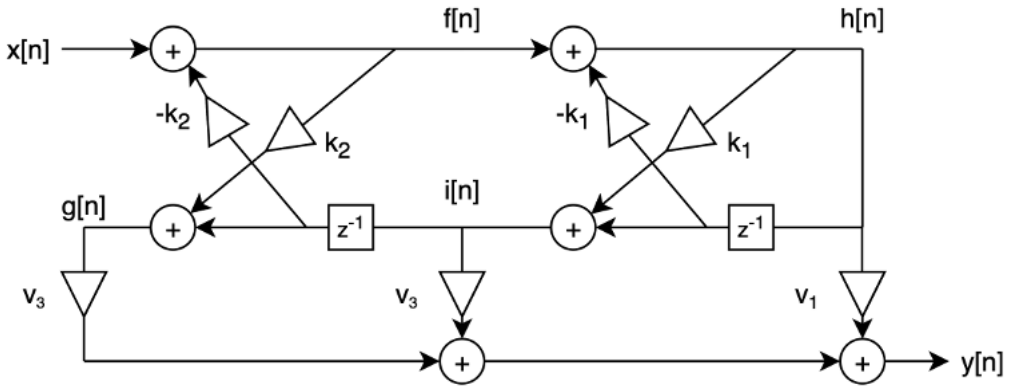

Figura 45 - Diagrama de blocos de um filtro *lattice* de segunda ordem

O código do filtro de segunda ordem mantém, em linhas gerais, os mesmos parâmetros e a mesma estratégia de implementação adotada para o filtro de primeira ordem, utilizando a mesma largura de palavra para os sinais de entrada e saída e o mesmo valor de deslocamento aritmético definido por `FB` para reescala dos produtos em ponto fixo.

A principal diferença está no aumento do número de estados e, consequentemente, na quantidade de operações. Neste caso, são utilizados dois registradores para armazenar as amostras anteriores necessárias à recursão, correspondentes aos termos $i[n-1]$ e $h[n-1]$.

Esses estados são implementados pelos registradores `i1` e `h1`, declarados nas linhas 18 e 19, garantindo que os valores atrasados estejam disponíveis para o cálculo da amostra corrente.

As multiplicações pelos coeficientes quantizados seguidas da reescala são realizadas de maneira análoga ao caso de primeira ordem, porém em maior número, conforme implementado entre as linhas 27 e 34. Em seguida, as atribuições das linhas 37 a 41 materializam as equações de diferença do filtro apresentadas anteriormente. Por fim, a atualização dos estados ocorre a cada borda de subida do *clock*, quando os valores atuais $i[n]$ e $h[n]$ são armazenados nos registradores `i1` e `h1`, passando a representar $i[n-1]$ e $h[n-1]$ no ciclo subsequente.

```
1     module lattice_2ordem
2     #(
3             parameter NBITS_IN = 33,
4             parameter integer FB = 10,
5             parameter signed k1 = 1,
6             parameter signed k2 = 1,
7             parameter signed v1 = 1,
8             parameter signed v2 = 1,
9             parameter signed v3 = 1
10
11    )
12    (
13            input clock,
14            input  signed [NBITS_IN - 1:0] in,
15            output signed [NBITS_IN + 14:0] out
16            );
17
18            reg signed [NBITS_IN:0] i1 = 0; // i[n-1]
19            reg signed [NBITS_IN:0] h1 = 0; // h[n-1]
20
21            wire signed [NBITS_IN+14:0] f; // f[n]
22            wire signed [NBITS_IN+14:0] h; // h[n]
23            wire signed [NBITS_IN+14:0] g; // g[n]
24            wire signed [NBITS_IN+14:0] i; // i[n]
25
26            //reescala
27            wire signed [NBITS_IN+14:0] k2i1  = (k2 * i1) >>> FB;
28            wire signed [NBITS_IN+14:0] k1h1  = (k1 * h1) >>> FB;
29            wire signed [NBITS_IN+14:0] k1h   = (k1 * h)  >>> FB;
30            wire signed [NBITS_IN+14:0] k2f   = (k2 * f)  >>> FB;
31
```

```
32              wire signed [NBITS_IN+14:0] v1g   = (v1 * g)  >>> FB;
33              wire signed [NBITS_IN+14:0] v2i   = (v2 * i)  >>> FB;
34              wire signed [NBITS_IN+14:0] v3h   = (v3 * h)  >>> FB;
35
36
37              assign f = in  - k2i1;
38              assign h = f   - k1h1;
39              assign g = k2f + i1;
40              assign i = k1h + h1;
41              assign out = v1g + v2i + v3h;
42
43
44              always @(posedge clock)
45              begin
46                      i1 <= i;
47                      h1 <= h;
48              end
49
50
51      endmodule
```

A Figura 46 mostra o circuito sintetizado no *Quartus* a partir desse código.

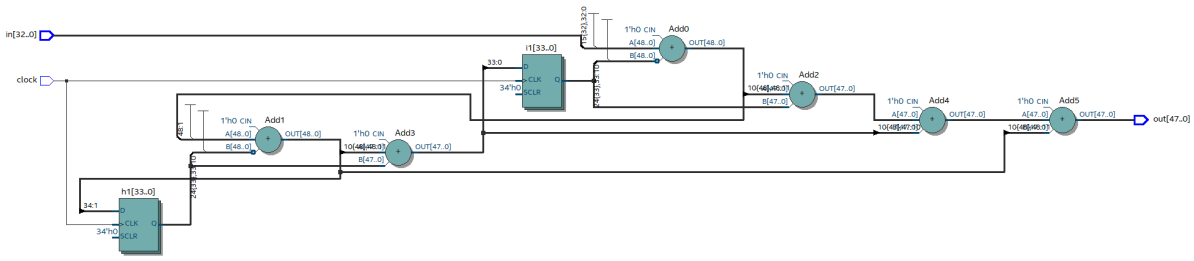

Figura 46 - Circuito digital implementado para um filtro IIR *lattice* de segunda ordem

Por fim, o código implementado para a filtragem baseia-se na instância dos módulos mostrados acima, em um arquivo principal, com os devidos pesos para cada filtro. A Figura 47 mostra o circuito sintetizado no *Quartus*, com todos os filtros implementados. O código implementado está disponível no anexo deste trabalho.

Como feito no caso anterior, uma simulação no *Quartus* foi desenvolvido para analisar a resposta ao impulso do filtro, com auxílio do *Questa*. A resposta ao impulso

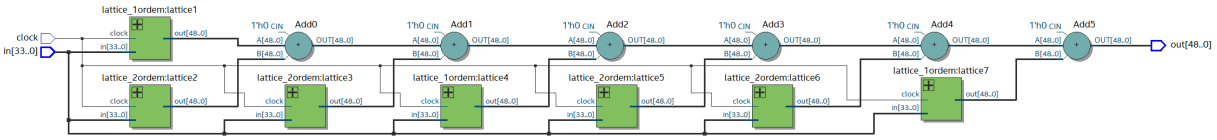

Figura 47 - Circuito digital sintetizado no *Quartus* para o circuito completo de um filtro IIR *lattice* quantizado

está mostrada na Figura 48. Nela é possível observar o formato da resposta ao impulso, semelhante ao que foi teorizado e mostrado na Figura 38.

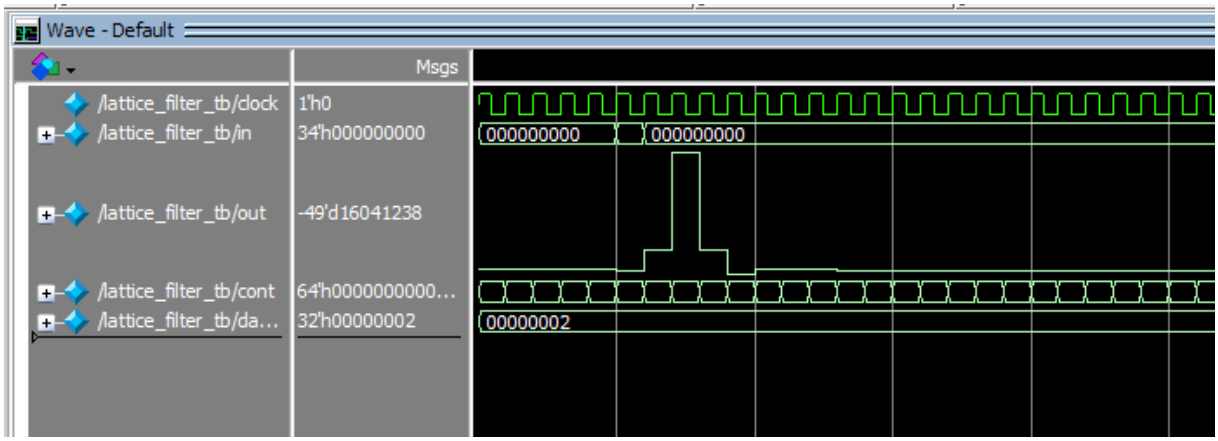

Figura 48 - Simulação da resposta ao impulso do filtro IIR *lattice* em FPGA

4.4 IMPLEMENTAÇÃO DE UM FILTRO IIR EM NOTAÇÃO DE PONTO FLUTUANTE EM FPGA

A implementação de um filtro IIR em notação de ponto flutuante pode, à primeira vista, parecer pouco natural em FPGA, já que esses dispositivos realizam operações aritméticas de forma mais direta em ponto fixo. Ainda assim, essa abordagem é viável ao representar os operandos no formato definido pelo padrão IEEE 754 e ao empregar blocos aritméticos capazes de executar soma e multiplicação nesse domínio (77).

A principal motivação é mitigar os efeitos de quantização associados às realizações convencionais em ponto fixo. Em filtros recursivos, erros de arredondamento e a quantização de coeficientes e estados internos se propagam e se acumulam ao longo das realimentações, podendo alterar a resposta do sistema e, em situações mais críticas, resultar em ciclos limite e até comprometer a estabilidade. A aritmética em ponto flutuante reduz essa sensibilidade

ao oferecer maior faixa dinâmica e menor dependência de reescalonamentos, aproximando o comportamento do filtro implementado daquele obtido em referência numérica (61).

Porém, é evidente que o aumento da complexidade dos circuitos implementados em FPGA implica maior consumo de recursos lógicos. Assim, o objetivo desta aplicação é avaliar o compromisso entre o erro obtido e a quantidade de lógica consumida na implementação. Esses resultados serão apresentados e discutidos no capítulo posterior. Nesta seção, serão descritos os circuitos desenvolvidos e empregados nas análises.

4.4.1 Representação de Números em Ponto Flutuante de Acordo Com o Padrão IEEE 754

A representação de números em ponto flutuante é amplamente utilizada em sistemas digitais quando se deseja preservar uma grande faixa dinâmica e reduzir erros de arredondamento em operações aritméticas. O padrão mais adotado para essa finalidade é o IEEE 754, que define formatos binários, regras de arredondamento e o comportamento para valores especiais, garantindo portabilidade e consistência entre diferentes arquiteturas de hardware e software (77).

De forma geral, um número em ponto flutuante no padrão IEEE 754 é codificado em três campos: sinal, expoente e fração. Para números normalizados, o valor representado pode ser escrito como:

$$x = (-1)^s \cdot (1.F) \cdot 2^{(E-\text{bias})}, \tag{4.24}$$

em que s é o bit de sinal, E é o expoente armazenado, F é a fração armazenada (parte fracionária do significando) e bias é o deslocamento aplicado ao expoente, permitindo a representação de expoentes positivos e negativos.

Os formatos mais utilizados definidos pelo IEEE 754 são o *single precision* e o *double precision*, que diferem principalmente na quantidade de bits destinada ao expoente e à fração, o que impacta diretamente a faixa dinâmica e a precisão numérica (77).

No formato *single precision* (32 bits), a palavra é composta por 1 bit de sinal, 8 bits de expoente e 23 bits de fração, resultando em aproximadamente 7 dígitos decimais de precisão (77). Já no *double precision* (64 bits), utiliza-se 1 bit de sinal, 11 bits de expoente e 52 bits de fração, alcançando aproximadamente 15 a 16 dígitos decimais de precisão (77).

Como exemplo, segue o número 10.5, representado em ponto flutuante:

$$10.5_{10} = 1010.1_2 \tag{4.25}$$

Colocando no formato $(1.F.2^e)$:

$$1010.1_2 = 1.0101_2 \cdot 2^3 \tag{4.26}$$

$$e = 3 \tag{4.27}$$

$$E = e + bias = 3 + 127 = 130 \tag{4.28}$$

Ou seja, até o momento:

$$F = 0101_2 \tag{4.29}$$

$$E = 130_{10} = 10000010_2 \tag{4.30}$$

Considerando a quantidade de bits de cada parte no formato *single precision*, e que $s = 0$, já que o valor é positivo, é possível concluir que:

$$s = 0_2 \tag{4.31}$$

$$E = 10000010_2 \tag{4.32}$$

$$F = 01010000000000000000000_2 \tag{4.33}$$

Com isso, o valor final no formato IEEE 754 é:

$$0\,10000010\,01010000000000000000000$$

4.4.2 Representação de Números em Ponto Flutuante no Formato Adaptado ao Padrão IEEE 754

O formato adotado neste trabalho é uma proposta alternativa inspirada no IEEE 754, desenvolvida por Santos V. A. M. (78), para representação binária em ponto flutuante. A palavra é organizada em quatro campos lógicos: sinal, expoente, bit implícito e parte fracionária da mantissa, o que simplifica a descrição dos blocos em hardware (78).

A codificação mantém a mantissa normalizada, com significando no intervalo $[1, 2)$, exigindo bit mais significativo igual a 1. Diferentemente do IEEE 754, o expoente é representado em complemento de dois, permitindo expoentes positivos e negativos sem o uso de *bias* (78).

Para precisão simples de 32 bits, o formato utiliza 1 bit de sinal s, 8 bits de expoente e e 23 bits de mantissa m. A mantissa armazena a parte fracionária do significando, enquanto o bit implícito representa a parte inteira igual a 1, decorrente da normalização (78).

A representação em ponto flutuante é dada por:

$$N_{float} = (-1)^s \times M \times 2^e \tag{4.34}$$

Diferentemente do IEEE 754, o formato adotado mantém explicitamente o bit mais significativo da mantissa na palavra representada. Além disso, o expoente é codificado em complemento de dois, permitindo representar diretamente valores positivos e negativos (78).

Como exemplo, a representação do número 10.5 no formato de ponto flutuante binário com 32 bits é dada por:

$$0\,11101101\,10101000000000000000000$$

Em que,

- $s = 0$
- $e = 11101101_2 = -19_{10}$
- $M = 10101000000000000000000_2 = 5.505.024_{10}$

Realizando o cálculo inverso, de acordo com a Equação 4.51:

$$N_{float} = (-1)^0 \times 5.505.024 \times 2^{-19} = 10.5 \tag{4.35}$$

O valor obtido do retorno nem sempre será exato, com isso é possível obter um erro de conversão que varia de acordo com o número de bits utilizados para a representação. Tal erro será discutido no capítulo posterior.

A Tabela 1 mostra o comparativo entre os valores representados no formato IEEE 754 e o formato adaptado.

	Representação binária
Padrão IEEE 754	0 10000010 01010000000000000000000
Formato Proposto	0 11101101 10101000000000000000000

Tabela 1 – Comparativo entre os formatos IEEE 754 e o proposto

A Tabela 2 mostra a mesma comparação, mas com separação entre sinal, expoente e mantissa.

	Bit do sinal	Expoente	Mantissa
Padrão IEEE 754	0	10000010	01010000000000000000000
Formato proposto	0	11101101	10101000000000000000000

Tabela 2 – Comparativo entre as representações: sinal, expoente e mantissa.

4.4.3 Circuitos Implementados em FPGA

A implementação em FPGA envolve o desenvolvimento de circuitos digitais para cada etapa, a saber: a conversão de *float* para *int*, o circuito de normalização e os operadores de soma e multiplicação. Por convenção de nomenclatura, o formato proposto pode ser referido como *int*, uma vez que a codificação resulta em uma palavra inteira.

O primeiro circuito desenvolvido é o de normalização, mostrado na Figura 49, cuja função é deslocar a mantissa até que ela assuma a forma normalizada, com o bit mais significativo igual a 1. Em paralelo, o expoente é ajustado para compensar esse deslocamento, preservando o valor numérico representado, conforme circuito mostrado na Figura 50 (74).

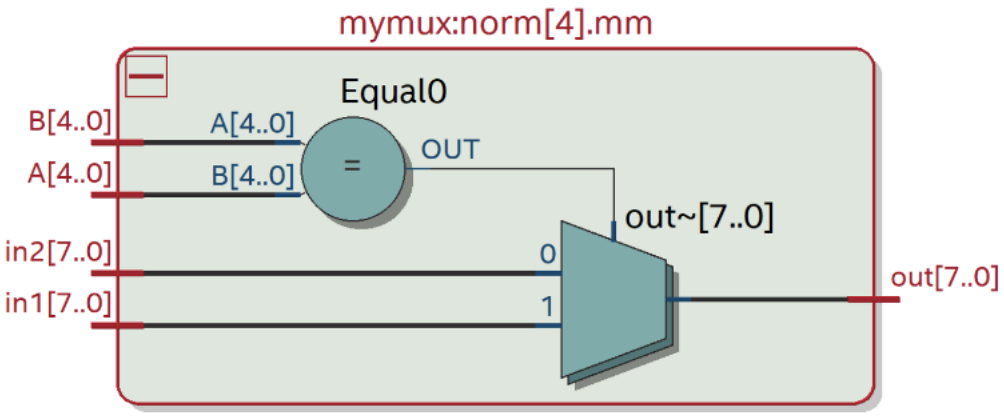

Figura 49 - Multiplexador para normalização da mantissa (74)

O próximo circuito desenvolvido é o que converte o número de ponto flutuante para o formato proposto, mostrado na Figura 51. Ele recebe o sinal com 23 bits, separa o bit de sinal do valor de entrada, obtém o módulo do número para formar a mantissa inicial e define o expoente como zero. Em seguida, envia esses campos ao bloco de normalização, que ajusta a mantissa por deslocamento e corrige o expoente de forma consistente, produzindo na saída a palavra já no formato ponto flutuante (74).

O próximo circuito, mostrado na Figura 52, implementa a soma entre dois números

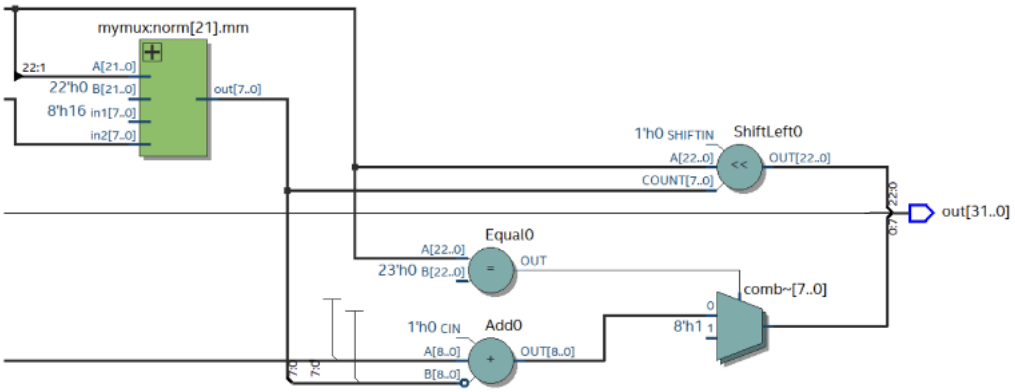

Figura 50 - Circuito para deslocamento de bits do expoente (74).

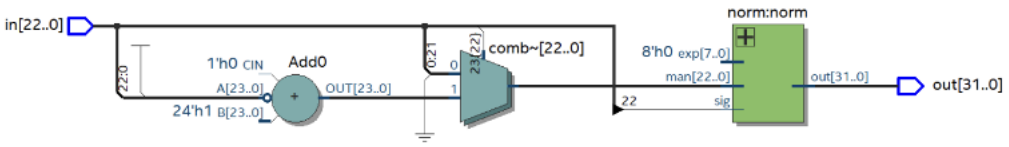

Figura 51 - Circuito de conversão para o formato proposto (74).

no formato de ponto flutuante proposto. O circuito primeiro desempacota as palavras de entrada em sinal, expoente e mantissa. Em seguida, realiza a desnormalização, isto é, iguala os expoentes deslocando a mantissa do operando com menor expoente, para que os dois valores fiquem na mesma escala. Com os expoentes alinhados, as mantissas são somadas já considerando os sinais. Por fim, o resultado passa novamente por um bloco de normalização, que ajusta a mantissa para a forma normalizada e corrige o expoente, gerando a palavra final no mesmo formato de entrada (74, 78).

O último circuito a ser desenvolvido foi o de multiplicação, mostrado na Figura 53. Nele, as duas palavras de entrada são desempacotadas em sinal, expoente e mantissa. O sinal do resultado é obtido pela combinação dos sinais de entrada, enquanto o expoente resultante é calculado pela soma dos expoentes. Em paralelo, as mantissas são multiplicadas, gerando um produto com maior largura, do qual são selecionados os bits mais significativos para manter o tamanho definido do campo de mantissa. Por fim, o resultado é enviado ao bloco de normalização, que ajusta a mantissa e corrige o expoente, garantindo que a saída permaneça no mesmo formato adotado no restante da arquitetura (74, 78).

O circuito emprega a equação descrita abaixo:

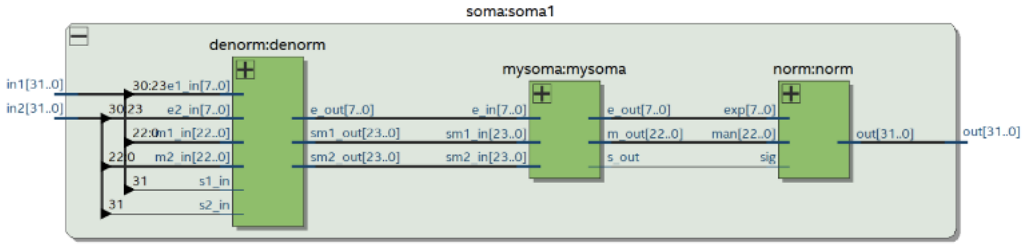

Figura 52 - Circuito de soma entre dois números no formato proposto (74).

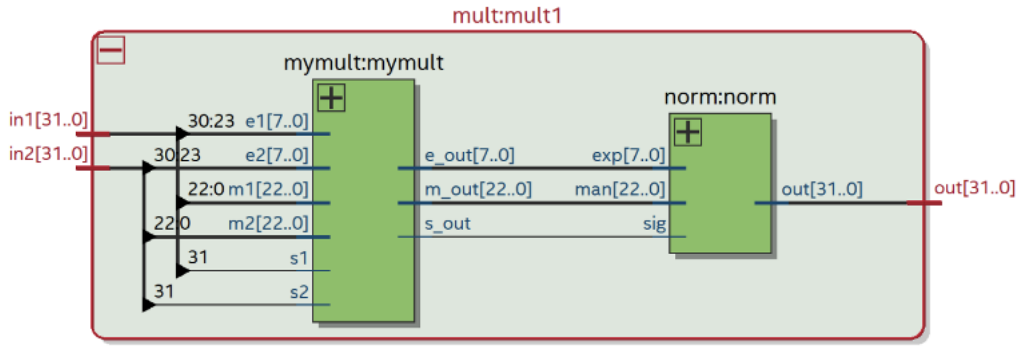

Figura 53 - Circuito de multiplicação entre dois números no formato proposto (74).

$$N = n_1 \times n_2 = (M_1 M_2) \cdot 2^{(e_1+e_2)} \tag{4.36}$$

Tais circuitos já foram previamente desenvolvidos pra um trabalho anterior (74), por isso detalhes de código não serão abordados neste trabalho, apenas a sua utilização para a aplicação do filtro IIR.

4.4.4 Filtro IIR Implementado em Notação de Ponto Flutuante

Com todos os circuitos auxiliares previamente desenvolvidos, o próximo passo é implementar o filtro IIR do *shaper* considerando esses formatos numéricos. Para isso, os coeficientes da função de transferência do *shaper* (Equação 3.12) também devem ser convertidos para o formato de ponto flutuante adotado. Essa conversão é realizada com o auxílio de um código em *Python*, disponível no anexo deste trabalho.

A implementação do filtro IIR em ponto flutuante segue a mesma lógica das abordagens anteriores, sendo necessário, inicialmente, desenvolver módulos para a filtragem de primeira e segunda ordens. Entretanto, como explicado anteriormente, as operações

devem agora ser realizadas por meio dos blocos apresentados na seção anterior, específicos para aritmética em ponto flutuante do formato proposto.

O código abaixo mostra a implementação de um filtro IIR de primeira ordem nesse formato. Ele recebe os coeficientes de 32 bits já em ponto flutuante e realiza os cálculos padrões de um filtro IIR convencional de primeira ordem (Equação 4.14, para M=1).

Primeiramente, o sinal de entrada, até então em aritmética de ponto fixo, deve ser convertido para o formato de ponto flutuante proposto, por meio do módulo `int2float`, implementado na linha 22. As multiplicações do sinal pelos coeficientes são realizadas nas linhas 27 e 28, com o auxílio do módulo `mult`. A soma final é feita com o módulo `soma`, conforme mostrado na linha 30. Por fim, o valor de saída é armazenado como amostra anterior a cada borda de *clock*. A Figura 54 mostra o circuito sintetizado em FPGA.

```
1   module iir1
2   #(
3           parameter MAN = 23,
4           parameter EXP = 8,
5
6           parameter [MAN + EXP:0] b0 = 32'hF0E11AF5,
7           parameter [MAN + EXP:0] a1 = 32'hF4FFB7AC,
8           parameter [MAN + EXP:0] a1_neg = 32'h74FFB7AC
9
10  )
11  (
12          input clk,
13          input signed [MAN - 1:0] x,
14          output [MAN+EXP:0] y_float
15  );
16
17  wire [MAN+EXP:0] x_float;
18  wire [MAN+EXP:0] y_gained;
19
20  reg [MAN+EXP:0] ry_float = 0;
21
22  int2float i2f(x, x_float);
23
24  wire [MAN+EXP:0] yz;
25  wire [MAN+EXP:0] yp;
26
27  mult mult1 (b0, x_float, yz);
28  mult mult2 (a1_neg, ry_float, yp);
29
```

```
30      soma soma1 (yz, yp, y_float);
31
32      always @(posedge clk) ry_float <= y_float;
33
34      endmodule
```

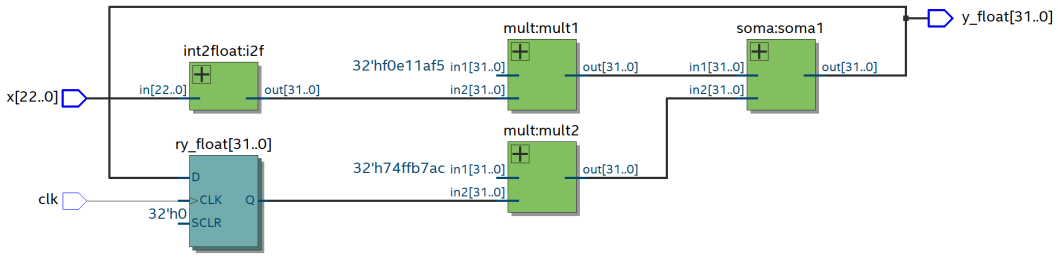

Figura 54 - Circuito do filtro IIR de primeira ordem com aritmética de ponto flutuante sintetizado em FPGA

A implementação do filtro de segunda ordem segue a mesma lógica, porém envolve um conjunto maior de operações devido ao aumento da complexidade do circuito. O código a seguir apresenta a implementação, e a Figura 55 mostra o circuito sintetizado em FPGA.

```
1       module iir2
2       #(
3               parameter MAN = 23,
4               parameter EXP = 8,
5
6               parameter [MAN + EXP:0] b0 = 32'h74DD394F,
7               parameter [MAN + EXP:0] b1 = 32'h746F15D2,
8               parameter [MAN + EXP:0] a1 = 32'h754320A4,
9               parameter [MAN + EXP:0] a2  = 32'h7449E230,
10              parameter [MAN + EXP:0] a1_neg = 32'hF54320A4,
11              parameter [MAN + EXP:0] a2_neg = 32'hF449E230
12
13      )
14      (
15              input clk,
16              input signed [MAN - 1:0] x,
17              output [MAN+EXP:0] y_float
18      );
19
20      wire [MAN+EXP:0] x_float;
```

```
21      wire [MAN+EXP:0] y_gained;
22
23      reg [MAN+EXP:0] ry1_float = 0;
24      reg [MAN+EXP:0] ry2_float = 0;
25      reg [MAN+EXP:0] rx_float = 0;
26
27      wire [MAN+EXP:0] yz1;
28      wire [MAN+EXP:0] yz2;
29      wire [MAN+EXP:0] yp1;
30      wire [MAN+EXP:0] yp2;
31
32      wire [MAN + EXP:0] s1;
33      wire [MAN + EXP:0] s2;
34
35      int2float i2f(x, x_float);
36
37      mult m1(b0, x_float,   yz1);
38      mult m2(b1,  rx_float,  yz2);
39      mult m3(a1_neg, ry1_float, yp1);
40      mult m4(a2_neg, ry2_float, yp2);
41
42      soma soma1(yz1, yz2, s1);
43      soma soma2(yp1, yp2, s2);
44      soma soma3(s1, s2, y_float);
45
46      always @(posedge clk) rx_float  <= x_float;
47      always @(posedge clk) ry1_float <= y_float;
48      always @(posedge clk) ry2_float <= ry1_float;
49
50
51      endmodule
52
```

Por fim, o circuito sintetizado da implementação da filtragem completa é mostrado na Figura 56. Ele integra todos os filtros desenvolvidos nas seções anteriores. Como a saída do filtro encontra-se no formato de ponto flutuante, não é viável visualizar diretamente a forma de onda no *Quartus*, como foi feito nos métodos anteriores. Nesse caso, a validação deve ser realizada pela exportação dos resultados para um software como o *Matlab*, permitindo a comparação com os valores de referência. Esse procedimento e a discussão dos resultados serão apresentados no capítulo posterior.

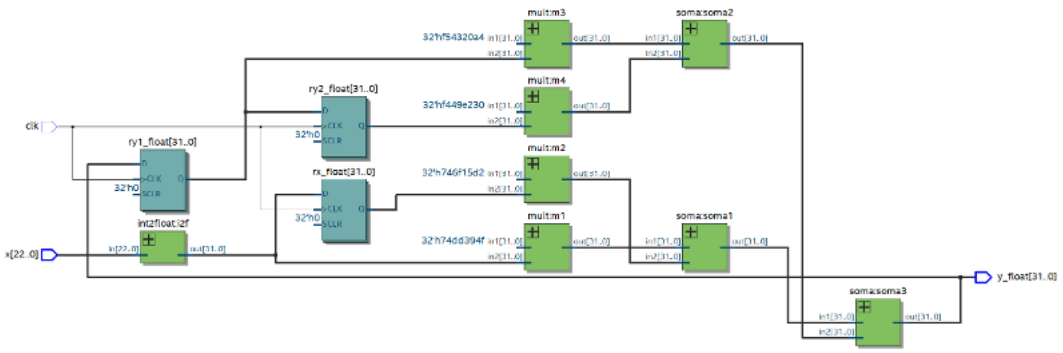

Figura 55 - Circuito do filtro IIR de segunda ordem com aritmética de ponto flutuante sintetizado em FPGA

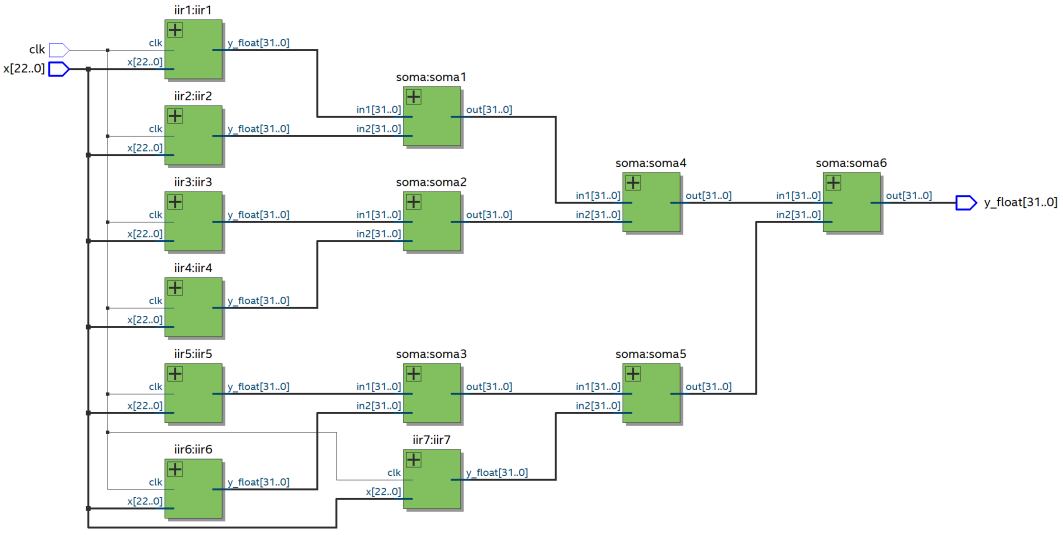

Figura 56 - Circuito do filtro IIR de segunda ordem com aritmética de ponto flutuante sintetizado em FPGA

5 RESULTADOS

O presente capítulo apresenta os resultados das implementações, em FPGA, dos três filtros IIR discutidos no capítulo anterior: o filtro IIR convencional quantizado, o filtro IIR quantizado na estrutura *lattice* e o filtro IIR em notação de ponto flutuante.

A análise é organizada em dois eixos principais:

- O erro associado a cada implementação, decorrente da quantização ou da representação em ponto flutuante. Esse erro é quantificado por meio do erro energético, definido na Equação 4.22.
- O esforço computacional requerido para a implementação em FPGA, avaliado a partir do consumo de recursos lógicos e de blocos DSP em cada caso.

Esses dois pontos devem ser analisados em conjunto. Embora o objetivo seja minimizar o erro, o esforço computacional também deve ser considerado, pois, em arquiteturas mais complexas, o consumo de recursos pode se tornar um gargalo para a implementação em FPGA.

5.1 FILTRO IIR QUANTIZADO

A primeira análise refere-se ao filtro IIR quantizado, apresentado na Seção 4.2. Nesse caso, o erro associado decorre da quantização dos coeficientes. A Figura 57 apresenta o erro ponto a ponto entre a filtragem ideal e a filtragem IIR quantizada, obtido pela aplicação da Equação 4.19 aos dois sinais da Figura 35.

A partir do gráfico, observa-se que o erro permanece, majoritariamente, na ordem de 10^{-3}. O erro energético obtido a partir desses valores é:

$$e_{IIR} = 3.5443 \times 10^{-4} \tag{5.1}$$

Esse resultado indica que, em termos de energia, a discrepância acumulada entre a saída ideal e a saída quantizada é pequena quando comparada à energia do sinal de referência, caracterizando uma boa aproximação do filtro implementado. Em outras palavras, a quantização dos coeficientes introduz uma perturbação de baixa magnitude, que tende a impactar apenas de forma sutil a forma de onda reconstruída, sem alterar significativamente o comportamento global da resposta do filtro.

No que diz respeito ao esforço computacional, a Figura 58 apresenta o sumário de compilação no *Quartus* para o circuito do filtro IIR quantizado mostrado na Figura 38.

Ressalta-se que a estimativa de recursos é obtida a partir da compilação para um dispositivo-alvo definido no próprio *Quartus* (5CGXFC7C7F23C8 da família Cyclone

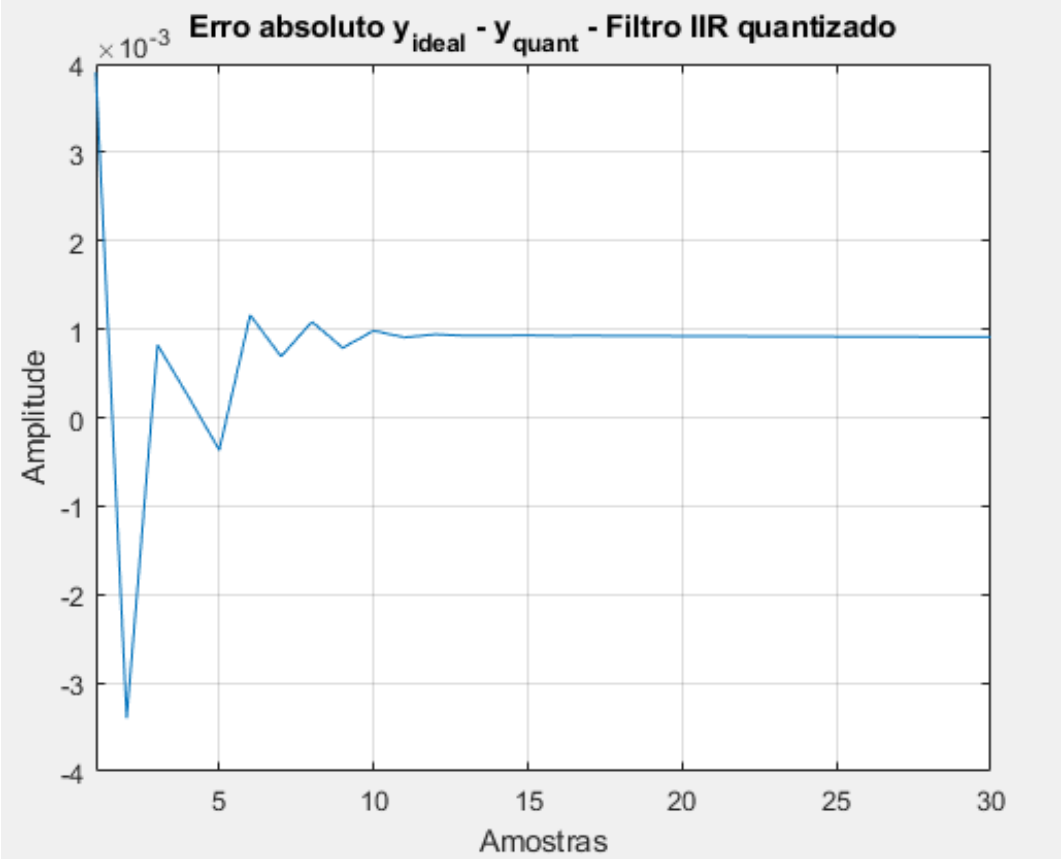

Figura 57 - Erro absoluto entre a resposta ao impulso ideal e a quantizada para um filtro IIR

V), sem a necessidade de uma placa física instalada. Assim, para fins de comparação, o procedimento é válido, uma vez que todas as implementações são compiladas para o mesmo dispositivo, sob as mesmas configurações.

Assim, na Figura em questão, destacam-se:

- A lógica utilizada: 427 de 56.480 (<1%).
- O total de registradores: 49.
- O total de blocos DSPs: 20 de 156 (13 %).

Em linhas gerais, os registradores armazenam os estados internos e atrasos do filtro e garantem a sincronização dos sinais a cada ciclo de *clock*. Os blocos DSP são unidades dedicadas a multiplicações e operações do tipo MAC, usadas para implementar com maior eficiência as contas do filtro. Quando essas operações não são mapeadas em DSP, elas podem ser realizadas na lógica programável do FPGA, principalmente em *ALMs*, que combinam LUTs e recursos aritméticos para implementar a lógica e parte das operações (79).

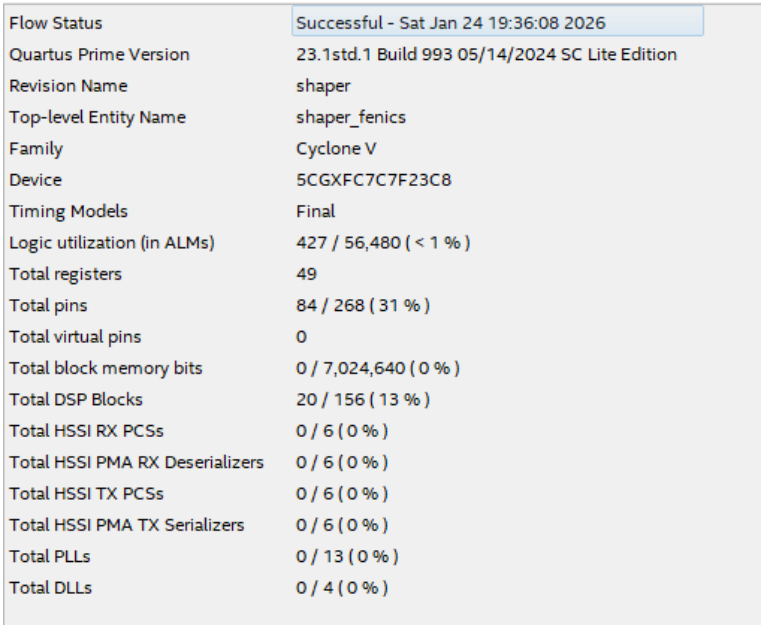

Figura 58 - Sumário de simulação para o filtro IIR quantizado no *Quartus*

A Tabela 3 mostra o resumo dos valores obtidas para o filtro IIR quantizado, com o erro energético e quantidade de registradores e DSPs utilizados na implementação em FPGA.

Filtro IIR quantizado	
Erro Energético	3.54×10^{-4}
Registradores	49
DSPs	20 (13%)
Lógica (ALMs)	427 (<1%)

Tabela 3 – Resumo do desempenho do filtro IIR quantizado

5.2 FILTRO IIR QUANTIZADO NA ESTRUTURA *LATTICE*

A análise ocorre de maneira semelhante para o filtro IIR com estrutura *lattice*. A Figura 59 mostra o erro ponto a ponto entre o filtro ideal e o filtro IIR *lattice*. É possível observar que o erro também permanece na casa de 10^{-3} e com um comportamento muito semelhante ao do IIR convencional.

O erro energético calculado para esse caso foi de:

$$e_{lattice} = 3.43 \times 10^{-4} \tag{5.2}$$

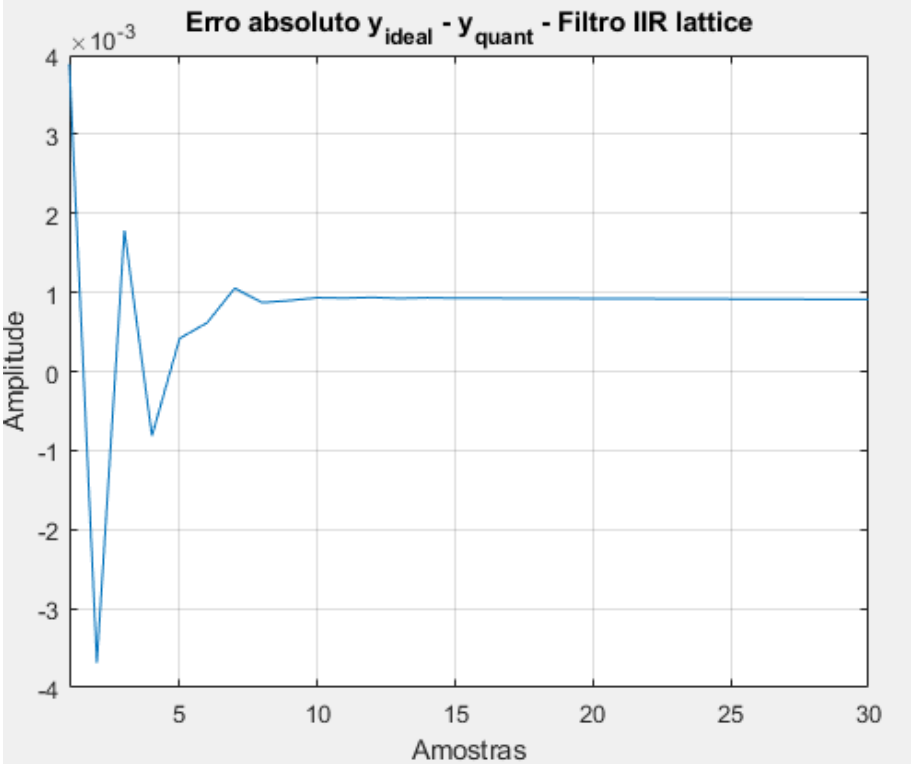

Figura 59 - Erro absoluto entre a resposta ao impulso ideal e a quantizada para o filtro IIR na estrutura *lattice*

O sumário de simulação é mostrado na Figura 60, indicando:

- A lógica utilizada: 589 de 56.480 (1%).
- O total de registradores: 140.
- O total de blocos DSPs: 28 de 156 (18%).

A Tabela 4 resume os valores obtidos para o filtro IIR com estrutura *lattice*.

Filtro IIR *lattice*	
Erro Energético	3.45×10^{-4}
Registradores	140
DSPs	28 (18%)
Lógica (ALMs)	589 (1%)

Tabela 4 – Resumo do desempenho do filtro IIR *lattice*.

Flow Status	Successful - Sat Jan 24 19:41:43 2026
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	shaper
Top-level Entity Name	lattice_filter
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	589 / 56,480 (1 %)
Total registers	140
Total pins	84 / 268 (31 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	28 / 156 (18 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Figura 60 - Sumário de simulação para o filtro IIR *lattice* no *Quartus*

5.3 FILTRO IIR EM PONTO FLUTUANTE EM FPGA

Para a análise do filtro IIR na estrutura *lattice*, inicialmente é necessário extrair os valores gerados na simulação do *Quartus* e compará-los com auxílio de um código desenvolvido em *Python*. Conforme realizado em um trabalho anterior (74), os dados obtidos ainda se encontram no formato de ponto flutuante proposto e, para permitir a comparação, devem ser convertidos para o formato *float* convencional. Essa conversão é realizada pelo mesmo código em *Python*.

A Figura 61, adaptada de (74), apresenta o diagrama de blocos do código desenvolvido em *Python* para validação da implementação do filtro em notação de ponto flutuante em FPGA.

No código, os dados provenientes da simulação são importados e convertidos para o formato *float*. Em seguida, calcula-se a resposta ao impulso ideal a partir dos coeficientes originais, sem quantização, de modo a obter um sinal de referência para comparação. A Figura 62 apresenta ambos os sinais no mesmo formato, permitindo a comparação direta.

Por fim, a Figura 63 apresenta o erro absoluto ponto a ponto obtido para esse caso. Observa-se que o erro se mantém, majoritariamente, na ordem de 10^{-6}, sendo inferior ao verificado nos dois métodos anteriores.

O erro energético, também calculado pelo código em *Python*, é dado por:

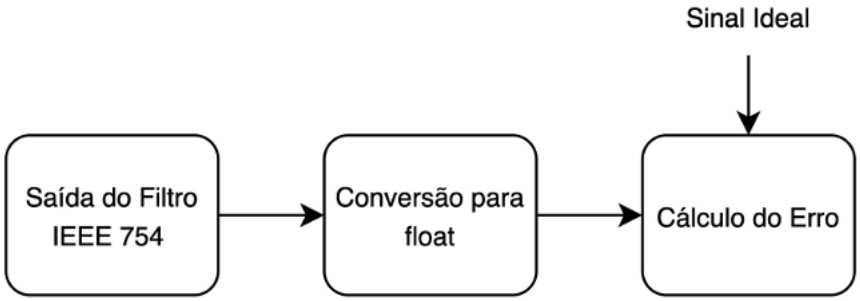

Figura 61 - Diagrama de blocos do código desenvolvido para validação do método, adaptado de (74).

$$e_{fp} = 7.28 \times 10^{-10} \tag{5.3}$$

Esse valor é significativamente menor do que os demais, o que era esperado, uma vez que, nesta implementação, não há quantização dos coeficientes como nos métodos em ponto fixo. O erro residual observado está associado principalmente a limitações numéricas da própria representação em ponto flutuante e às operações de arredondamento inerentes ao hardware.

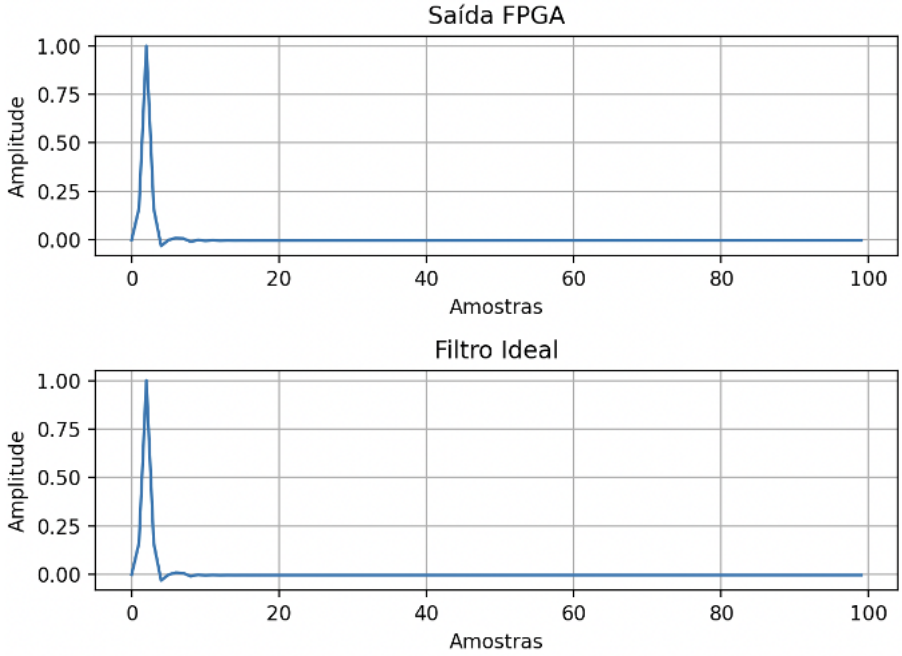

Figura 62 - Comparação da saída do filtro em FPGA com a ideal para a implementação em ponto flutuante.

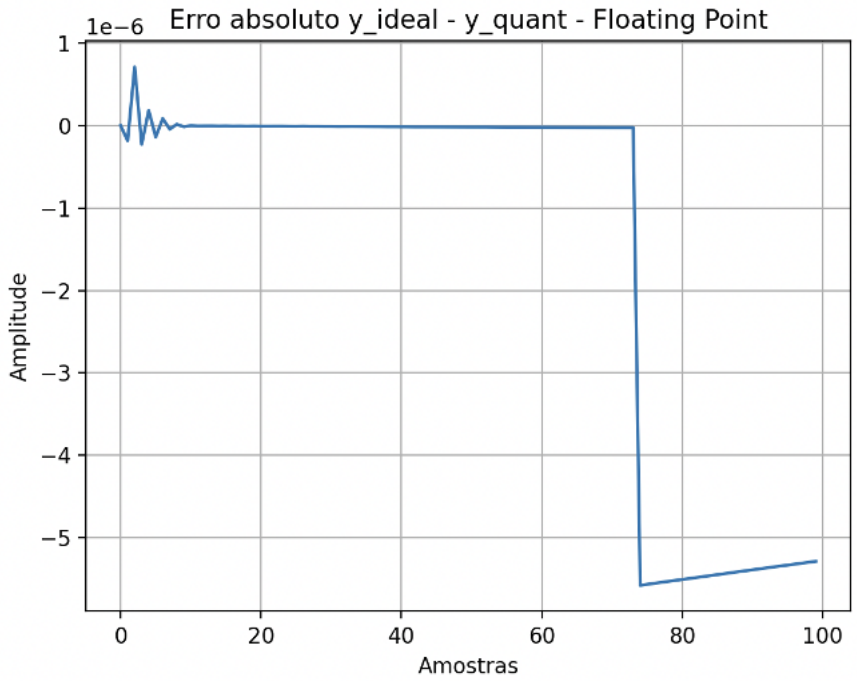

Figura 63 - Erro absoluto entre a resposta ao impulso ideal e a gerada em FPGA para o filtro IIR com notação de ponto flutuante

Em contrapartida, a Figura 64 mostra o sumário de simulação do *Quartus*, indicando:

- A lógica utilizada: 7.867 de 56.480 (14%).
- O total de registradores: 112.
- O total de blocos DSPs: 22 de 156 (14%).

A Tabela 5 mostra o resumo dos valores para este caso.

Filtro IIR em ponto flutuante	
Erro Energético	7.28×10^{-10}
Registradores	112
DSPs	22 (14%)
Lógica (ALMs)	7.867 (14%)

Tabela 5 – Resumo do desempenho do filtro IIR em ponto flutuante.

Embora este método apresente o menor erro energético entre as implementações em FPGA, o consumo elevado de lógica pode se tornar um gargalo para sua aplicação

Flow Status	Successful - Fri Jan 23 23:11:08 2026
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	teste
Top-level Entity Name	iir
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	7,867 / 56,480 (14 %)
Total registers	112
Total pins	56 / 268 (21 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	22 / 156 (14 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Figura 64 - Sumário de simulação para o filtro IIR em notação de ponto flutuante no *Quartus*

prática. Esse resultado é esperado, pois a aritmética em ponto flutuante exige circuitos auxiliares para operações como normalização, alinhamento de expoentes, arredondamento e tratamento de casos especiais, aumentando significativamente a quantidade de recursos quando comparada às realizações em ponto fixo, nas quais as operações são implementadas de forma mais direta no *Quartus*.

É importante destacar que o erro energético está diretamente relacionado ao número de bits utilizado na representação em ponto flutuante. Trata-se de um método flexível, no qual os valores podem ser representados com diferentes larguras de palavra, inferiores a 32 bits, conforme a configuração adotada (78).

Em um trabalho anterior (74), diferentes configurações de bits foram avaliadas com o objetivo de estimar o erro energético e mantê-lo abaixo de 1%. A Figura 65 apresenta a relação entre o erro obtido e a quantidade de bits utilizada em cada caso.

Observa-se que, a partir de 14 bits, o erro já permanece abaixo de 1%, reduzindo-se progressivamente à medida que a precisão aumenta. Para 32 bits, por exemplo, obtém-se o erro apresentado na Equação 5.3.

5.4 COMPARAÇÃO ENTRE OS MÉTODOS

A Tabela 6 mostra o resumo dos três métodos empregados, focando no erro energético obtido e no esforço computacional realizado.

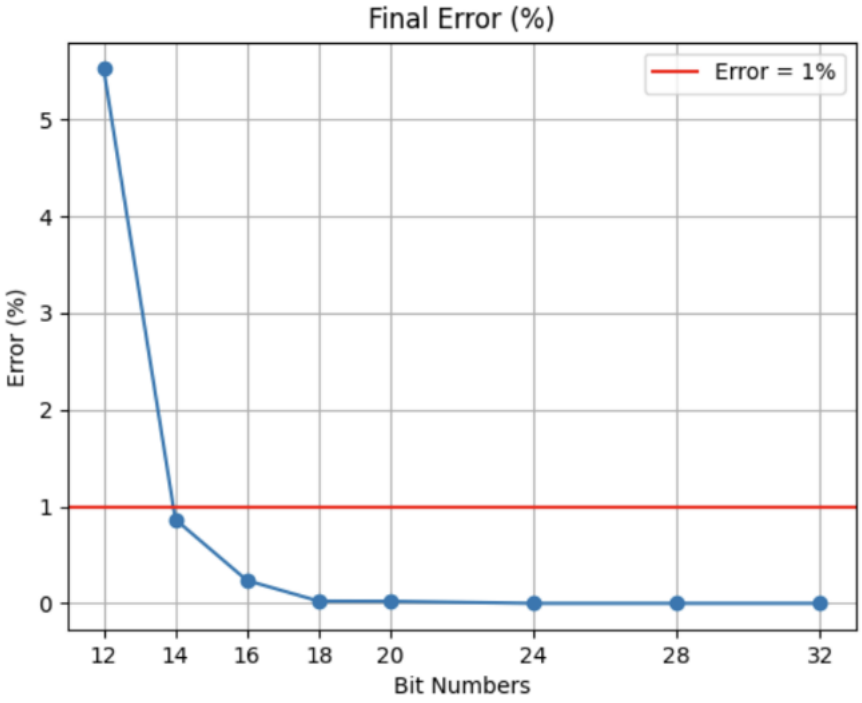

Figura 65 - Erro energético em relação ao total de bits na representação de ponto flutuante (74)

	IIR quantizado	**IIR *lattice***	**IIR ponto flutuante**
Erro Energético	3.54×10^{-4}	3.43×10^{-4}	7.28×10^{-10}
Registradores	49	140	112
DSPs	20 (13%)	28 (18%)	22 (14%)
Lógica (ALMs)	427 (<1%)	589 (1%)	7.867 (14%)

Tabela 6 – Comparação de todos os métodos

Conforme discutido anteriormente, o método em ponto flutuante apresenta o menor erro entre as três abordagens, uma vez que não há quantização dos coeficientes, fator que tende a aumentar o desvio em relação ao resultado ideal. Nesse sentido, o erro energético obtido nos métodos quantizados é da ordem de 10^6 vezes superior ao observado na implementação em ponto flutuante. Ainda assim, um erro na escala de 10^{-4} permanece bastante reduzido diante do objetivo da aplicação considerada neste trabalho.

Considerando o esforço computacional, o método em ponto flutuante é o que apresenta maior consumo de lógica, como era esperado devido à complexidade das operações envolvidas. Enquanto os métodos quantizado e *lattice* consomem menos de 1% e cerca de 1% da lógica disponível, respectivamente, o método em ponto flutuante consome aproximadamente 14%, representando um aumento significativo. Esse crescimento pode se

tornar um fator limitante a depender do dispositivo escolhido, pois a implementação do simulador completo envolve diversos outros módulos além do filtro. Além disso, a análise apresentada aqui corresponde apenas ao circuito utilizado para obter a resposta ao impulso do *shaper*, e não ao conjunto completo do sistema.

Os métodos de IIR quantizado e em estrutura *lattice* mostram-se boas alternativas quando se busca um equilíbrio entre erro e esforço computacional. Entre eles, o *lattice* utiliza ligeiramente mais lógica e registradores, porém tende a ser mais robusto em termos de estabilidade após a quantização, uma vez que sua parametrização pode ser menos sensível a perturbações dos coeficientes do que a realização direta convencional.

Portanto, em resumo, caso o objetivo seja obter o menor erro possível e haja disponibilidade suficiente de *hardware* para a implementação, o método em ponto flutuante é o mais indicado. Isso ocorre porque a ausência de quantização dos coeficientes reduz os erros numéricos e tende a preservar com maior fidelidade a resposta do filtro, mantendo as propriedades de estabilidade do sistema (74).

Caso o objetivo seja minimizar o consumo de recursos computacionais, mantendo ainda uma boa robustez em relação à quantização, a estrutura *lattice* surge como uma alternativa adequada. Por outro lado, quando a prioridade é reduzir ao máximo o uso de lógica, o filtro IIR convencional quantizado torna-se a opção mais simples, pois apresenta o menor consumo de recursos entre as abordagens avaliadas e um erro energético próximo ao obtido com o método *lattice*.

6 CONCLUSÃO

Neste trabalho, o principal objetivo foi a implementação de um simulador de pulsos digitais capaz de reproduzir, de forma controlada e reprodutível, sinais representativos das condições esperadas para a Fase II do LHC, servindo como base para análises sistemáticas do processamento digital sob efeitos de *pile-up* e ruído. A partir dessa plataforma, foi apresentada a implementação e avaliação de diferentes métodos de filtragem digital aplicados ao estágio de *shaper* do sistema de leitura do *TileCal*, com foco na implementação em FPGA. A motivação central esteve associada às exigências impostas pelo cenário da Fase II do LHC, no qual o aumento das taxas de eventos e das condições de ocupação torna necessária a adoção de soluções digitais mais robustas, flexíveis e compatíveis com as limitações de recursos de hardware.

Inicialmente, foi discutido o contexto experimental no qual o TileCal está inserido, bem como a evolução de sua eletrônica de leitura e a necessidade de um simulador de pulsos capaz de reproduzir de forma realista as condições operacionais esperadas. Nesse sentido, o simulador desenvolvido mostrou-se uma ferramenta adequada para a geração de sinais representativos, permitindo ter a base para análise comparativa das diferentes estratégias de filtragem em um ambiente controlado e reprodutível.

No que se refere aos métodos estudados, foram implementadas três abordagens principais de um filtro IIR: a implementação convencional com coeficientes quantizados, o método *lattice* e a implementação em ponto flutuante baseada na representação adaptada do IEEE. Cada uma dessas soluções apresenta características distintas em termos de complexidade de hardware, sensibilidade à quantização e comportamento numérico. Os resultados obtidos evidenciam que a escolha do método de filtragem envolve um compromisso direto entre precisão numérica, estabilidade e consumo de recursos lógicos.

A implementação em ponto flutuante mostrou-se a alternativa mais robusta do ponto de vista da precisão, uma vez que elimina os efeitos associados à quantização dos coeficientes e dos estados internos, preservando a estabilidade do sistema. No entanto, essa abordagem implica em um aumento significativo da complexidade do circuito e da utilização de recursos de FPGA. Por outro lado, o método *lattice* apresentou maior tolerância à quantização quando comparado ao IIR convencional, mantendo um nível de erro reduzido e oferecendo maior previsibilidade em termos de estabilidade, ainda que com um custo moderado de recursos adicionais. Já o filtro IIR convencional destacou-se pela simplicidade de implementação e pelo menor consumo de lógica, configurando-se como uma solução viável quando as restrições de hardware são predominantes.

De forma geral, os resultados indicam que não existe uma solução única que seja ótima para todos os cenários. A escolha do método mais adequado depende diretamente dos requisitos da aplicação, especialmente no que diz respeito à precisão desejada, à

estabilidade numérica e à disponibilidade de recursos computacionais. Assim, as análises apresentadas neste trabalho fornecem subsídios importantes para a tomada de decisão em projetos futuros envolvendo filtragem digital em sistemas de aquisição de dados de alta energia.

Como perspectivas de trabalhos futuros, destacam-se a validação das arquiteturas propostas com dados reais provenientes do detector, a extensão das implementações para filtros de ordem superior e a investigação de arquiteturas híbridas que conciliem técnicas de ponto fixo e ponto flutuante. Além disso, a integração do simulador de pulsos com cadeias completas de reconstrução digital pode contribuir para estudos mais aprofundados sobre desempenho, latência e impacto na qualidade da medida de energia.

REFERÊNCIAS

1 M. Thomson, *Modern Particle Physics*, Cambridge University Press, Cambridge, 2013.

2 R. L. Workman *et al.* (Particle Data Group), "Review of Particle Physics", *Progress of Theoretical and Experimental Physics*, vol. 2024, no. 8, 083C01, 2024. Disponível em: `https://pdg.lbl.gov`

3 P. W. Higgs, "Broken Symmetries and the Masses of Gauge Bosons", *Physical Review Letters*, vol. 13, pp. 508–509, 1964.

4 G. Aad *et al.* (ATLAS Collaboration), "Observation of a New Particle in the Search for the Standard Model Higgs Boson with the ATLAS Detector at the LHC", *Physics Letters B*, vol. 716, pp. 1–29, 2012.

5 S. Chatrchyan *et al.* (CMS Collaboration), "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC", *Physics Letters B*, vol. 716, pp. 30–61, 2012.

6 CERN Courier, *The birth of CERN*, vol. 44, no. 8, 2014. Disponível em: `https://cds.cern.ch/record/1733676/files/vol44-issue8-p011-e.pdf`. Acesso em: outubro de 2025.

7 CERN. *Our History*. Disponível em: `https://home.cern/about/who-we-are/our-history`. Acesso em: outubro de 2025.

8 A. Einstein. *Does the Inertia of a Body Depend Upon Its Energy Content?* Annalen der Physik, vol. 18, pp. 639–641, 1905.

9 D. Griffiths. *Introduction to Elementary Particles*. 2nd Edition, Wiley-VCH, 2008.

10 CERN. *How Particle Accelerators Work*. Disponível em: `https://home.cern/science/engineering/accelerators`. Acesso em: outubro de 2025.

11 K. Cranmer et al. *The ATLAS Experiment at the CERN Large Hadron Collider*. JINST 3 (2008) S08003.

12 CERN. *The Synchrocyclotron: CERN's First Accelerator*. Disponível em: `https://home.cern/science/accelerators/synchrocyclotron`. Acesso em: outubro de 2025.

13 CERN. *The Proton Synchrotron*. Disponível em: `https://home.cern/science/accelerators/proton-synchrotron`. Acesso em: outubro de 2025.

14 CERN. *The Intersecting Storage Rings*. Disponível em: `https://home.cern/science/accelerators/intersecting-storage-rings`. Acesso em: outubro de 2025.

15 CERN. *The Super Proton Synchrotron*. Disponível em: `https://home.cern/science/accelerators/super-proton-synchrotron`. Acesso em: outubro de 2025.

16 CERN. *The Large Electron–Positron Collider*. Disponível em: https://home.cern/science/accelerators/large-electron-positron-collider. Acesso em: outubro de 2025.

17 CERN. *The Large Hadron Collider*. Disponível em: https://home.cern/science/accelerators/large-hadron-collider. Acesso em: outubro de 2025.

18 Particle Data Group. *Review of Particle Physics*. Prog. Theor. Exp. Phys. 2024, 083C01 (2024). Disponível em: https://pdg.lbl.gov. Acesso em: outubro de 2025.

19 G. Arnison *et al.* (UA1 Collaboration). *Experimental observation of isolated large transverse energy electrons with associated missing energy at* $\sqrt{s} = 540$ *GeV*. Phys. Lett. B 122, 103–116 (1983).

20 M. Banner *et al.* (UA2 Collaboration). *Observation of single isolated electrons of high transverse momentum in events with missing transverse energy at the CERN* $\bar{p}p$ *collider*. Phys. Lett. B 122, 476–485 (1983).

21 The Nobel Prize in Physics 1984. *Awarded to Carlo Rubbia and Simon van der Meer for their decisive contributions to the large project, which led to the discovery of the field particles W and Z*. Disponível em: https://www.nobelprize.org/prizes/physics/1984/summary/. Acesso em: outubro de 2025.

22 Wikimedia Commons. *File: CERN accelerator complex (cropped 2).jpeg — Wikimedia Commons, the free media repository*. 2024. Disponível em: https://commons.wikimedia.org/w/index.php?title=File:CERN_accelerator_complex_(cropped_2).jpeg&oldid=842653427. Acesso em: novembro de 2025.

23 L. Evans and P. Bryant, *LHC Machine*, *Journal of Instrumentation*, vol. 3, p. S08001, 2008. doi:10.1088/1748-0221/3/08/S08001.

24 CERN, *The Large Hadron Collider*, CERN Brochure, CERN-Brochure-2021-004-Eng, Geneva, 2021. Disponível em: https://cds.cern.ch/record/2809109/files/CERN-Brochure-2021-004-Eng.pdf.

25 AC Team, *The four main LHC experiments*, unpublished (CERN CDS record), 1999. Disponível em: https://cds.cern.ch/record/40525.

26 CERN, *ALICE*. Disponível em: https://home.cern/science/experiments/alice. Acesso em: novembro de 2025.

27 CERN, *LHCb*. Disponível em: https://home.cern/science/experiments/lhcb. Acesso em: novembro de 2025.

28 CERN, *CMS*. Disponível em: https://home.cern/science/experiments/cms. Acesso em: novembro de 2025.

29 CERN, *ATLAS*. Disponível em: https://home.cern/science/experiments/atlas. Acesso em: novembro de 2025.

30 ATLAS Collaboration, “The ATLAS Experiment at the CERN Large Hadron Collider,” *Journal of Instrumentation*, vol. 3, no. 08, p. S08003, 2008. doi: 10.1088/1748-0221/3/08/S08003.

31 J. Pequenao, “Computer generated image of the whole ATLAS detector,” CERN, Geneva, 2008. [Online]. Disponível em: `https://cds.cern.ch/record/1095924`.

32 ATLAS Collaboration, “ATLAS Public Results and Detector Description,” ATLAS Experiment, CERN, 2024. [Online]. Disponível em: `https://atlas.cern`.

33 ATLAS Collaboration, “ATLAS Inner Detector: Technical Design Report,” CERN-LHCC-97-016, Geneva, 1997. [Online]. Disponível em: `https://cds.cern.ch/record/331063`.

34 ATLAS Collaboration, “ATLAS Liquid Argon Calorimeter: Technical Design Report,” CERN-LHCC-96-041, Geneva, 1996. [Online]. Disponível em: `https://cds.cern.ch/record/331061`.

35 ATLAS Collaboration, “ATLAS Tile Calorimeter: Technical Design Report,” CERN-LHCC-96-042, Geneva, 1996. [Online]. Disponível em: `https://cds.cern.ch/record/331062`.

36 J. Pequenao, “Computer generated image of the ATLAS Calorimeter,” CERN, Geneva, 2008. [Online]. Disponível em: `https://cds.cern.ch/record/1095927`.

37 ATLAS Collaboration, “ATLAS Muon Spectrometer: Technical Design Report,” CERN-LHCC-97-022, Geneva, 1997. [Online]. Disponível em: `https://cds.cern.ch/record/331068`.

38 J. Pequenao, “Computer generated image of the ATLAS Muons subsystem,” CERN, Geneva, 2008. [Online]. Disponível em: `https://cds.cern.ch/record/1095929`.

39 J. Pequenao and P. Schaffner, “How ATLAS detects particles: diagram of particle paths in the detector,” CERN, Geneva, 2013. [Online]. Disponível em: `https://cds.cern.ch/record/1505342`.

40 B. S. M. Peralva, L. M. A. Filho, A. S. Cerqueira, and J. M. Seixas, “The TileCal Energy Reconstruction for Collision Data Using the Matched Filter,” ATL-TILECAL-PROC-2013-023, CERN, Geneva, 2013. [Online]. Disponível em: `https://cds.cern.ch/record/1629575`.

41 K. Anderson, J. E. Pilcher, H. Sanders, F. Tang, S. Berglund, C. Bohm, K. Jon-And, G. Blanchot, and M. Cavalli-Sforza, “Front-end Electronics for the ATLAS Tile Calorimeter,” in *Proceedings of the 6th Workshop on Electronics for LHC Experiments*, CERN, Geneva, 1998. [Online]. Disponível em: `https://api.semanticscholar.org/CorpusID:59415839`.

42 R. Febbraro, “Calibration and signal reconstruction in the ATLAS Tile Hadronic Calorimeter,” ATL-TILECAL-PROC-2010-006, CERN, Geneva, 2011. doi: 10.1016/j.nuclphysbps.2011.04.048. [Online]. Disponível em: `https://cds.cern.ch/record/1281912`.

43 ATLAS TDAQ Collaboration, “ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report,” CERN-LHCC-2003-022, Geneva, 2003. [Online]. Available: https://cds.cern.ch/record/616089.

44 A. Tropina (ATLAS Collaboration), *ATLAS Tile calorimeter: Performance and upgrade for HL-LHC conditions*, CERN, Geneva, 2025. Report No. ATL-TILECAL-PROC-2025-017. Disponível em: https://cds.cern.ch/record/2948883.

45 CERN. *High-Luminosity LHC*. Disponível em: https://home.cern/science/accelerators/high-luminosity-lhc. Acesso em: novembro de 2025.

46 O. Solovyanov (ATLAS Collaboration), *Upgrade of ATLAS Hadronic Tile Calorimeter for the High Luminosity LHC*, CERN, Geneva, 2025. Report No. ATL-TILECAL-PROC-2024-006. DOI: 10.22323/1.469.0263. Disponível em: https://cds.cern.ch/record/2903985.

47 P. Rapheeha (ATLAS Collaboration), *Upgrade of ATLAS Hadronic Tile Calorimeter for the High Luminosity LHC*, CERN, Geneva, 2025. Report No. ATL-TILECAL-PROC-2025-014. Disponível em: https://cds.cern.ch/record/2948495.

48 D. Oliveira Goncalves (ATLAS Collaboration), *Energy reconstruction of the ATLAS Tile Calorimeter under high pile-up conditions using the Wiener Filter*. ATL-TILECAL-PROC-2019-002, CERN, Geneva, 2019. Disponível em: https://cds.cern.ch/record/2674807.

49 ATLAS Collaboration, *ATLAS Tile Calorimeter — Approved Public Plots*. Disponível em: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsTile. Acesso em: Novembro de 2025.

50 E. Fullana *et al.*, “Digital Signal Reconstruction in the ATLAS Hadronic Tile Calorimeter”, *IEEE Transactions on Nuclear Science*, vol. 53, no. 4, pp. 2139–2143, 2006. doi: 10.1109/TNS.2006.877267.

51 F. C. Luna *et al.*, “REAL-TIME FPGA-BASED SIMULATOR FOR THE TILE CALORIMETER READOUT SYSTEM IN THE ATLAS EXPERIMENT,” in *Anais do Encontro Nacional de Modelagem Computacional e Encontro de Ciência e Tecnologia de Materiais*, Ilhéus–BA, Hotel Praia do Sol, 2024. Disponível em: https://www.even3.com.br/anais/enmc2024/920998-REAL-TIME-FPGA-BASED-SIMULATOR-FOR-THE-TILE-CALORIMETER-READOUT-SYSTEM-IN-THE-ATLAS-EXPERIMENT. Acesso em: 24 nov. 2025.

52 D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Reading, MA, 1997.

53 S. K. Park and K. W. Miller, “Random Number Generators: Good Ones Are Hard to Find,” *Communications of the ACM*, vol. 31, no. 10, pp. 1192–1201, 1988.

54 P. L’Ecuyer, “Random Numbers for Simulation,” *Communications of the ACM*, vol. 33, no. 10, pp. 85–97, 1990.

55 T. G. Lewis and W. H. Payne, “Generalized Feedback Shift Register Pseudorandom Number Algorithms,” *Journal of the ACM*, vol. 20, no. 3, pp. 456–468, 1973.

56 ATLAS Collaboration, “Operation and performance of the ATLAS Tile Calorimeter in Run 2,” *JINST*, vol. 16, p. P12023, 2021. doi: 10.1088/1748-0221/16/12/P12023.

57 S. M. Ross, *Introduction to Probability Models*, 11th ed., Academic Press, 2014.

58 A. M. Law, *Simulation Modeling and Analysis*, 5th ed., McGraw-Hill, 2014.

59 PASCHOALIN, T. C. A.; QUIRINO, T. M.; ANDRADE FILHO, L. M. de A. Implementação em FPGA do circuito do condicionamento de sinal do calorímetro hadrônico do ATLAS. In: *Anais do Simpósio Brasileiro de Automação Inteligente / Simpósio Brasileiro de Sistemas Elétricos (SBAI/SBSE)*, 2025.

60 ANDERSON, K.; PILCHER, J.; SANDERS, H.; TANG, F.; BERGLUND, S.; BOHM, C.; HOLMGREN, S. O.; JON-AND, K.; BLANCHOT, G.; CAVALLI-SFORZA, M. Front-end electronics for the ATLAS Tile Calorimeter. 1998.

61 A. V. Oppenheim, R. W. Schafer e J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed., Upper Saddle River, NJ, USA: Prentice Hall, 1999.

62 RABINER, L. R.; GOLD, B. *Theory and Application of Digital Signal Processing*. Englewood Cliffs: Prentice-Hall, 1975.

63 T. W. Parks and C. S. Burrus, *Digital Filter Design*, Topics in Digital Signal Processing, Wiley, 1987.

64 J. O. Smith, *Introduction to Digital Filters with Audio Applications*, online book, 2007. Disponível em: `https://ccrma.stanford.edu/~jos/filters/`. Acesso em: 16 jan. 2026.

65 J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed., Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2007.

66 S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, 4th ed., McGraw-Hill, New York, NY, USA, 2011.

67 AMD (Xilinx), *Infinite Impulse Response Filter Structures in Xilinx FPGAs*, White Paper WP330, 2009.

68 D. Datta and H. S. Dutta, “High performance IIR filter implementation on FPGA,” *Journal of Electrical Systems and Information Technology*, vol. 8, art. no. 2, 2021, doi:10.1186/s43067-020-00025-4.

69 A. Eddla and V. Y. J. Pappu, “Low Area and Power-Efficient FPGA Implementation of Improved AM-CSA-IIR Filter Design for the DSP Application,” *International Journal of Electrical and Electronic Engineering & Telecommunications*, vol. 11, no. 4, pp. 294–303, Jul. 2022, doi:10.18178/ijeetc.11.4.294-303.

70 A. Sharma, T. K. Rawat, and A. Agrawal, “Design and FPGA implementation of lattice wave digital notch filter with minimal transient duration,” *IET Signal Processing*, vol. 14, pp. 440–447, 2020.

71 P. Bharade, Y. Joshi, and R. Manthalkar, “Design and implementation of IIR lattice filter using floating point arithmetic in FPGA,” in *Proc. 2016 Conference on Advances in Signal Processing (CASP)*, 2016, doi:10.1109/CASP.2016.7746188.

72 Typhoon HIL, *FIR Filter*, Typhoon HIL Software Manual, Available at: https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/fir_filter.html, Accessed on: 20 Feb. 2025.

73 National Instruments, *IIR Direct Form Structures*, LabVIEW Digital Filter Design Toolkit Documentation, Available at: https://www.ni.com/docs/en-US/bundle/labview-digital-filter-design-toolkit/page/iir-direct-form-structures-digital-filter-des.html, Accessed on: 18 Jan. 2026.

74 F. Luna, L. Andrade, and T. Paschoalin, “Design and Implementation of a Digital IIR Filter with Floating-Point Representation,” in *Proceedings of the IEEE Conference on Hardware/Signal Processing*, pp. 1–6, 2025.

75 HAYKIN, S. *Adaptive Filter Theory*. 4. ed. Upper Saddle River, NJ: Prentice Hall, 2002. ISBN 0-13-090126-1.

76 D. R. Brown III, *ECE503: Digital Signal Processing – Lecture 6: Structures for Discrete-Time Systems (Digital Filter Structures)*, Worcester Polytechnic Institute (WPI), 2012. Disponível em: `https://spinlab.wpi.edu/courses/ece503_2012/6filterstructures.pdf`. Acesso em: 20 jan. 2026.

77 IEEE, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), published Jul. 22, 2019, doi:10.1109/IEEESTD.2019.8766229.

78 SANTOS, V. A. M. *Implementação de circuitos aritméticos em ponto flutuante, utilizando formato com número de bits configurável*. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica, Robótica e Automação Industrial), Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora, 2017.

79 Intel Corporation, *Fitter Resource Usage Summary Report, Quartus Prime Help (Standard Edition) 17.0*. Disponível em: `https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/mapIdTopics/mwh1465496451103.htm`. Acesso em: 23 jan. 2026.

ANEXO A – Código em *Verilog* de um filtro IIR quantizado

```
1   module shaper
2   #(
3           parameter BITS_IN = 34,
4           parameter G_ENTRADA = 2**32,
5           parameter G_SAIDA_LOG = 10
6   )
7   (
8           input  clock,
9           input  signed [BITS_IN-1:0] in,
10          output signed [BITS_IN+14:0] out
11  );
12
13
14  wire signed [BITS_IN+14:0] out1, out2, out3, out4, out5, out6;
15
16  ////////////////////////////
17  iir_ordem1
18  #(
19          .BITS_IN(BITS_IN),.G_ENTRADA(G_ENTRADA),.G_SAIDA_LOG(10),
20          .b0(-3), .a1(-1022)
21  ) iir1
22  (
23          .clock(clock), .in(in), .out(out1)
24  );
25
26  ////////////////////////////
27  iir_ordem2
28  #(
29          .BITS_IN(BITS_IN), .G_ENTRADA(G_ENTRADA), .G_SAIDA_LOG(10),
30          .b0(746),.b1(444), .a1(1074),.a2(296)
31  ) iir2
32  (
33          .clock(clock), .in(in),.out(out2)
34  );
35
36  ////////////////////////////
37  iir_ordem2
38  #(
39          .BITS_IN(BITS_IN), .G_ENTRADA(G_ENTRADA), .G_SAIDA_LOG(10),
40          .b0(-3362), .b1(-361), .a1(-29), .a2(167)
```

```
41      ) iir3
42      (
43              .clock(clock), .in(in), .out(out3)
44      );
45
46
47      ////////////////////////////
48      iir_ordem1
49      #(
50              .BITS_IN(BITS_IN), .G_ENTRADA(G_ENTRADA), .G_SAIDA_LOG(10),
51              .b0(2644), .a1(-373)
52      ) iir4
53      (
54              .clock(clock), .in(in),.out(out4)
55      );
56
57      ////////////////////////////
58
59      iir_ordem2
60      #(
61              .BITS_IN(BITS_IN), .G_ENTRADA(G_ENTRADA), .G_SAIDA_LOG(10),
62              .b0(-24), .b1(-0), .a1(0), .a2(0)
63      ) iir5
64      (
65          .clock(clock), .in(in), .out(out5)
66      );
67
68      assign out = out1 + out2 + out3 + out4 + out5;
69
70      endmodule
71
```

ANEXO B – Função para filtro IIR genérico no *Matlab*

```
function [y,B,A] = iir_generico(B,A,N,Gx,Gy)

x = zeros(N,1);
x(1) = 1;
x = x * Gx;

B = floor(B*Gy);
A = floor(A*Gy);

%y[n] = Zn1(1)*x[n] + Zn1(2)*x[n-1] - Zd1(2)*y[n-1]

y = zeros(N,1);
rx = zeros(length(B),1);
ry = zeros(length(A)-1,1);

for n = 1:N
  yz = 0;

  rx(1) = x(n);
  for i = 1:length(B)
    yz = B(i)*rx(i) + yz;
  end

  if length(B) > 1
    for i = length(B):-1:2
      rx(i) = rx(i-1);
    end
  end

  yp = 0;

  if length(A) > 1
    for i = 2:length(A)
      yp = A(i)*ry(i-1) + yp;
    end
  end

  y(n) =  yz- yp;

  if length(A) > 1
```

```
41          for i = length(A):-1:2
42            ry(i) = ry(i-1);
43          end
44        end
45
46        ry(1) = fix(y(n)/Gy);
47
48      end
49
50      end
51
```

ANEXO C – Código em *Verilog* para implementação de filtro IIR quantizado na estrutura *lattice*

```
1   module lattice_filter
2   #(
3           parameter BITS_IN = 34,
4           parameter G_ENTRADA = 2**32,
5           parameter FB = 10
6   )
7   (
8           input  clock,
9           input  signed [BITS_IN-1:0] in,
10          output signed [BITS_IN+14:0] out
11  );
12
13  wire signed [BITS_IN+14:0] out1, out2, out3, out4, out5, out6, out7;
14
15  // ----------------- Filtro 1 -----------------
16
17  lattice_1ordem
18  #(
19          .NBITS_IN(BITS_IN),
20          .FB(FB),
21          .k(-1022),
22          .v1(0),
23          .v2(-4)
24  )lattice1
25  (
26          .clock(clock),
27          .in(in),
28          .out(out1)
29  );
30
31  // ----------------- Filtro 2 -----------------
32
33  lattice_2ordem
34  #(
35          .NBITS_IN(BITS_IN),
36          .FB(FB),
37          .k1(833),
38          .k2(295),
39          .v1(0),
```

```
40              .v2(444),
41              .v3(384)
42
43      )lattice2
44      (
45              .clock(clock),
46              .in(in),
47              .out(out2)
48      );
49
50      // ----------------- Filtro 3 -----------------
51
52      lattice_2ordem
53      #(
54              .NBITS_IN(BITS_IN),
55              .FB(FB),
56              .k1(-25),
57              .k2(167),
58              .v1(0),
59              .v2(-362),
60              .v3(-3371)
61
62      )lattice3
63      (
64              .clock(clock),
65              .in(in),
66              .out(out3)
67      );
68
69      // ----------------- Filtro 4 -----------------
70
71      lattice_1ordem
72      #(
73              .NBITS_IN(BITS_IN),
74              .FB(FB),
75              .k(-374),
76              .v1(0),
77              .v2(2643)
78      )lattice4
79      (
80              .clock(clock),
81              .in(in),
```

```
82              .out(out4)
83      );
84
85      // ----------------- Filtro 5 -----------------
86
87      lattice_2ordem
88      #(
89              .NBITS_IN(BITS_IN),
90              .FB(FB),
91              .k1(-1),
92              .k2(0),
93              .v1(0),
94              .v2(3),
95              .v3(-5617560)
96
97      )lattice5
98      (
99              .clock(clock),
100             .in(in),
101             .out(out5)
102     );
103
104     // ----------------- Filtro 6 -----------------
105
106     lattice_2ordem
107     #(
108             .NBITS_IN(BITS_IN),
109             .FB(FB),
110             .k1(-1),
111             .k2(0),
112             .v1(0),
113             .v2(1),
114             .v3(5488266)
115
116     )lattice6
117     (
118             .clock(clock),
119             .in(in),
120             .out(out6)
121     );
122
123     // ----------------- Filtro 7 -----------------
```

```
124
125     lattice_1ordem
126     #(
127             .NBITS_IN(BITS_IN),
128             .FB(FB),
129             .k(-1),
130             .v1(0),
131             .v2(129268)
132     )lattice7
133     (
134             .clock(clock),
135             .in(in),
136             .out(out7)
137     );
138
139     assign out = out1 + out2 + out3 + out4 + out5 + out6 + out7;
140
141     endmodule
```

ANEXO D – Código em *Python* para conversão de um número para o formato de ponto flutuante proposto

```python
    def f2mf(va: str, nbmant: int, nbexpo: int) -> int:
        f = float(va)

        if f == 0.0:
            return 1 << (nbmant + nbexpo - 1)

        ifl = struct.unpack('!I', struct.pack('!f', f))[0]

        s = (ifl >> 31) & 0x00000001
        e = ((ifl >> 23) & 0xFF) - 127 - 22
        m = ((ifl & 0x007FFFFF) + 0x00800000) >> 1

        s = s << (nbmant + nbexpo)

        e = e + (23 - nbmant)
        sh = 0

        while e < -math.pow(2, nbexpo - 1):
            e += 1
            sh += 1

        e = e & (int(math.pow(2, nbexpo)) - 1)
        e = e << nbmant

        if nbmant == 23:
            if ifl & 0x00000001:
                m += 1
        else:
            sh = 23 - nbmant + sh
            carry = (m >> (sh - 1)) & 0x00000001
            m = m >> sh
            if carry:
                m += 1

        return s + e + m
```