UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA - CEEI
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA - UAEE

MARLEY LOBÃO DE SOUSA

Trabalho de Conclusão de Curso

ANÁLISE COMPARATIVA DE IMPLEMENTAÇÕES EM FPGA DE UMA APLICAÇÃO DE APRENDIZAGEM DE MÁQUINA UTILIZANDO A ARITMÉTICA POSIT E O PADRÃO IEEE-754

Campina Grande, PB
7 de abril de 2022

MARLEY LOBÃO DE SOUSA

ANÁLISE COMPARATIVA DE IMPLEMENTAÇÕES EM FPGA DE UMA APLICAÇÃO DE APRENDIZAGEM DE MÁQUINA UTILIZANDO A ARITMÉTICA POSIT E O PADRÃO IEEE-754

Trabalho de Conclusão de Curso submetido à Universidade Federal de Campina Grande, como parte dos requisitos necessarios para obtenção do grau de Bacharel em ciências no domínio da Engenharia Eletrica.

Professor Orientador: Gutemberg Gonçalves dos Santos Júnior D.Sc
Professor Avaliador: Marcos Ricardo Alcântara Morais D.Sc

Campina Grande, PB
7 de abril de 2022

MARLEY LOBÃO DE SOUSA

ANÁLISE COMPARATIVA DE IMPLEMENTAÇÕES EM FPGA DE UMA APLICAÇÃO DE APRENDIZAGEM DE MÁQUINA UTILIZANDO A ARITMÉTICA POSIT E O PADRÃO IEEE-754

Trabalho de Conclusão de Curso submetido à Universidade Federal de Campina Grande, como parte dos requisitos necessarios para obtenção do grau de Bacharel em ciências no domínio da Engenharia Eletrica.

Aprovado em: ____/____/____

Gutemberg Gonçalves dos Santos Júnior D.Sc
Professor Orientador

Marcos Ricardo Alcântara Morais D.Sc
Professor Avaliador

Campina Grande, PB
7 de abril de 2022

Dedico este trabalho aos meus pais, Marco e Raquel.

AGRADECIMENTOS

Primeiramente agradeço a Deus, que em toda sua plenitude, me criou com inteligência e capacidade suficiente para estudar, compreender e admirar a sua bela natureza e seus processos físicos.

Deixo meus agradecimentos para meus pais, Marco e Raquel, minhas irmãs, Mirley e Maísa, e minhas tias, Cássia e Dalva, por todo carinho, incentivo e colaboração durante meu processo de crescimento pessoal e profissional durante minha graduação.

Também agradeço à minha namorada, Geovanna, por sempre estar ao meu lado em todos os momentos, compartilhar dificuldades e conquistas, durante os desafios que tenho enfrentado.

Deixo meus agradecimentos também a Lyang Medeiros, Murilo Santos, Breno Alencar, Samuel Cesarino, Matheus Morais, Mateus Pereira e Andresso da Silva, pois apesar de nosso período de formação não ter sido nada fácil, vocês fizeram com que a experiência fosse, sem dúvida, leve, feliz e diferenciada.

Agradeço a Lucas Farias, pela ajuda na resolução dos problemas enfrentados durante o aprendizado das ferramentas das quais eu ainda não havia tido contato, e por toda motivação em continuar estudando microeletrônica.

Agradeço ao professor Gutemberg Júnior, por toda a sua orientação neste projeto, pela assistência ao também me orientar como monitor, pelo conhecimento passado nas displinas do curso de graduação, pelo incentivo no meu interesse na área de microeletrônica, bem como, por disponibilizar as ferramentas necessárias para desenvolvimento deste trabalho.

Eu via sempre o Senhor comigo porque ele está ao meu lado direito, para que nada me deixe abalado. Por isso o meu coração está feliz, e as minhas palavras são palavras de alegria; e eu, um ser mortal, vou descansar cheio de esperança, pois tu, Senhor, não me abandonarás no mundo dos mortos. Eu tenho te servido fielmente, e por isso não deixarás que eu apodreça na sepultura. Tu me tens ensinado os caminhos que levam à vida, e a tua presença me encherá de alegria.

Atos 2:25-28

RESUMO

Esse projeto apresenta uma comparação entre diferentes representações numéricas de computadores em termos do número de células lógicas, tempo e potência, ao serem aplicadas a módulos de aprendizagem de máquina apenas no cenário de FPGA. Efetuou-se a comparação por meio da análise de razões entre as características do *hardware* gerado pelo sintetizador do Intel® Quartus® Prime. Dessa forma, validações foram realizadas em relação a quantidade de elementos lógicos e frequência máxima alcançada, ao se utilizar blocos aritméticos de soma e multiplicação. Além disso, o treino de redes neurais por meio de Posits ou ponto fixo, ambos com 16 bits para representação, produzindo a mesma acurácia final também foi validada. Ao comparar consumo de potência entre Posits e *floats* com a mesma quantidade de bits, o Posit não se mostrou tão vantajoso. Porém, comparando consumo de potência entre Posits e *floats*, com os *floats* possuindo uma maior quantidade de bits, os resultados se mostraram muito promissores para a aritmética Posit. Contudo, para a aplicação de rede neural, as métricas de elementos lógicos, tempo e potência apresentaram melhores resultados para o Posit, a medida em que se aumentou a quantidade total de bits para representação. Portanto, no contexto de FPGA a aritmética Posit possui um maior custo de recursos, o que pode nortear a decisão de uso ou não de aceleradores de *hardware* na aplicação de interesse.

Palavras-chave: *Hardware*, FPGA, Posit, IEEE-754, Aprendizagem de Máquina.

ABSTRACT

This project presents a comparison between different numerical representations of computers in terms of the number of logic cells, time and power, when applied to machine learning modules only in the FPGA context. Comparison was made by analysis of the ratio between the characteristics of the hardware generated by the Intel® Quartus® Prime synthesizer. This way, validations were made in relation to the quantity of logical elements and maximum frequency reached, when using adder and multiplier arithmetic blocks. In addition, the training of neural networks by means of Posits or fixed point, both with 16 bits for representation, producing the same final accuracy was also validated. When comparing power consumption between Posits and floats with the same amount of bits, Posit did not prove to be as advantageous. However, comparing power consumption between Posits and floats, with the floats having a larger amount of bits, the results proved very promising for Posit arithmetic. However, for the neural network application, the logic elements, time, and power metrics showed better results for Posit as the total amount of bits for representation was increased. Therefore, in the FPGA context Posit arithmetic has a higher resource cost, which may guide the decision of whether or not to use hardware accelerators in the application of interest.

Keywords: Hardware, FPGA, Posit, IEEE-754, Machine Learning.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

IEEE - *Institute of Electrical and Electronics Engineers*

IEEE-754 - Norma padronizada pelo IEEE para representação de números reais

ES - *Expoent Size*

PACoGen - *Posit Arithmetic Core Generator*

FPGA - *Field Programmable Gate Array*

FPU - *Floating Point Unit*

Unum - *Universal Number*

LOD - *Leading One Detector*

LZD - *Leading Zero Detector*

RTL - *Register Transfer Level*

HDL - *Hardware Description Language*

OTP - *One Time Programmable*

BRAM - *Block Random Access Memory*

SoC - *System On Chip*

CPLD - *Complex Programmable Logic Device*

SDC - *Synopsys Design Constraint*

PTC - *Power and Thermal Calculator*

Sumário

1 INTRODUÇÃO

Nos dias atuais, há uma demanda bastante significativa por desenvolvimento de novas tecnologias que levem a humanidade a alcançar novos patamares. Assim, algumas aplicações que precisam de alta velocidade de processamento de dados e baixo consumo de energia exigem melhor desempenho dos equipamentos utilizados. Isso se torna ainda mais crítico em sistemas que requerem elevada precisão de cálculos, devido a um possível risco inerente a atividade desenvolvida.

Entretanto, muitas vezes, inovações esbarram em alguma barreira temporariamente intransponível. Para continuar avançando, muitas dessas tecnologias já existentes são otimizadas ao máximo, a fim de que o desenvolvimento de produtos melhores seja possível e suficientemente adequados para o propósito da aplicação a que se destinam.

Em ambos os sentidos, descoberta ou otimização de tecnologias, o processamento de informação é muito importante, principalmente no que diz respeito às aplicações que envolvem sistemas embarcados. Pois além desses sistemas necessitarem de aumento no grau de integração e baixo consumo de energia, como dito por Ternovoy et al. (2020), em vários casos, a elevada precisão dos cálculos realizados é imprescindível, além de um custo de *hardware* que seja viável para seu uso.

1.1 JUSTIFICATIVA

Para realizar a representação de grandezas do mundo real nos computadores, se faz uso atualmente dos números em ponto flutuante (*floats*), que possuem sua própria aritmética. Essa representação é descrita no padrão IEEE (em português, Instituto de Engenheiros Eletricistas e Eletrônicos) 754, que determina entre outros itens, a quantidade de bits a serem alocados, quantos deles são separados para a representação do expoente, que representa a flutuação da vírgula, assim como a parte fracionária do número a ser representado.

Na Figura 1, é possível visualizar como exemplo um dos formatos padronizados, a representação de ponto flutuante em precisão simples. O formato de precisão simples se caracteriza por possuir 8 bits para representação do expoente e 23 bits para a mantissa.

Alguns ajustes devem ser feitos para representação de ponto flutuante. O expoente deve ser devidamente somado com um valor bias para o intervalo de representação adequado. Já a mantissa, leva em consideração um bit 1 escondido para diminuir a memória utilizada pelos bits representação. Esse bit deve ser adicionado no momento em que a cadeia de bits é decodificada.

Figura 1: Representação de Precisão Simples na IEEE-754

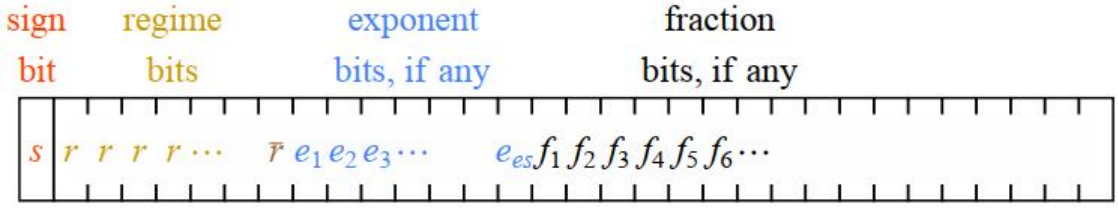
Fonte: Peric et al. (2021)

Portanto, a precisão é determinada pelo espaço reservado para a representação desejada, logo quanto mais precisão é requerida, mais bits devem ser reservados, o que se caracteriza como um problema para sistemas que precisam renunciar precisão para redução de consumo de energia, tal como dito por Fatemi Langroudi et al. (2018).

Dados os problemas citados, entre outros existentes no uso dessa representação padrão, foram desenvolvidas ao longo dos anos, algumas representações alternativas, com o objetivo de melhorar a precisão numérica dos resultados, sem necessitar de um *hardware* maior a ponto de não permitir o desenvolvimento de determinadas aplicações. Uma delas, é a aritmética Posit apresentada por Gustafson (2017).

A aritmética Posit possui vários aspectos positivos, que se sobressaem de maneira significativa sobre o padrão IEEE-754 citado anteriormente. Um deles, é possuir diferentes tamanhos na quantidade de bits alocados para representação, tal como na Figura 2. Entretanto, como diferencial, esse valor não necessita ser alterado para melhorar a precisão dos números envolvidos.

Figura 2: Representação do formato numérico na aritmética Posit



Fonte: Gustafson e Yonemoto (2017)

O melhor uso dos bits destinados a representação de um número real melhora a faixa de variação dinâmica e a precisão (principalmente para valores com módulo entre 0 e 1), o que permite que representações que no IEEE-754 são ditas como *Not a Number* - NaN, *overflow* ou *underflow*, sejam utilizadas para melhorar o resultado obtido nas operações realizadas, de acordo com Gustafson e Yonemoto (2017).

Uma das implementações do sistema numérico Posit é o PACoGen, que se caracteriza por ser um gerador de *hardware* de código fonte aberto capaz de utilizar parâmetros da representação para realizar operações aritméticas básicas.

Portanto, procura-se aproveitar algumas vantagens existentes entre as representações indicadas em termos de precisão, bem como, um *trade-off* viável de consumo no que diz respeito a elementos lógicos, potência e frequência, principalmente aqueles que são executados utilizando recursos dedicados.

O trabalho proposto busca aproveitar o crescente uso da inteligência artificial, para melhorar as aplicações referentes a redes neurais, tais como a de Johnson et al. (2016), que utilizem processamento de linguagem natural, reconhecimento de discurso, como em Amodei et al. (2015), entre outros usos de aprendizagem de máquina, para produzir *hardware* por meio do núcleo gerador da aritmética Posit, o PACoGen, desenvolvido por Jaiswal e So (2019), e assim permitir um melhor entendimento do custo de recursos consumidos pela aritmética Posit, como apresentado por Uguen et al. (2019), no contexto de FPGA.

As seções que seguem estão organizadas da seguinte forma: No seção 2, está apresentada a fundamentação teórica para embasar conceitualmente o trabalho proposto; na seção 3, está apresentada a metodologia, com as ferramentas em que se produziu este trabalho; na seção 4, se encontram os resultados obtidos; na seção 5, estão as discussões acerca das proposições indicadas; na seção 6, estão tecidas as considerações finais; e por fim, na seção 7 as referências utilizadas.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Realizar análise comparativa de implementações em FPGA de uma aplicação de aprendizagem de máquina utilizando a aritmética POSIT e o padrão IEEE-754.

1.2.2 Objetivos Específicos

- Realizar revisão de conceitos de aprendizagem de máquina, eletrônica digital, linguagem SystemVerilog HDL, representações IEEE-754 e Posit;
- Projetar e adaptar para diferentes módulos de *hardware*, em FPGA, e fazer comparações em diferentes representações numéricas nas operações aritméticas;
- Avaliar o desempenho do *harware* produzido em relação a características de tempo, área de utilização do chip e consumo de potência;
- Analisar o funcionamento do PACoGen em simulação de FPGA em relação as diferenças entre as implementações produzidas.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão apresentados os conceitos fundamentais necessários para compreender o presente trabalho como um todo. Para isso, serão mostradas diferentes representações numéricas de computadores que foram utilizadas nas simulações, incluindo ponto fixo, norma IEEE-754, bem como a aritmética Posit.

Um núcleo gerador de *hardware* para a aritmética Posit irá ser incluído, com o objetivo de facilitar as operações (soma e multiplicação), nos módulos implementados que a utilizarem. Pois dado que são projetos de certa complexidade, exigem um maior número de operações em comparação com representações binárias mais simples, como ponto fixo, por exemplo.

Também será explicada a aplicação a que se destina, em aprendizado de máquina, mais especificamente em redes neurais, que atualmente viabilizam diversas aplicações de alto nível de complexidade, e que podem se utilizar de tais representações para otimizar ainda mais seus usos.

Será mostrada a tecnologia FPGA e como a mesma pode contribuir muito em avanços significativos por meio de aceleradores de *harware*, servindo como um recurso dedicado para que o sistema receba respostas muito mais rapidamente.

Por fim, a ferramenta Intel® Quartus Prime e seus recursos internos de simulação que serviram para o desenvolvimento, serão indicados dentro do ambiente de produção do código de descrição de *hardware*.

2.1 SISTEMAS DE REPRESENTAÇÃO NUMÉRICA

O sistema binário é a base dos sistemas eletrônicos digitais atualmente. Pois permite a representação em níveis discretos muito bem definidos, em que os valores possuem correspondentes diretos de tensão, como dito por Ranhel (2021). Sendo o nível lógico 0 para o 0V (baixo ou *ground*), e o nível lógico 1 para a tensão requerida pelos circuitos lógicos utilizados (alto ou V_{DD}).

Esse sistema se popularizou rapidamente pincipalmente por dois motivos, primeiro por se caracterizar com uma maior imunidade ao ruído existente nos sinais de tensão elétrica, comparado a outros sitemas. Em segundo lugar, as operações com o sistema binário são bastante simples, facilitando sua implementação e incorporação a diferentes níveis de projeto. Utilizando combinações entre os símbolos citados, é possível representar uma grande quantidade de valores numéricos, sejam pertencentes a diferentes conjuntos, como números naturais, negativos, complexos, entre outros.

Dentre essa gama de possibilidades, se encontram, inclusive, os números reais. Estes por sua vez, são usados para representar uma quantidade contínua que inclui todos os números positivos, negativos e o zero.

Os números reais são na verdade uma expansão dos números racionais, que é o conjunto que representa números representados por uma razão de dois números inteiros, considerando ainda mais os casos em que a razão entre dois números não seja inteiro, mas possua uma parte fracionária que pode até ser infinita.

Entretanto, para fazer uso dos bits 0 e 1, essa parte fracionária não pode ser representada com tamanho infinito em computadores reais, pois nesse caso, seria necessária uma quantidade infinita de memória no circuito digital projetado.

Portanto, para usar números com parte fracionária em computadores, se faz necessário o uso de alguma representações com regras bem definidas para que operações entre valores possam ser executadas, bem como não ocorram problemas na interpretação dos resultados obtidos.

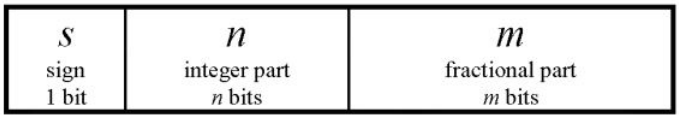
As representações de números reais mais conhecidas e utilizadas atualmente são o ponto fixo e o ponto flutuante, também comnhecidos como *floats*, que serão mostrados mais adiante.

2.1.1 Representação por Ponto Fixo

Uma das maneiras mais comuns de se representar números reais é a representação por ponto fixo. Para representar um valor de N bits em ponto fixo, é preciso dividir essa quantidade em uma parte inteira e outra parte fracionária. Geralmente, o bit mais significativo é separado para representar o sinal do número, embora também seja possível suprimí-lo e operar apenas com números positivos.

Portanto, incluindo um bit *s* de sinal, mais *n* bits para a parte inteira e *m* bits para a parte fracionária, obtemos a configuração apresentada na Figura 3. Vale salientar que os valores são ainda interpretados em complemento de dois.

Figura 3: Representação valor em ponto fixo



Fonte: Peric et al. (2021)

Pala ilustrar o funcionamento do ponto fixo, seus valores extremos serão transformados. Para facilitar a visualização, a parte inteira, os bits entre o bit de sinal e os bits fracionários, serão colocados em negrito.

Então para um número de 16 bits no total, com 3 bits para a parte inteira e 12 bits para a parte fracionária, é sabido que o valor 0**111**000000000000 = 7 é o maior número inteiro positivo que pode ser representado, enquanto que, 1**000**000000000000 = -8 é o menor valor inteiro.

Já o menor valor real positivo, é aquele em apenas o bit menos significativo é 1 e todos os outros são 0, assim 0**000**000000000001 = 2^{-12} = 0,000244140625. Logo, o maior valor real negativo ocorre quando todos os bits são 1, 1**111**111111111111 = -2^{-12} = -0,000244140625. Finalmente, para obter o maior valor real positivo, fazemos todos os btis 1, exceto o de sinal, 0**111**111111111111 = 7,999755859.

Com esses exemplos, podemos notar que nenhum valor além do zero no intervalo entre -0,000244140625 e 0,000244140625, consegue ser representado com 3 bits para a parte inteira e 12 bits para a fracionária. Sendo assim, os valores nesse intervalo são aproximados para o zero, e chamamos essa região de *underflow*, justamente pela memória finita que foi discutida anteriormente para os bits de representação.

De modo análogo, os valores menores que -8 e maiores que 7,999755859, ficam na região denominada de *overflow*, e não conseguem ser representados. Pois seriam necessários mais bits para que sua representação também fosse possível.

Portanto, o que caracteriza o ponto fixo é a posição da vírgula, que para qualquer valor permanece no mesmo ponto da palavra binária. Não há qualquer indicativo simbólico colocado entre os zeros e uns para atestar a presença da vírgula, simplesmente se define onde se localiza e se mantém para execução das operações.

Como exemplo, tomemos o valor 2,75 = 0**010**110000000000, ao multiplicarmos por 2, que pode ser executado realizando um deslocamento para a esquerda em binário, obtemos assim 5,5 = 0**101**100000000000. Ou seja não houve necessidade de maiores alterações na operação na palavra binária por conta da vírgula. Isso traz vantagens em relação a manipulação dos valores, dado que a facilidade de operações aritméticas pode simplificar os circuitos produzidos.

Dessa maneira, o projetista de sistemas que faz uso dessa representação precisa ser cuidadoso o suficiente para perceber as nuances que a mesma traz consigo, bem como, prestar bastante atenção para que os procedimentos matemáticos levem em consideração a presença da vírgula e as casas decimais, para que os computadores consigam representar corretamente os valores informados, que podem ser provenientes de sensores, e os resultados que podem voltar na forma de alteração de grandezas por meio de atuadores, em uma determinada aplicação.

2.1.2 Representação por Ponto Flutuante - Norma IEEE-754

O padrão IEEE-754 descreve formatos de ponto flutuante, uma maneira de representar números reais no *hardware*. A representação em ponto flutuante por vezes é melhor que a de ponto fixo principalmente pela maior precisão oferecida, que por consequência gera menos erros.

Em 1985, o IEEE estabeleceu a norma IEEE-754, que padronizou a representação e operação de números de ponto flutuante para projetistas de computadores e *softwares*. O intuito dessa norma, foi padronizar não só a quantidade de bits, mas também a posição dos bits de expoente e mantissa na palavra binária.

Em 2019, o padrão foi revisado pela última vez e perdura até os dias de hoje (IEEE, 2019). Em vários processadores atuais, há uma unidade FPU, que significa em português, unidade de ponto futuante, que é responsável por cálculos que permitem melhor comunicação entre periféricos, e se utilizam de variações de precisão dessa norma.

Portanto, essa representação é utilizada em muitos processadores modernos, para cálculo de valores no conjunto dos números reais. Para aumentar o intervalo dinânico dos produzidos, ao invés de números em ponto fixo, números de ponto flutuante no padrão IEEE-754 são mais utilizados (Ternovoy et al., 2020).

Nessa representação, de forma semelhante ao ponto fixo, há três campos onde os bits são distribuídos, o sinal de 1 bit, o expoente e a mantissa com a quantidade de bits crescentes à medida que a precisão aumenta. Analogamente a notação científica da matemática, utilizada pra reduzir o tamanho da expressão que simboliza valores extremamente elevados ou pequenos, mudamos apenas a base da multiplicação para 2, em decorrência da variedade de símbolos permitidos.

Logo, a equação 1 é a que calcula os valores numéricos para a norma IEEE-754.

$$valor = (-1)^{sinal} * mantissa * 2^{expoente} \tag{1}$$

Foram estabelecidos alguns tipos de precisão para esta norma, de maneira que, *softwares*, linguagens de programação e *hardware* pudessem usar os formatos determinados como padrão. Esses diferentes tipos são meia (16 bits), simples (32 bits), dupla (64 bits), quádrupla (128 bits), óctupla (256 bits) precisão ou até casos especiais de precisão estendida, que possuem quantidades de bits variadas para diferentes arquiteturas.

Na tabela 1, vemos as referidas precisões. Observe que sempre há um bit de sinal, que tem nível lógico 0 para positivo ou nível lógico 1 para negativo. Além disso, há bem mais bits disponíveis para a mantissa do que para o expoente, dado que, com pouca movimentação da vírgula no número já é possível representar diversos valores.

Tabela 1: Padrões de precisão na IEEE-754

Precisão	Bits de Sinal	Bits de Expoente	Bits de Mantissa	Valor do Bias
16	1	5	10	15
32	1	8	23	127
64	1	11	52	1023
128	1	15	112	16383
256	1	19	236	262143

Fonte: Autoria Própria

Para utilizar essa representação, é importante destacar que o valor dos bits do expoente não podem ser simplesmente colocados na equação 1. Antes disso, se faz necessário subtrair um valor, chamado de bias para correção do mesmo, que também está na tabela 1.

O bias de cada precisão é calculado para todos bits do expoente sendo 1, exceto o primeiro. Por exemplo, para meia precisão (16 bits), são 5 bits de expoente, temos então **0**1111 = 15, já para precisão simples (32 bits), são 8 bits de expoente, logo temos **0**1111111 = 127. Outro fato importante, é que na mantissa há um bit 1 antes da vírgula, que fica escondido, e deve ser adicionado. Esse bit não é colocado na formulação geral. Isso acontece porque todos os valores já o possuem como sendo o primeiro bit 1 em toda a extensão do número em binário, logo seria desperdício armazená-lo. Finalmente, a equação 1, com o valor acrescido no expoente e do primeiro bit 1 da mantissa, se transforma na equação 2.

$$valor = (-1)^{sinal} * (1 + mantissa) * 2^{expoente-bias} \tag{2}$$

É interessante notar que vários bytes são usados para representar um valor em *float*. No caso de meia precisão, 2 bytes, em precisão simples, 4 bytes e assim por diante. Todos esses bits podem ser tratados em alto nível por *software*, ou até em baixo nível, pelo circuito que irá tratar e operar os mesmos, como no caso de processadores de 8 bits, que irão levar em consideração a posição e o significado de cada um deles.

Para encontrar o maior valor representável, fazemos todos os bits 1, exceto o de sinal e o último do expoente. Enquanto que, para o menor número representável, são todos os bits 0, exceto pelo último do expoente.

Com essas regras em mãos, podemos calcular, como exemplos, os maiores e menores valores positivos para o *float* de 16 bits. Para o caso de 16 bits, o maior número representável é 0**11110**1111111111, ou seja, 11110 = 30 no expoente e 1111111111 = 0,9990234375 na mantissa. Colocando na expressão 2, obtemos o resultado na expressão 3.

$$valor = (-1)^0 * (1 + 0{,}9990234375) * 2^{30-15} = 65504 \tag{3}$$

Já para o menor valor representável em 16 bits, temos 0**00001**0000000000, ou seja, 00001 = 1 no expoente e 0000000000 = 0 na mantissa. Colocando na expressão 2, obtemos o resultado na expressão 4.

$$valor = (-1)^0 * (1 + 0) * 2^{1-15} = 0{,}00006103515625 \tag{4}$$

Entretanto, na representação numérica da norma IEEE-754 existem diversos problemas. Um dos principais, é o desperdício de representações, pois em precisão simples (32 bits), existem $16 * 10^6$ maneiras de representar um NaN (*Not a Number*), enquanto que, em precisão dupla (64 bits), existem $9 * 10^{18}$. Um NaN ocorre basicamente quando há exceções nas operações, por exemplo, o resultado de uma divisão por zero (Ternovoy et al., 2020).

Além disso, também de acordo com Ternovoy et al. (2020), é matematicamente incorreta no sentido de haver duas formas para o zero, uma positiva e outra negativa, e haver *overflow* para infinito e *underflow* para 0, o que significa que o erro relativo pode ser infinito ou até haver perda de informação, respectivamente para os dois casos.

E uma semelhança com a representação por ponto fixo é a acurácia constante, pois para uma mesma precisão não há variação da quantidade de bits na mantissa, e assim há um limite na representação.

2.1.3 Aritmética Posit

Em 2015, surgiu uma nova representação em ponto flutuante chamada Unum - *Universal Number*, que contorna algumas da limitações enfrentadas pelo padrão IEEE-754, e que possuem alguns tipos (Gustafson, 2017).

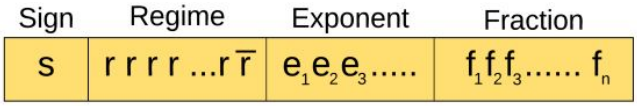
Diferentemente da representação convencional estabelecida pela norma do IEEE, o Unum possui a característica de variar seu comprimento, de modo que consegue se adaptar dinamicamente ao tamanho de bits que foram definidos para a representação do valor real a ser representado.

A representação Unum difere entre números que podem ser representados de forma exata e os que não podem. Em relação aos que não podem ser representados de maneira exata, um intervalo entre os dois números reais adjacentes mais próximos é usado para representa-lo. A aritmética Unum faz diferenciação se os operandos são números representados de forma exata ou intervalos definidos como dito anteriormente (Mee, 2019).

Existem alguns tipos de Unum, neste trabalho iremos focar no tipo III. Mais conhecido como Posit, o Unum tipo III foi projetado para uma substituição direta do *float* padrão IEEE-754. Com um número fixo de bits, os Posit ainda são bastante flexíveis, dado que o número de bits para cada campo definido pode variar, como ainda será mostrado. O grande objetivo é proporcionar simplicidade na implementação tanto de *hardware* quanto de *software* utilizando circuito lógicos para realizar operações básicas de forma mais simples.

Na Figura 4 se encontra a estrutura de uma representação Posit com bits de sinal, regime, exponente e fração. A parte do sinal é igual a norma IEEE-754: 0 para números positivos, 1 para números negativos. Se o valor for negativo, o complemento de 2 deve ser realizado antes de decodificar os demais campos de bits.

Figura 4: Configuração na representação da aritmética Posit



Fonte: Gohil et al. (2021)

O maior diferencial dos Posits está nos bits de regime. Para compreendê-los, considere as combinações binárias mostradas na Figura 5, com o número *k* determinando o comprimento de percurso dos bits, há uma variação do tamanho desse campo para uma mesma representação. (O símbolo ”x” ao tratar-se de bits significa ”não importa”).

Figura 5: Significado da variável *k* no comprimento dos bits de regime

Binary	0000	0001	001x	01xx	10xx	110x	1110	1111
Numerical meaning, k	−4	−3	−2	−1	0	1	2	3

Fonte: Gustafson e Yonemoto (2017)

As cadeias binárias começam com algum bit sendo 0 ou 1 e esse mesmo bit se repete na sequência, terminando o campo de regime quando o próximo bit é invertido ou quando os bits de representação não tem mais capacidade. Seja m o número de bits idênticos na execução; se os bits forem 0, então *k = -m*; se forem 1, então *k = m* - 1.

Os processadores podem ser orientados a ”encontrar primeiro 1” ou ”achar primeiro 0” em *hardware*, e assim a lógica de decodificação de bits de regime está facilmente disponível para uso. O regime indica um fator de escala de $useed^k$, onde $useed = 2^{2^{es}}$ (Gustafson e Yonemoto, 2017).

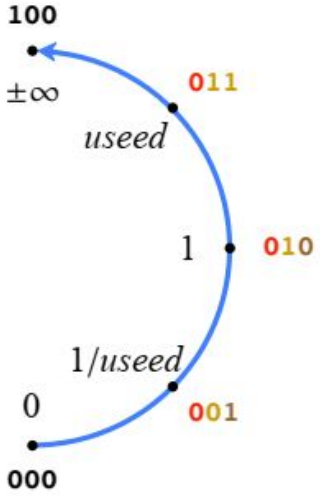
Os próximos bits são os de expoente, considerados como um inteiro sem sinal. Para os Posits, não existe um valor de bias, eles são simplesmente representados por 2^e. A aparição destes bits está totalmente condicionada a quantidade de bits, quantificados por *ES*, que restam à direita dos bits de regime. Portanto, aqui se encontra uma primeira vantagem de utilizar os Posits, a precisão cônica, pois números próximos a 1 em magnitude têm mais precisão do que os valores extremamente grandes ou extremamente pequenos.

Se houverem bits após a cadeia de regime e exponente, eles representam os bits de fração, assim como a fração 1,*f* em *floats* convencionais, com o bit escondido que é sempre 1, como já explicado anteriormente.

Para exemplificar, vamos supor um simples Posit de 3 bits melhor ilustrado na Figura 6. Observe que é visualizado apenas a metade dos reais representados.

Há apenas dois valores de exceção Posit, que são o zero absoluto, com todos os bits iguais a 0 e $\pm\infty$, que seria apenas um bit 1 seguido de todos os demais bits 0, e seus significados nos campos da palavra binária de bits não seguem a notação geral adotada. Para os outros bits na Figura 6, os bits são codificados por cores: vermelho para sinal, amarelos para bits iguais de regime e marrons para bit oposto de regime. Observe que os valores positivos são exatamente $useed^k$ representado pelo regime.

Figura 6: Valores positivos para o Posit de 3 bits



Fonte: Gustafson e Yonemoto (2017)

Algumas propriedades interessantes são que a precisão Posit aumenta com a quantidade de bits e os valores permanecem na mesma localização do círculo quando um bit 0 é aumentado na palavra binária. Aumentando um bit 1 na palavra binária, cria-se um novo valor entre dois Posits no círculo.

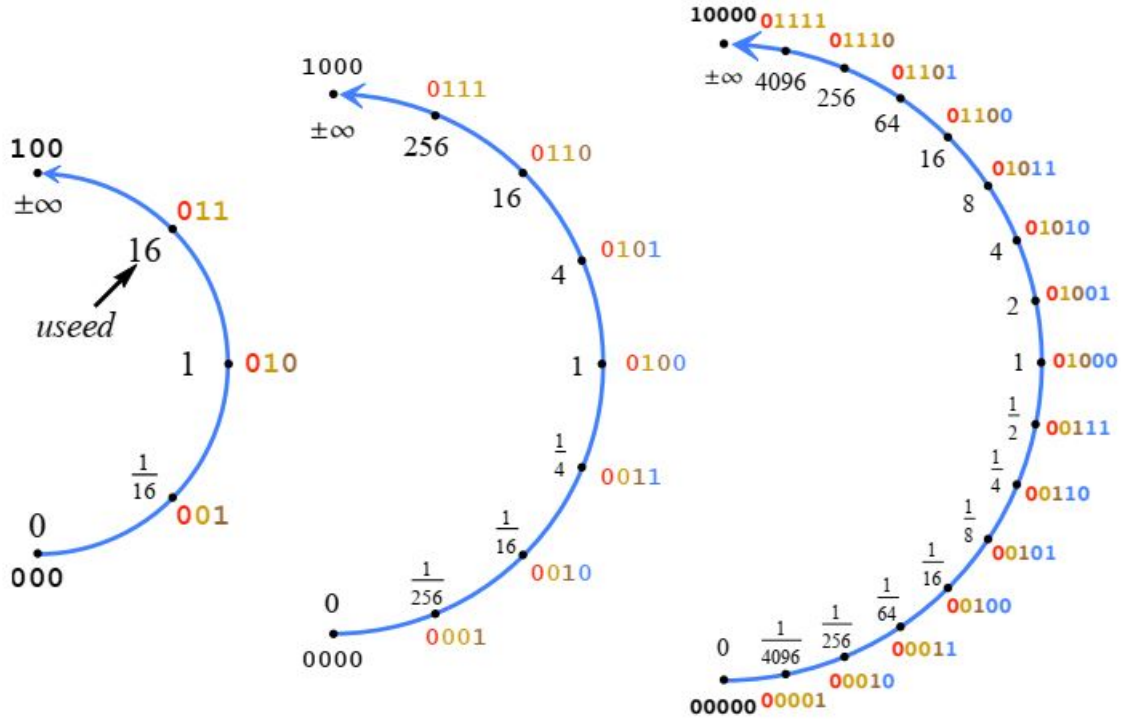
O valor atribuído a cada um dos dois valores intermediários deve ser feito deixando que o valor *maxpos* seja o maior valor positivo e o *minpos* seja o menor valor positivo no círculo definido com um bit. Na Figura 6, *maxpos* é o valor *useed* e o *minpos* é 1/*useed*.

Portanto, para fazer interpolação entre os valores, se deve:

- Entre o valor *maxpos* e $\pm\infty$, o novo valor é *maxpos* x *useed*; e entre 0 e *minpos*, o novo valor é *minpos/useed* (novo bit de regime).
- Entre os valores existentes x = 2^m e y = 2^n onde m e n diferem por mais do que 1, o novo valor é sua média geométrica, $\sqrt{xy}$ = $2^{(m+n)/2}$ (novo bit de expoente).
- Caso contrário, o novo valor é a equidistante entre os valores x e y existentes de maneira adjacente, que representa a média aritmética (x + y)/2 (novo bit de fração).

Como exemplo de interpolação, a Figura 7 mostra a construção de um Posit de 3 bits para 5 bits com *ES* = 2, então *useed* = $2^{2^2} = 16$ (bits de expoente em azul):

Figura 7: Construção de Posit com dois bits de expoente



Fonte: Gustafson e Yonemoto (2017)

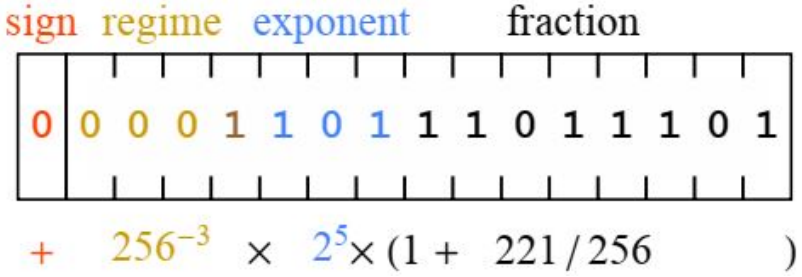
Se mais um bit fosse adicionado na Figura 7 para fazer Posit de 6 bits, deveria ser representando a faixa de valores entre 1/16 e 16, acrescentando bits de fração, mas não bits de expoente.

Para montar a equação que representa o valor na representação por um Posit *p*, a palavra binária de bits irá variar de -2^{n-1} a $2^{n-1}-1$. Sendo *k* o inteiro representado pelos bits do regime, e caso exista, e seja o inteiro sem sinal representado pelos bits de expoente. Se o conjunto de bits de fração for $f_1 f_2 ... f_{fs}$, possivelmente não existente, *f* será o valor representado por $1, f_1 f_2 ... f_{fs}$. Então o valor do Posit *p* poderá ser obtido conforme ilustrado na equação 5.

$$valor = \begin{cases} 0, & \text{se } p = 0, \\ \infty, & \text{se } p = -2^{n-1}, \\ sinal(p) \times useed^k \times 2^e \times f, & \text{todos os outros casos.} \end{cases} \quad (5)$$

Os bits de regime e expoente, são operados pela a função como um *float* padrão, pois em conjunto, definem a escala de potência de 2 da fração onde cada incremento é usado em um deslocamento de 2^{ES} bits. Um exemplo de decodificação de um Posit, inclusive com um valor não convencional para *ES*, é mostrado na Figura 8.

Figura 8: Exemplo de decodificação Posit



Fonte: Gustafson e Yonemoto (2017)

O bit de sinal 0 significa que o valor é positivo. Os bits de regime 0001 iniciam com três zeros, o que significa que *k* é -3. Portanto, a contribuição do fator de escala do regime é 256^{-3}. Os bits 101 de expoente representam 5 como um inteiro binário sem sinal e contribuem com outro fator de escala de 2^5. Finalmente, os bits da fração 11011101 representam 221 como um inteiro binário sem sinal de modo que a fração é 1 + 221/256. A expressão mostrada abaixo da palavra de bits na Figura 8 representa então $477/134217728 \approx 3{,}55393 \times 10^{-6}$.

Os Posits apresentam algumas vantagens quando comparado ao *float* da norma IEEE-754. A primeira vantagem, é que são bastante econômicos, pois não há redundância nas palavras de bits, e possui apenas uma única representação para o infinito. Logo, os demais padrões de bits são números reais diferentes de zero, distintos e válidos. Pois não há representações de NaN, já que o infinito já o substitui para todos os casos de exceção.

Além disso, apresenta uma matemática correta, existindo uma única representação para o zero absoluto. Em relação ao *overflow* e *underflow* acontecem de forma gradual, assim o número de dígitos de fração não são fixos para o formato. O que realmente acontece é que uma maior magnitude de expoente reduz automaticamente o número de dígitos de fração, o que faz com que haja uma evolução gradual para o zero ou infinito.

2.2 PACoGen - POSIT ARITHMETIC CORE GENERATOR

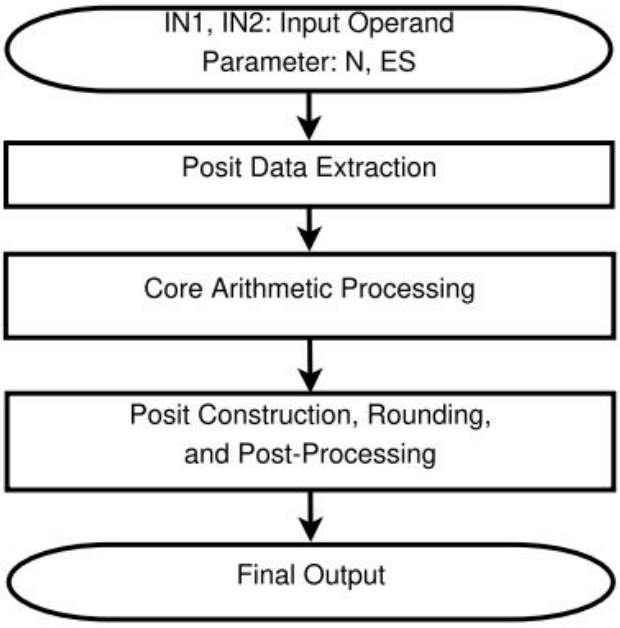
Com a aritmética Posit, campos de bits e respectivos significados apresentados, podemos agora visualizar uma de suas implementações mais conhecidas.

Como mostrado mais acima pelo desenvolvimento de Gustafson, os Posits oferecem muitos benefícios sobre o padrão do IEEE, incluindo melhor faixa dinâmica e precisão sobre o mesmo campo de bits, bem como cálculos aritméticos mais precisos e exatos. Esses bons resultados trouxeram consigo um profundo interesse na área de desenvolvimento de *software*. Assim, surgiram diversas ferramentas desenvolvidas para tratar o Unum usando várias linguagens como Julia, Python, C, C++, MATLAB, R, entre outras.

Entretanto, um trabalho muito limitado tem sido direcionado para soluções de *hardware*. Então, no trabalho de Jaiswal e So (2019), foi desenvolvida uma solução de código fonte aberto, que é definido como um núcleo gerador de *hardware* para a aritmética Posit (PACoGen), sendo recentemente implementada para sistemas de números Posit sob as características de Unum. O foco da solução está na aritmética básica de Posit como adição/subtração, multiplicação e divisão aritmética.

Nesta seção, é apresentada uma descrição detalhada dos geradores HDL para a aritmética Posit com foco nos somadores e multiplicadores, que foram os módulos utilizados para realizar as operações nos *designs* propostos. Estes são parametrizados para qualquer tamanho da palavra Posit (N) e tamanho do expoente (ES). O cálculo de fluxo básico para estas operações aritméticas é apresentado na Figura 9. Ela começa com a extração de dados Posit, então realiza o processamento pelo núcleo aritmético relacionado a determinada aritmética definida pelas constante, seguido pela construção Posit, arredondamento e processamento final.

Figura 9: Fluxo básico da aritmética Posit



Fonte: Jaiswal e So (2019)

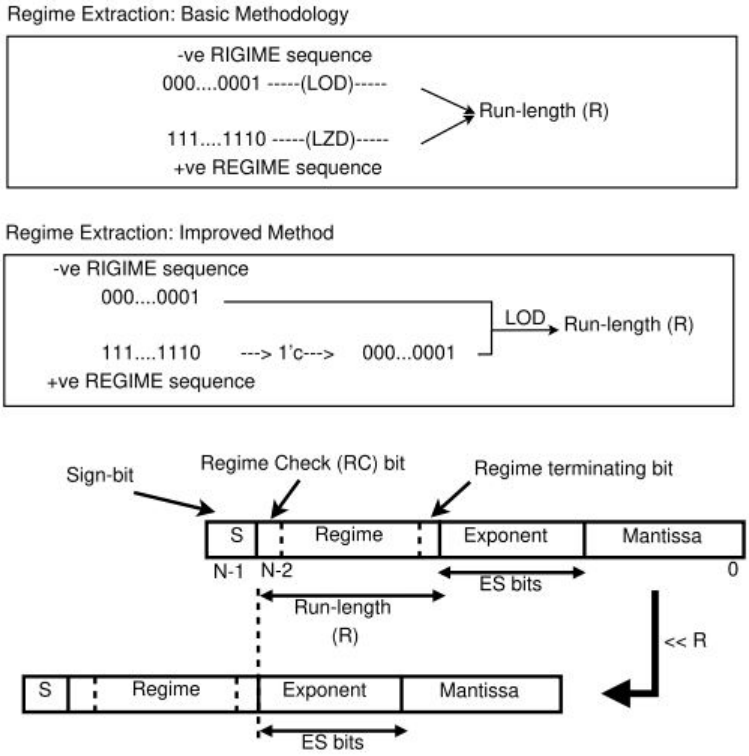
Os detalhes sobre cada uma dessas etapas são discutidos mais a frente. O gerador é implementado em Verilog HDL e parametrizado tomando o tamanho de palavra Posit (*N*) e o tamanho do expoente (*ES*) como parâmetros, gerando o *hardware* desejado. Uma vez que a largura máxima do regime na palavra pode ser (*N*-1) bits, os bits *RS* podem armazenar seu valor numérico absoluto máximo.

O algoritmo de extração de dados Posit verifica primeiro os operandos de entrada para casos especiais (como zero e infinito). Todos os bits com 0 levam a um zero na representação posit, entretanto, todos os bits com 0, exceto pelo sinal de nível lógico 1 leva à representação de infinito em Posit.

Os bits mais significativos dos operandos Posit fornecem os respectivos bits de sinal. Para operandos negativos (se o bit de sinal for 1), é necessário uma conversão em complemento 2, sem contar o bit de sinal, para produzir os operandos transformados.

A ideia de extração de dados Posit é mostrada na Figura 10. O bit mais significativo do operando, já transformado por complemento de 2, atua como o bit de verificação de regime (*RC*), de maneira que, seja 0 para palavra de regime negativa e 1 para palavra de regime positiva. Então, também é necessário descobrir a posição do bit do último bit de regime. Como mostrado na Figura 10, a idéia principal é usar um *Leading One Detector* (LOD, em português, detector de bit 1) para detectar o 1 em uma palavra de 0s (0...01) e usar um *Leading Zero Detector* (LZD, em português, detector de bit 0) para detectar o 0 em uma palavra de 1s (1...10).

Figura 10: Extração de dados Posit



Fonte: Jaiswal e So (2019)

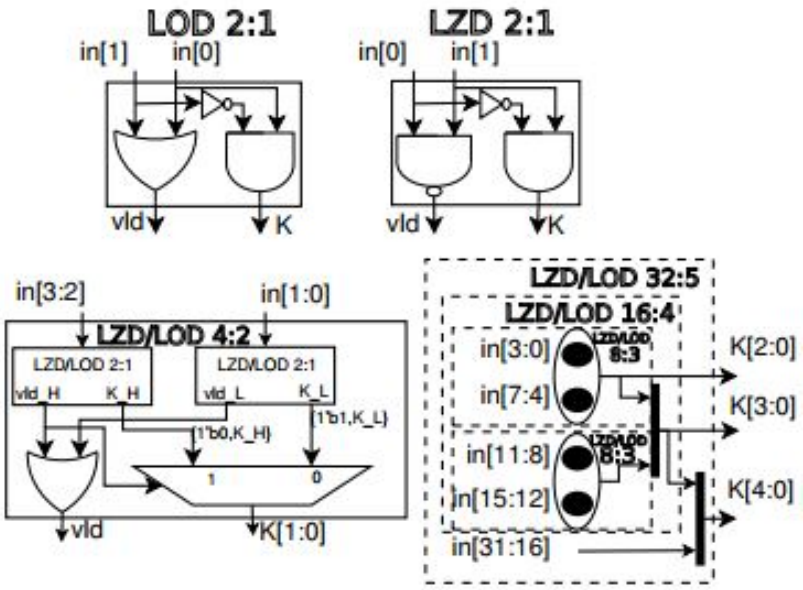
Entretanto, se uma palavra com 1...10 for complementada (complemento de 1), então o uso de um único LOD é suficiente para encontrar a palavra de regime que termina no último bit. Isto é mostrado na Figura 10 como método melhorado de extração de regime, e este processo fornece o comprimento de execução da palavra de regime (*R*). O valor efetivo do regime absoluto seria esse mesmo *R* para uma palavra de 0s, e *R*-1 para uma palavra de 1s.

Como mostrado na Figura 11, é gerado um circuito lógico que pode ser construído na forma hierárquica para os módulos LOD e LZD em 2:1. Essa arquitetura mais completa é feita com base na construção de um LOD/LZD de tamanho maior usando LOD/LZD 2:1.

Após encontrar o comprimento do regime, a palavra é deslocada para fora do operando por deslocamento dinâmico à esquerda por *R*. Este processo permite a combinação de expoente e mantissa nos bits mais significativos. Também com o comprimento do expoente definido (bits *ES*) e sua posição conhecida nesta fase do cálculo, o expoente e a mantissa (a parte restante) são facilmente extraídos. O comprimento máximo da palavra de bit da mantissa é de *N*-*ES*-2.

Esse deslocador dinâmico baseia-se no deslocador *barrel* e é parametrizado com tamanho de palavra *N* e quantidade de deslocamentos *S*.

Figura 11: Circuitos lógicos para os módulos LOD e LZD



Fonte: Jaiswal e So (2018)

O circuito requer um multiplexador de *N* bits em módulos menores 2:1 para cada bit de *S*. Assim, aqui seria necessário números *RS* em alguns multiplexadores 2:1 para cada (*N*-1) bit de tamanho. Assim, a unidade de extração Posit extrai os bits de sinal, os valores efetivos do regime absoluto, os valor dos expoentes e mantissa dos respectivos operandos Posit (*XIN1,S1,RC1,R1,E1,M1 e XIN2,S2,RC2,R2,E2,M2*) e passa para a próxima unidade da Figura 9.

O núcleo aritmético de processamento é uma unidade que funciona principalmente análoga a uma unidade aritmética de ponto flutuante padrão. Portanto, a aritmética central da adição/subtração de mantissa, e o cálculo do expoente resultante e do valor do regime são processados. A operação aritmética efetiva (adição/subtração) entre ambos os operandos de mantissa é encontrada por uma operação lógica **XOR** entre ambos os bits de sinal. Para realizar a adição/subtração de mantissa, o operando maior e o operando menor precisam ser descobertos. Uma comparação direta entre os operandos já permite obter essa informação. Ela requer um circuito ”maior ou igual” de (*N*-1) bits para efetuar a comparação.

Logo, nesta comparação, são computados os componentes grandes e pequenos dos bits de sinal, checagem do bit de regime, regime, expoente e mantissa. Todos requerem, cada um deles, um MUX 2:1 de comprimento da palavra de bits.

Para realizar a operação na mantissa, é necessário um alinhamento decimal da mantissa dos operandos.

Para conseguir isto, é necessário que a mantissa menor seja deslocada dinamicamente para a direita pela diferença do valor total efetivo de um grande expoente e um pequeno expoente (*Ediff*). Esse valor é calculado pela combinação da diferença dos valores efetivos do regime (considerando seus sinais, depois que foram deslocados à esquerda pelos bits *ES*), e diferenças exponenciais. A pequena mantissa é deslocada por *Ediff* por um deslocamento dinâmico à direita. É um deslocamento dinâmico parametrizado projetado de maneira semelhante ao deslocamento *barrel* para a esquerda, feito anteriormente, mudando apenas a direção para a direita.

A adição/subtração de mantissa menor deslocada e a mantissa maior é efetuada usando um somador/subtrator de (*N*-*ES*-2) bits. Para o caso de operação de adição, caso haja *overflow* de mantissa, detectado por seu bit mais significativo, a mantissa deve ser deslocada por 1 bit para a esquerda, o que requer um MUX 2:1 de (*N*-*ES*-2) bits. O valor final do expoente aumenta posteriormente, em decorrência do *overflow* da mantissa.

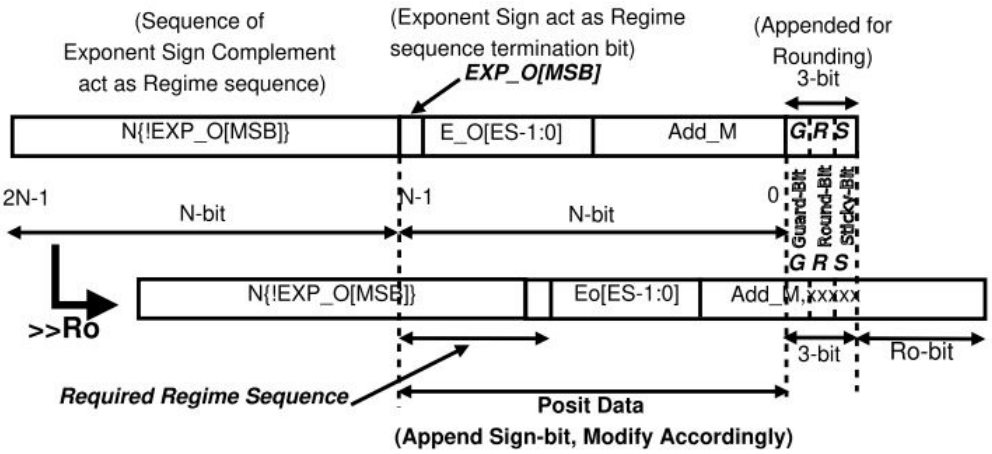
Para uma mantissa eficaz na operação de subtração, deve-se evitar a perda de bits na parte mais significativa da mantissa. Nesse caso, o mais adequado seria uma normalização da mantissa. Isto requer uma operação de detecção de 1s (LOD) e depois um deslocamento dinâmico para a esquerda. Após o processamento descrito, os cálculos do regime final e expoente são realizados .

Nesta parte final, o bit de sinal, o regime, o expoente e mantissa são por fim processados, e passam para a última fase do fluxo.

A unidade de composição de dados e andamento produz os ajustes finais nos cálculos realizados. Esta unidade faz a composição, arredondamento e processamento final dos Posits. A estratégia para a configuração final do Posit é feita utilizando uma palavra binária de comprimento $2N+3$ bits. Assim, os bits mais significativos da palavra de N bits são um complemento do bit de sinal na saída do expoente total. Ou seja, o objetivo é criar a palavra desejada do regime. Uma vez que o regime consiste em uma sequência de 1s para expoente positivo e uma seqüência de 0s para a sequência negativa, uma palavra de bits de sinal complementados do atual expoente, atuaria como a sequência de regime desejada. Além disso, como o bit final de regime é o oposto dos demais bits de regime, portanto, o mesmo bit de sinal atuaria como o último bit de regime.

Os próximos *N*-1 bits são os *ES* bits que compõem a saída do expoente, seguido pela mantissa. Finalmente, 3 bits zeros são anexados ao lado da parte menos significativa, e são chamados de *Guard*, *Round* e *Sticky* para fins de arredondamento. Todos esses bits podem ser observados na Figura 12.

Figura 12: Construção do Posit final



Fonte: Jaiswal e So (2019)

O arredondamento é realizado gerando um bit especial pela combinação da posição do bit de precisão da mantissa (*L*), o próximo bit como *Guard*, o próximo bit como *Round* e a operação **OR** de todos os bits restantes como *Sticky*. Esse bit especial é computado por meio da expressão 6.

$$bit_{especial} = G.(R + S) + L.G.(!(R + S)) \tag{6}$$

Esse bit é adicionado no bit de precisão da mantissa. Isto admite a combinação necessária da palavra de regime, expoente e mantissa arredondada. Deve-se notar que o arredondamento da adição é necessária somente quando o valor absoluto de regime na saída for menor que a largura máxima da mantissa do formato Posit (*N-ES*-3). Além disso, para saída com bit de sinal 1, um complemento de dois é tomado como por exigência de formato Posit, que é então anexado na parte mais significativa. A saída final de Posit é produzida após verificação excepcional para zero e infinito. Para o caso de zero, todos os bits serão 0, e para o infinito também, exceto o bit de sinal, caso contrário, o valor de posit computado será finalmente fornecido como saída final.

Todos os procedimentos acima são parametrizados para *N* e *ES*. A discussão de processamento acima (Figura 9), relacionada à extração de dados Posit, construção de dados, arredondamento e processamento final é bastante semelhante para o multiplicador Posit, exceto para o cálculo aritmético central.

Semelhante à aritmética do somador Posit, a aritmética de multiplicação também consiste em três seções principais de processamento: extração de posit, núcleo aritmético e construção de posit. Os dados Posit são extraídos da mesma maneira, seguindo então para a aritmética de multiplicação.

O cálculo para o bit de sinal requer uma operação **XOR** entre os bits de sinal de entrada. Para a multiplicação de mantissa é necessário um multiplicador inteiro (*N-ES*-2)X(*N-ES*-2), cuja saída da parte mais significativa é verificada caso seja necessário indicar *overflow*. Ainda na multiplicação da mantissa, há um deslocamento por 1 bit para uma normalização adequada. Para o cálculo do expoente, inicialmente os valores reais do regime são computados usando os respectivos bits de verificação de regime e valor absoluto do regime. Estes valores de regime são combinados com o respectivo expoente para fornecer respectivos valores efetivos de cada operando, que são então adicionados junto com o possível *overflow* da mantissa, para fornecer o valor total do expoente de saída. Usando o valor total do expoente de saída.

Após o processamento no núcleo aritmético de multiplicação, a construção, arredondamento e processamento final é realizado de forma similar ao que foi implementado para o somador.

2.3 APRENDIZAGEM DE MÁQUINA

Aprender como um processo genérico é adquirir novos, ou modificar os comportamentos, valores, conhecimentos, habilidades ou preferências existentes. Behaviorismo, cognitivismo, construtivismo, experiencialismo e aprendizagem social definem a teoria da aprendizagem pessoal, ou seja, como os seres humanos aprendem (Alzubi et al., 2018).

De forma semelhante, as máquinas também aprendem com experiências, por meio de dados de entrada. Logo, de maneira mais simples, aprendizagem de máquina é uma categoria da área de inteligência artificial, que permite que os computadores aprendam por conta própria sobre as mais diversas áreas do conhecimento. Em fins práticos, os computadores se tornam capazes de tomar decisões a fim de melhorar as ações e assim atingir uma elevada razão de acerto, no qual o as ações corretas são o objetivo final.

Aprendizagem de máquina é uma área que tem crescido rapidamente nos últimos anos, principalmente no contexto da análise e computação de dados que normalmente permite que as aplicações funcionem de forma inteligente. O aprendizado de máquina normalmente fornece sistemas com a capacidade de aprender e melhorar a experiência automaticamente sem ser especificamente programado e é geralmente referido como a mais popular tecnologia mais recente na quarta revolução industrial (Sarker, 2021).

Portanto, é um dos temas com mais interesse não só da comunidade de ciências da computação, mas também de toda a área de tecnologia, que deseja otimizar seus processos, melhorando as ações do próprio sistema que aprende a operar da melhor forma.

Algumas aplicações de aprendizagem de máquina que são bastante utilizadas serão explicadas nesta seção. Mais a frente, essas aplicações serão os módulos que farão parte das comparações nas métricas de *hardware*. Essas aplicações serão multiplicação de matrizes, função sigmoide e redes neurais.

2.3.1 Multiplicação de Matrizes

Matrizes são uma forma de organizar informações numéricas em uma tabela retangular formada por linhas e colunas. Assim, esse formato em tabela facilita a execução de vários cálculos simultâneos com as informações existentes, que embora não sejam aplicações diretas do aprendizado de máquina, são necessários em diversos cálculos desta área para auxiliar a execução das operações realizadas, além de também serem alvo de estudo da aritmética Posit, tal como feito por Chen et al. (2018).

Por definição, as matrizes tem o formato *m* x *n* (*m, n* $\in$ N*), onde *m* é o número de linhas e *n* o número de colunas. Logo, seja a matriz genérica A_{mxn}, possui *m* linhas e *n* colunas. Para representar os elementos da matriz *A*, podemos usar a notação a_{ij}, em que *i* representa o índice da linha e *j* representa o índice da coluna para o refeciado elemento, como na expressão 7.

Para realizar a operação de multiplicação entre matrizes, devemos ter em mente dois aspectos importantes.

Primeiro, que a multiplicação de duas matrizes *A* e *B* pode ter resultados diferentes dependendo da ordem utilizada, ou seja, $AB \neq BA$. Também é importante verificar se o *n* da primeira matriz é igual ao *m* da segunda matriz, ou seja, o número de colunas da matriz da esquerda é igual ao número de linhas da matriz da direita, pois somente assim a multiplicação poderá ser realizada.

Ainda na expressão 7, vemos a representação de uma multiplicação de matrizes genérica, onde a regra de multiplicação se faz válida, pois *m* é o número de colunas de *A*, ao passo que também é o número de linhas de *B*, de modo que o resultado será uma matriz de ordem *p*x*q*.

$$A_{pxm}B_{mxq} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \dots & \vdots \\ a_{p1} & a_{p2} & \dots & a_{pm} \end{vmatrix} \text{x} \begin{vmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \dots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mq} \end{vmatrix} = \begin{vmatrix} c_{11} & c_{12} & \dots & c_{1q} \\ c_{21} & c_{22} & \dots & c_{2q} \\ \vdots & \vdots & \dots & \vdots \\ c_{p1} & c_{p2} & \dots & c_{pq} \end{vmatrix} = C_{pxq} \quad (7)$$

Ao se conhecer a ordem das matrizes que se vai multiplicar, por exemplo, já é também conhecido a ordem do resultado, o que facilita na geração de *hardware* que seja adequado. Quando as matrizes são quadradas, ou seja, número de linhas igual ao número de colunas, *m = n*, a multiplicação sempre é válida. Portanto, esse foi o tipo de matriz implementado neste trabalho. Ainda é possível realizar a multiplicação entre matrizes não quadradas, entretanto, o resultado deve ser analisado com cuidado, visto que cada posição da matriz resultante deve possuir um significado coerente de acordo com a aplicação.

Na expressão 8, vemos a multiplicação de matrizes *AB*, de ordem 2x2, gerando uma terceira matriz *C*, também de ordem 2x2.

$$A_{2x2}B_{2x2} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \text{x} \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} = \begin{vmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{vmatrix} = C_{2x2} \tag{8}$$

Portanto, dado que uma multiplicação matricial é decomposta em um conjunto de produtos, se torna muito útil em cálculos de aprendizagem de máquina, principalmente ao se trabalhar com grandes quantidades de dados. Além disso, como um dos exemplos de Uguen et al. (2019), essa é uma das aplicações que vem sendo investigada, a fim de avaliar a eficiência de se utilizar a aritmética Posit.

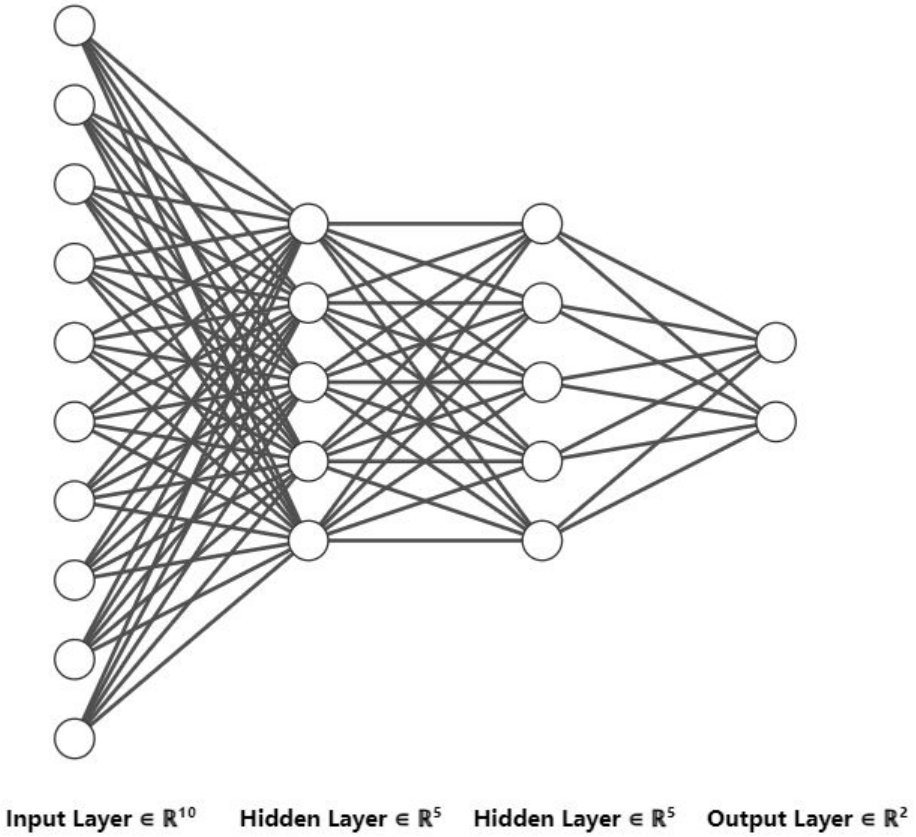
2.3.2 Redes Neurais

As redes neurais são apenas um dos métodos de escolha para a construção de algoritmos de aprendizagem de máquina. O grande entusiasmo do momento para esta área é investigar tarefas numéricas, tais como a resolução de equações diferenciais parciais de alta dimensão. O sucesso atual é proveniente da solução de vários problemas desafiadores de aprendizagem como jogos, reconhecimento facial (inclusive, detecção do uso de máscaras) e carros autônomos. Estas redes possuem a capacidade de gerar aproximações de uma função desconhecida a partir de observações de dados, de maneira bastante precisa (DeVore et al., 2021).

O uso de redes neurais é um método de escolha para construir algoritmos numéricos para aprendizagem de máquina e inteligência artificial de modo geral.

Na Figura 13, podemos ver a representação gráfica de uma rede neural com quatro camadas. A primeira camada, com dez nós, é por onde os dados entram no sistema. A segunda e a terceira camada são chamadas ocultas e estão com cinco nós cada. A última camada é a de saída, que possui dois nós, por onde a saída da rede é apresentada.

Figura 13: Exemplo de rede neural com quatro camadas



Fonte: Autoria Própria

As linhas entre os nós são chamadas de pesos e são atualizados conforme a rede recebe os dados e treina, de maneira que o valor dos pesos são atualizados para os próximos cálculos, fazendo com que a rede consiga aprender com os resultados já obtidos.

Embora as redes neurais já existam há mais de 70 anos, é consideravelmente recente a sua alta popularidade, pois elas alcançaram um desempenho bastante elevado em uma variedade impressionante de aplicações. Exemplos disso são a visão computacional, empregada, por exemplo, em carros autônomos, o processamento de linguagem natural (Johnson et al., 2016), usado no Google Tradutor, ou reconhecimento de dircurso (Amodei et al., 2015), entre muitos outros exemplos.

Para algoritmos tão poderosos, quanto os que representam essas redes neurais, está o entendimento de como usá-las em forma de aproximação em relação a outros métodos numéricos clássicos. Portanto, é um fato que a maioria das aplicações de redes neurais são construídas sobre alguma forma de aproximação de funções não lineares para obter resultados que até então tem sido muito satisfatórios.

Isto inclui não apenas probabilidade, estatística e cálculo numérico, mas também as novas descobertas nas próprias redes neurais para outros domínios de aplicação que se utilizem de métodos numéricos complexos para aumentar ainda mais a capacidade de chegar a soluções eficazes.

Um dos fatores preponderantes sobre o aprendizado das redes neurais são as funções de ativação, que são responsáveis diretas pela saída dos nós presentes nas camadas mostradas na Figura 13. Uma dessas funções é a função sigmoide, que será apresentada logo na próxima seção, e que é muito utilizada nos neurônios das redes neurais artificiais atualmente.

2.3.3 Função Sigmoide

A função sigmoide é uma das funções de ativação mais conhecidas no uso de redes neurais de avanço devido à sua não-linearidade e à simplicidade computacional de sua derivada. A equação que define a função sigmoide se encontra na expressão 9 e a sua derivada na expressão 10.

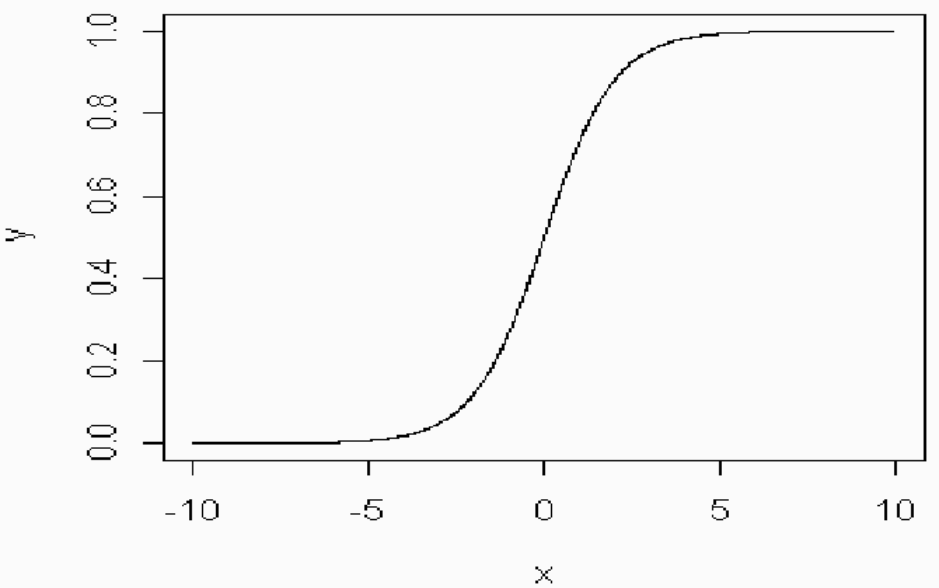
$$f(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

$$f'(x) = f(x) \times (1 - f(x)) \tag{10}$$

Com o gráfico que pode ser visto na Figura 14, podemos ver que a sigmoide é uma função real limitada que é definida para todos os valores de entrada reais e que tem uma derivada positiva em todos os pontos. Ela mostra um grau suficiente de suavidade e é, também, uma extensão adequada das não-linearidades que não variam de forma abrupta usadas nas redes neurais. A função satura para o zero, com valores inferiores a -5, assim como satura para 1, para valores superiores a 5.

Essa função tem se popularizado cada vez mais ao passo que cada vez mais aplicações vem utilizando-a como função de ativação. Porém, até certo tempo, a sigmoide era a mais utilizada em redes neurais artificiais por possuírem um significado biologico mais plausível, dado que os próprios neurônios são bioinspirados para que por meio de pulsos elétricos propagados um resultado final pudesse ser obtido. Então, como neurônios biológicos funcionam de foma binária, a função sigmoide é uma boa forma de modelar esse comportamento.

Figura 14: Gráfico da função sigmoide



Fonte: Guimarães (2022)

Na próxima seção, será mostrada a metodologia que foi seguida para produção dos resultados alcançados por este trabalho, bem como o ambiente de desenvolvimento e ferramentas utilizadas.

3 METODOLOGIA

A metodologia desse trabalho subdivide-se em três partes. Na primeira parte, é apresentada a tecnologia a que se propõem este trabalho, assim como as linguagens e técnicas de descrição de *hardware* utilizadas, bem como, uma possível aplicação real, mesmo que os resultados obtidos tenham sido adquiridos por simulação.

Na segunda parte, há o detalhamento do ambiente de desenvolvimento, as ferramentas internas e as métricas de avaliação do *hardware* em questão. Por fim, na última parte, será apresentado o simulador a nível de RTL (*Register Transfer Level*) que foi usado para realizar a análise dos níveis lógicos de entrada e saída e assim validar e comparar os resultados obtidos com os esperados.

3.1 FPGA - FIELD PROGRAMMABLE GATE ARRAY

Os circuitos integrados digitais podem ser utilizados como Array de Portas Programáveis de Campo (FPGA) que são fabricados a partir de módulos de silício, contendo desde portas simples até blocos lógicos e interconexões configuráveis entre estes módulos. Esses dispositivos podem executar grandes quantidades de tarefas, desde que sejam programadas para isso.

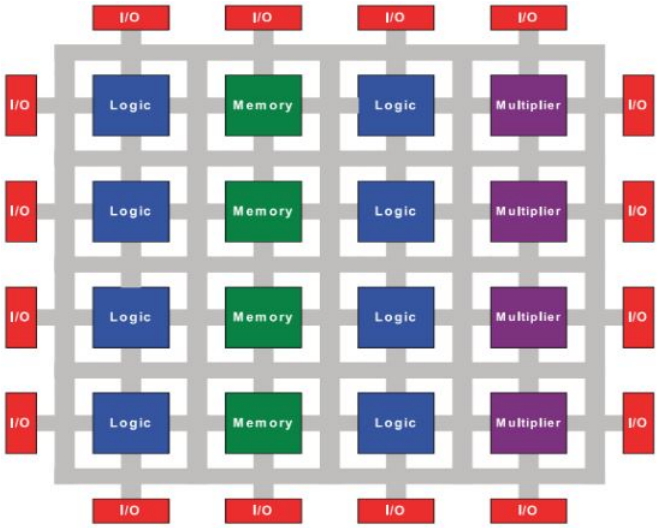
A fim de trazer aplicações para as FPGAs, projetistas podem utilizar diferentes formas de programá-la. Algumas FPGAs só podem ser programadas uma vez, e denominam-se de OTP (*One time programmable*), enquanto outras podem ser constantemente reprogramadas.

FPGAs são dispositivos com semicondutores usados para tabelas de busca programáveis com um número limitado de entradas para implementar tabelas da verdade para circuitos lógicos. Na forma de RAMs de bloco (BRAMs) e *flip-flops*, as FPGAs podem muitas vezes fornecer desde memórias convencionais até memórias on-chip de alta largura de banda (Maxfield, 2004). A arquitetura geral da estrutura das FPGAs consistindo de portas lógicas, RAMs integradas, multiplicadores e blocos de E/S como pode ser visto na Figura 15.

A FPGA utilizada nas simulações deste trabalho foi a Intel® 5CGXFC7C7F23C8 da família de dispositivos Cyclone V, que possui 56480 elementos lógicos e 268 pinos de entrada e saída. Para realizar a programação, as especificações foram escritas primeiramente em linguagem de descrição de *hardware* (HDL). Duas linguagens de descrição de *hardware* foram utilizadas: SystemVerilog e Verilog. Verilog e SystemVerilog são muito semelhantes e sua comparação é análoga a diferença entre C e C++.

SystemVerilog possui suporte a estruturas de dados como enum, *union*, *struct*, *string* e classes, enquanto Verilog não. Isso permite que SystemVerilog seja mais versátil, tanto no projeto de funcionamento quanto para o processo de verificação do sistema, por isso foi mais utilizada na produção de *testbenches*. Enquanto que Verilog foi utilizada para descrever os circuitos digitais.

Figura 15: Arquitetura Genérica de FPGA



Fonte: Yazdeen et al. (2021)

3.1.1 Aceleradores de Hardware

Cada vez mais os sistemas computacionais, de redes e armazenamento aumentaram sua capacidade de processamento drasticamente. Os sistemas de computação estão cada vez mais complexos, dinâmicos e heterogêneos, com conexões para outros sistemas que, por vezes, usam tecnologias completamente distintas. As FPGAs são uma alternativa como arquiteturas computacionais sofisticadas que permitem que tais aplicações sejam adaptativas em larga escala através de códigos, gerenciamento e execução.

Os cientistas que querem escrever aplicações paralelas de alto desempenho enfrentam agora um desafio com a arquitetura dos sistemas (Zebari e Yaseen, (2011);Zeebaree et al. (2020)). O desenvolvimento de aplicações que se utilizem desse conceito pode fazer com que o desempenho de diversos sistemas sejam muito maiores, já que irão apenas fornecer dados e receber resultados, sem se preocupar com o processamento, diminuindo a carga de trabalho. Portanto, este trabalho se propõe a mostrar como a aritmética Posit pode se utilizar desse conceito de *hardware* dedicado para acelerar ainda mais as aplicações existentes.

3.2 QUARTUS PRIME

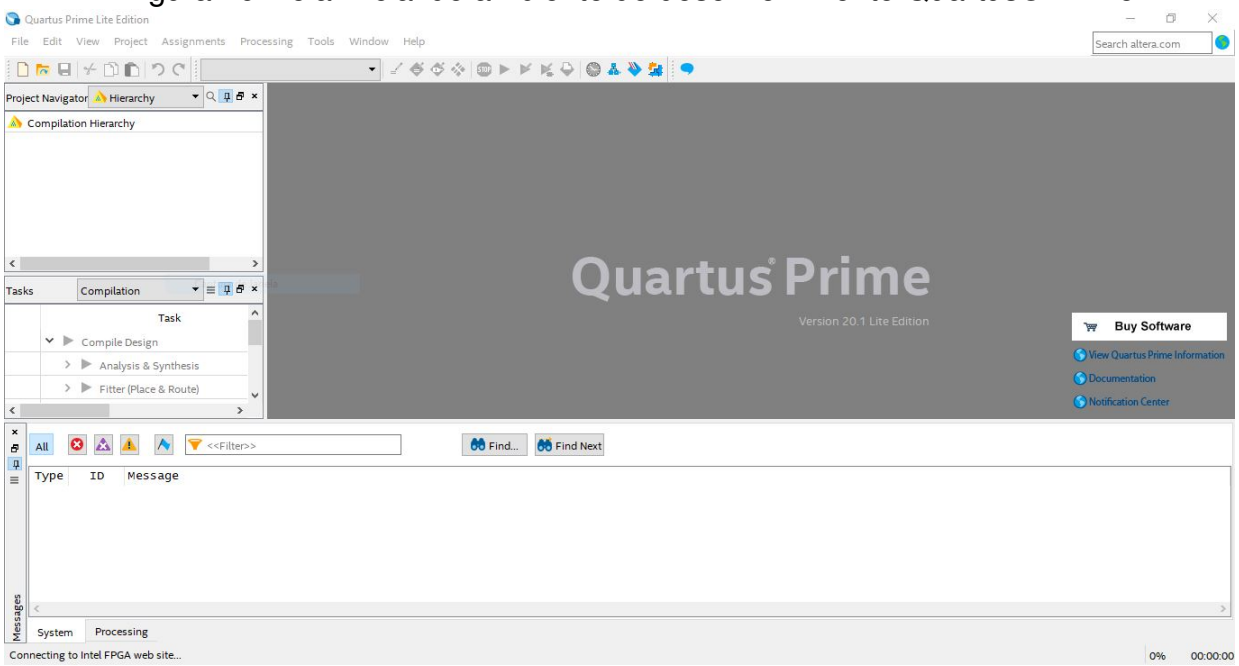
O *software* de *design* Intel® Quartus® Prime inclui diversas ferramentas para criar e desenvolver projetos destinados a FPGAs, sistema em um chip (SoC) e dispositivos lógicos programáveis complexos (CPLD), desde a entrada e síntese do *design* até a otimização, verificação e simulação. Capacidades dramaticamente aumentadas em dispositivos com elementos lógicos multimilionários estão fornecendo aos projetistas uma plataforma robusta para atender às oportunidades da próxima geração (Intel, 2018).

O Quartus® foi escolhido como ambiente de desenvolvimento porque foi uma das ferramentas utilizadas nas disciplinas pertencentes ao curso de graduação em engenharia elétrica. A escolha também foi norteada por algumas características importantes como: otimização sem impacto na etapa de organização de elementos lógicos ou o roteamento de outras partições, além de gerar relatórios detalhados de métricas de *hardware*, que foram necessários para as avaliações realizadas neste trabalho.

Na Figura 16, está a tela inicial do Quartus® Prime, no qual a versão utilizada neste projeto foi a 2020.1.

Ainda na Figura 16, podemos observar na lateral esquerda, a janela para navegar pelas abas de hierarquia e arquivos de módulos e a janela de *tasks* de operação. Já na parte inferior, há um painel de mensagens de avisos e erros, enquanto na parte central fica o código referente a descrição de *hardware* em edição no momento.

Figura 16: Tela inicial do ambiente de desenvolvimento Quartus® Prime



Fonte: Autoria Própria

Agora serão apresentadas algumas ferramentas internas que foram utilizadas em simulação para colher os dados relevantes que serão apresentados na seção de resultados. São elas: o *Timing Analyzer* para análise de tempo, o *Power Analyzer* para análise de potência consumida e a ferramenta *Fitter* para análise de área utilizada em chip. Além disso, para validar o funcionamento dos circuitos, pela observação dos sinais de entrada e saída, foi utilizada a ferramenta ModelSim.

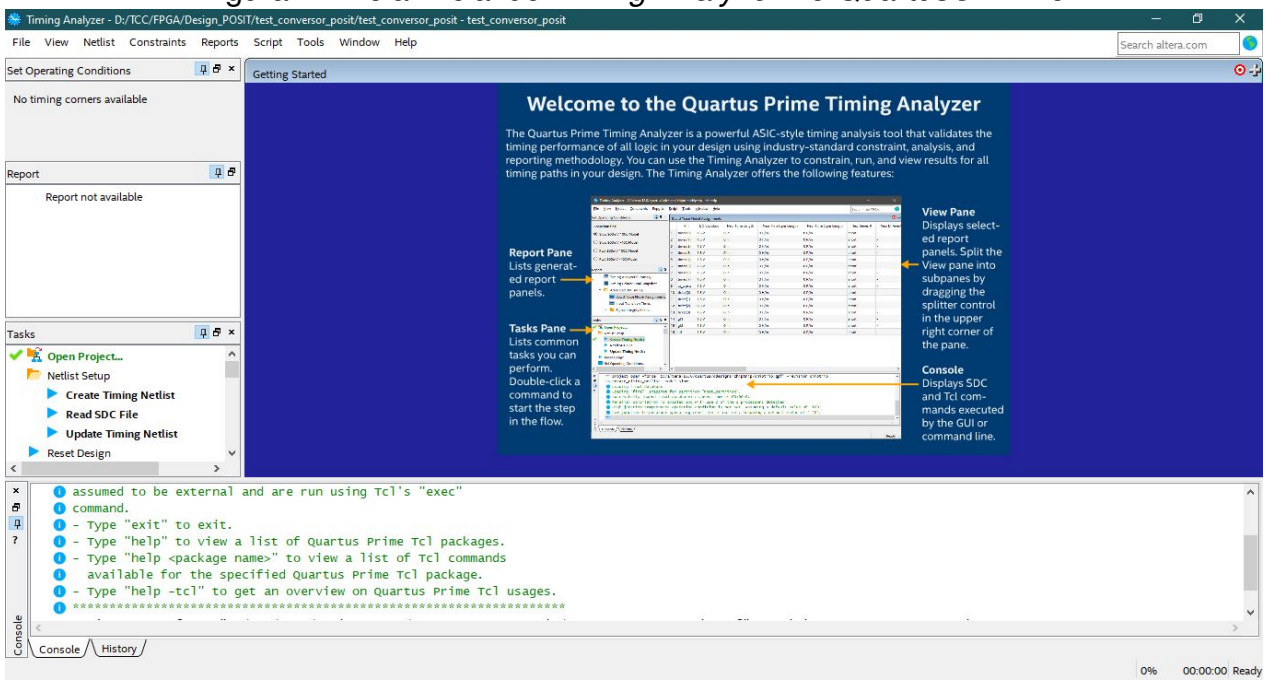
3.2.1 Avaliação de Tempo

Para realizar análise de tempo, foi utilizada a ferramenta *Timing Analyzer*, que fornece suporte nativo ao *Synopsys Design Constraint* (SDC) e permite criar, gerenciar e analisar restrições complexas de tempo e executar rapidamente a verificação de tempo avançada (Intel, 2021).

Entre essas restrições, estão configurações de tempo de *setup*, tempo de *hold*, *clocks* utilizados nos módulos e correlações entre a temperatura da FPGA e frequência máxima atingida durante a simulação.

A métrica de *hardware* analisada para o tempo nos *designs* desenvolvidos será a frequência máxima de funcionamento. Na Figura 17, podemos ver a tela inicial da ferramenta de análise de tempo.

Figura 17: Tela inicial do *Timing Analyzer* no Quartus® Prime



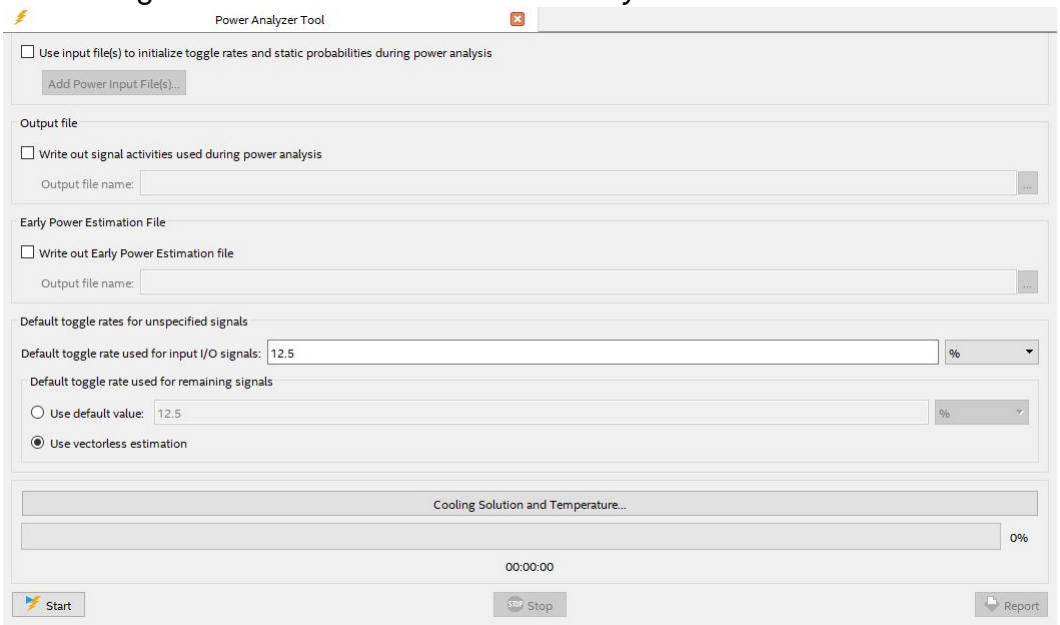
Fonte: Autoria Própria

3.2.2 Avaliação de Potência

Já para realizar análise de potência, foi utilizada a ferramenta *Power Analyzer*, que apresenta a nova Calculadora Térmica e de Potência (PTC) para alguns dispositivos Intel®, um estimador de potência inicial baseado em Excel, e a ferramenta analisadora de potência. Estas funções de análise de potência lhe dão a capacidade de estimar o consumo de energia desde o conceito de projeto inicial até a implementação do projeto (Intel, 2021).

Na Figura 18, podemos ver a tela inicial da ferramenta de análise de potência. Essa análise leva em consideração a taxa de alternância padrão utilizada para sinais de E/S, permitindo analisar e otimizar com precisão o consumo de energia dinâmica e estática, quantificadas em mW (miliwatts).

Figura 18: Tela inicial do *Power Analyzer* no Quartus® Prime



Fonte: Autoria Própria

3.2.3 Avaliação de Área Utilizada em Chip

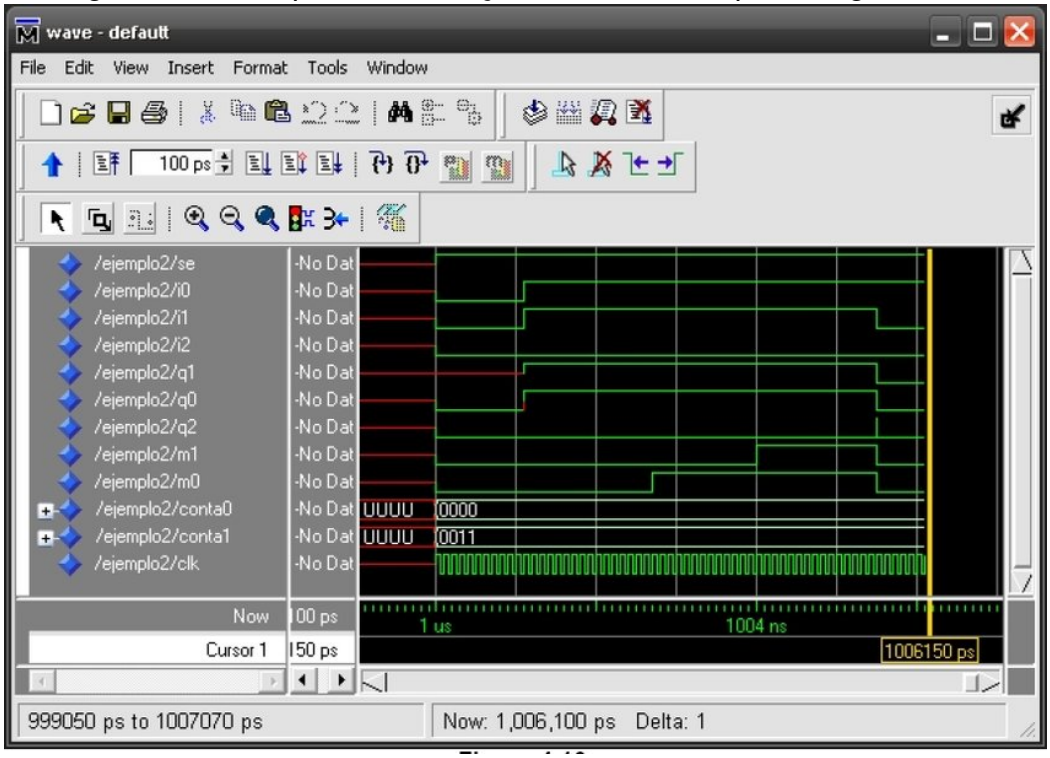
Finalmente, para realizar análise de área de chip, foi utilizada a ferramenta *Fitter*, que por sua vez, aplica as operações de *Place* e *Route* para posicionar e interligar os elementos lógicos da FPGA (unidade de medida usada para comparação neste trabalho), de modo a satisfazer as restrições indicadas no *design* e então proporcionar o melhor arranjo possível no *array* de portas existentes. A quantificação de elementos é adquirida no relatório geral após compilação do *design*.

3.3 ModelSim - SIMULADOR DE HARDWARE

O Modelsim é um simulador de HDL desenvolvido pela Mentor Graphics®, que suporta simulação das linguagens SystemVerilog, Verilog e VHDL, conseguindo simular o código a nível de RTL (*Register Transfer Level*) e *Gate Level*. Em nível de RTL é analisado o circuito a nível de comportamento dos registradores e em *Gate Level* é analisado a nível de *netlist* com inclusão de atrasos das portas lógicas e etc.

Na Figura 19, podemos ver a janela principal do ModelSim, em uma simulação de um *design* com os níveis lógicos dos sinais. Portanto, para validar os circuitos produzidos, essa ferramenta foi utilizada, identificando casos de erro e mal funcionamento e os adequando ao projeto de *hardware* modelado.

Figura 19: Exemplo de simulação no ModelSim por código em HDL



Fonte: Alonso et al. (2009)

4 RESULTADOS OBTIDOS

Foram produzidos três aplicações para realizar a comparação da aritmética Posit com a representação padrão IEEE-754, foram elas: multiplicação de matrizes, a função sigmoide e uma rede neural artificial.

Essas aplicações foram separadas para analisar diferentes aspectos tanto no que diz respeito a comparações na variação do expoente (*ES*) na aritmética POSIT; na comparação direta entre *floats* e Posits, ambos com o mesmo número de bits; e comparação entre a representação IEEE e Posits em diferentes quantidades de bits.

Os módulos utilizados como implementações para as operações aritméticas de soma e multiplicação foram o de Sinha (2021), para a norma IEEE-754, e Jaiswal (2019), para a aritmética Posit, sendo ambos soluções de código aberto na plataforma GitHub para uso e aprimoramento da comunidade. É importante ressaltar que o somador e multiplicador IEEE supracitados, foram desenvolvidos para a aplicação de precisão simples, entretanto, neste trabalho, os mesmos foram adaptados para as demais precisões.

O objetivo foi investigar a diferença entre representações numéricas e qual seu impacto no *hardware* produzido, por meio de métricas como tempo, potência e área de chip.

4.1 MULTIPLICAÇÃO DE MATRIZES

O primeiro *design* desenvolvido foi a multiplicação de matrizes. A operação foi organizada para ser realizada de acordo com o que foi explicado na seção 2.3.1. Para facilitar, tanto a produção quanto a depuração, as matrizes operadas são quadradas e de baixa ordem.

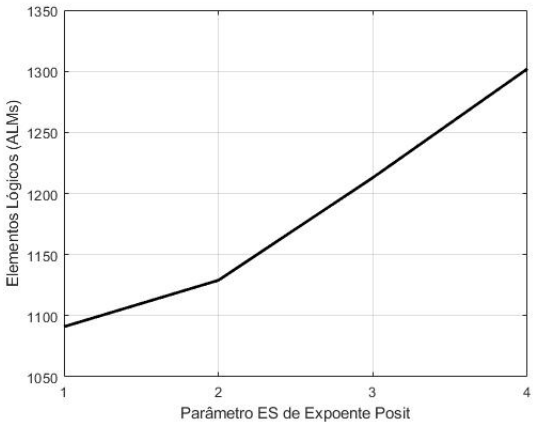
Para os casos de matrizes com elevada ordem possam ser operadas, as mesmas devem separadas para realização de apenas parte da multiplicação ou sejam aplicadas metodologias de resolução em que o IP elaborado seja usado como módulo auxiliar, por exemplo, ao se reduzir a ordem de uma matriz e só então a operação possa seguir para execução.

Então, para os resultados aqui obtidos, circuitos para matrizes 2x2 foram descritos e sintetizados com módulos de soma e multiplicação da aritmética Posit e IEEE-754, e se estendem para os demais casos que obedeçam as condições matemáticas exigidas, desde que a FPGA utilizada suporte o número de entradas e saídas requisitado.

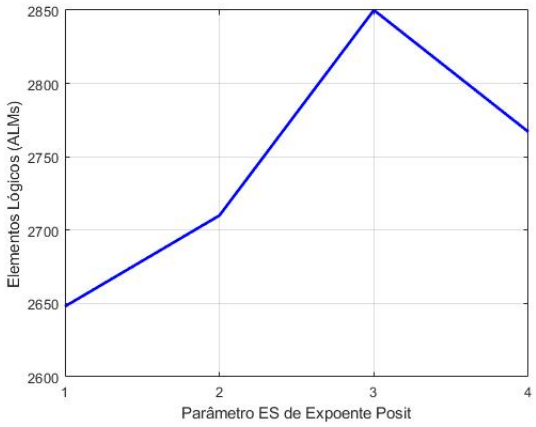
Caso a FPGA não possua recursos suficientes para suportar a aplicação, o recomendado é customizar um circuito integrado, que não é o alvo deste trabalho.

Inicialmente, avaliando a área de chip, que é uma métrica quantificada pelo número de elementos lógicos, na Figura 20, é fácil notar que, para 8 e 16 bits o menor custo com elementos lógicos se dá com *ES* = 1, enquanto que, para 32, a mesma característica ocorre com *ES* = 2.

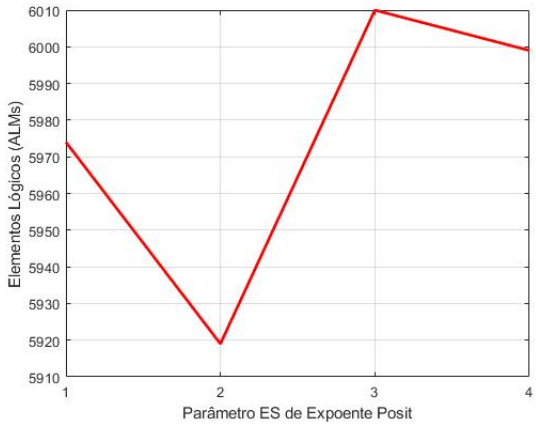
Figura 20: Elementos lógicos no *design* de multiplicação de matrizes em Posit com variação do parâmetro *ES* = 1, 2, 3, 4



(a) 8 bits



(b) 16 bits



(c) 32 bits

Fonte: Autoria Própria

Na Tabela 2, podemos comparar inicialmente *floats* de 8 bits com Posits de 8 bits. Com o melhor caso, de apenas um bit de expoente, mostrado na Figura 20(a). Então, temos que a quantidade de elementos lógicos do IEEE em relação ao Posit corresponde a 28,69% menor, além do *float* operar com uma frequência máxima de 2,52 vezes maior, em contraste com o valor do Posit. Em relação a potência, as duas representações se aproximaram bastante nos resultados obtidos.

Tabela 2: Comparação de IEEE-754 de 8 bits e Posit(8,1) na multiplicação de matrizes

Precisão	Área (ALMs)	Frequência Máx. (MHz)	Potência (mW)
IEEE	313	53,64	535,53
Posit	1091	21,26	542,39

Fonte: Autoria Própria

Já para 16 bits, fazendo uma comparação análoga pela Tabela 3, vemos que, aplicando as mesmas razões, o percentual de elementos lógicos, sobe para 30,74%, e a frequência máxima passa a ser 2,17 vezes maior. O resultado se distancia apenas para a potência, que passa a ser 90,09%, em relação ao Posit com a mesma quantidade de bits.

Tabela 3: Comparação de IEEE-754 de 16 bits e Posit(16,1) na multiplicação de matrizes

Precisão	Área (ALMs)	Frequência Máx. (MHz)	Potência (mW)
IEEE	814	35,94	559,69
Posit	2648	16,54	621,28

Fonte: Autoria Própria

Agora para 32 bits, com a Tabela 4, também de forma totalmente análoga, obtemos 34,14% para os elementos lógicos, 2,03 para a frequência máxima e 79,99% da potência dissipada, também sendo a única, para este caso de 32 bits, onde a razão se afasta da igualdade nas comparações de representação numérica.

Tabela 4: Comparação de IEEE-754 de 32 bits e Posit(32,2) na multiplicação de matrizes

Precisão	Área (ALMs)	Frequência Máx. (MHz)	Potência (mW)
IEEE	2021	24,75	661,27
Posit	5919	12,2	826,73

Fonte: Autoria Própria

Devido ao PaCoGen não conseguir sintetizar o circuito com *ES* = 0, este caso não foi incluso nas comparações. Esse valor de parâmetro poderia permitir resultados ainda mais otimistas em relação a aritmética Posit.

Também é possível fazer comparações com quantidades de bits diferentes, em que, o Posit possua uma menor quantidade, já que possuem uma maior precisão com consumo de *hardware* maior como visto para tamanhos de palavra binária iguais.

Portanto, comparando o IEEE de 16 bits com o Posit de 8 bits e *ES* = 1, vemos que a razão de elementos lógicos sobe para 74,61%, a razão de frequência máxima desce para 1,69, e assim vemos pela primeira vez a vantagem clara da aritmética Posit na razão de potências em 1.03. Agora, comparando o IEEE de 32 bits com o Posit de 16 bits e *ES* = 1, vemos que a razão de elementos lógicos sobe para 76,32%, a razão de frequência máxima desce para 1,5, enquanto a razão de potência fica em 1,06.

4.2 FUNÇÃO SIGMOIDE

O próximo *design* desenvolvido foi a implementação da função sigmoide. A operação foi organizada para ser realizada de acordo com o que foi explicado na seção 2.3.3. Da mesma maneira que foi feito para matrizes 2x2, foram utilizados os módulos de soma e multiplicação da aritmética Posit e IEEE-754.

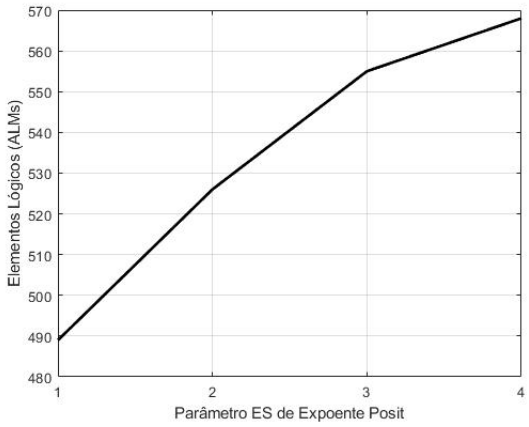
O módulo utilizado como *design* para a função sigmoide foi o de Dukare (2019), com código aberto na plataforma GitHub para uso e aprimoramento da comunidade. Esse módulo possui um modelo de alto nível, que está em outra representação. O modelo foi aproveitado para utilizar o somadores e multiplicadores já citados e realizar a comparação.

Avaliando novamente a área de chip, que é uma métrica quantificada pelo número de elementos lógicos, na Figura 21, é fácil notar que, para todas as quantidades de bits nesse *design* (8, 16 e 32 bits), o menor custo com elementos lógicos se dá com *ES* = 1.

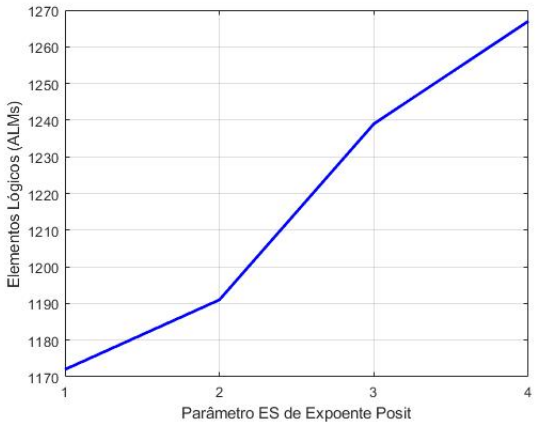
Diferentemente do *design* de multiplicação de matrizes, o *design* da função sigmoide segue com um número crescente de elementos lógicos, conforme o número de bits totais aumenta, sem oscilação.

Na Tabela 5, podemos comparar inicialmente *floats* de 8 bits com Posits de 8 bits. Com o melhor caso, de apenas um bit de expoente, mostrado na Figura 21(a). Então, temos que, a quantidade de elementos lógicos do IEEE em relação ao Posit corresponde a 23,31% menor, além do *float* operar com uma frequência máxima de 2,63 vezes maior, em contraste com o valor do Posit. Em relação a potência, a razão similar encontrada totalizou 97,81% menor.

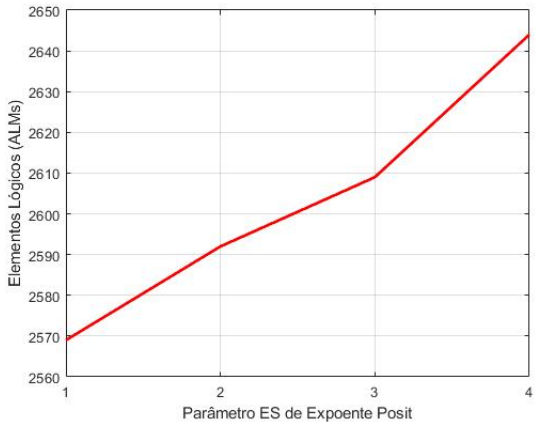
Figura 21: Elementos lógicos no *design* da função sigmoide em Posit com variação do parâmetro *ES* = 1, 2, 3, 4



(a) 8 bits



(b) 16 bits



(c) 32 bits

Fonte: Autoria Própria

Tabela 5: Comparação de IEEE-754 de 8 bits e Posit(8,1) na função sigmoide

Precisão	**Área (ALMs)**	**Frequência Máx. (MHz)**	**Potência (mW)**
IEEE	114	21,28	526,22
Posit	489	8,10	537,98

Fonte: Autoria Própria

Para 16 bits, a comparação é feita pela Tabela 6, em que, aplicando as mesmas razões, o percentual de elementos lógicos, sobe para 27,73%, e a frequência máxima passa a ser 2,39 vezes maior. O resultado se distancia apenas para a potência, que passa a ser 89,86%, em relação ao Posit com a mesma quantidade de bits.

Tabela 6: Comparação de IEEE-754 de 16 bits e Posit(16,1) na função sigmoide

Precisão	**Área (ALMs)**	**Frequência Máx. (MHz)**	**Potência (mW)**
IEEE	325	13,51	533,10
Posit	1172	5,66	593,26

Fonte: Autoria Própria

Finalmente, para 32 bits, com a Tabela 7, também de forma totalmente análoga as comparações anteriores, obtemos 32,81% para os elementos lógicos, 2,33 para a frequência máxima e 78,86% da potência dissipada, também sendo a única, para este caso de 32 bits, onde a razão se afasta da igualdade nas comparações de representação numérica.

Tabela 7: Comparação de IEEE-754 de 32 bits e Posit(32,2) na função sigmoide

Precisão	**Área (ALMs)**	**Frequência Máx. (MHz)**	**Potência (mW)**
IEEE	843	9,96	564,83
Posit	2569	4,26	716,21

Fonte: Autoria Própria

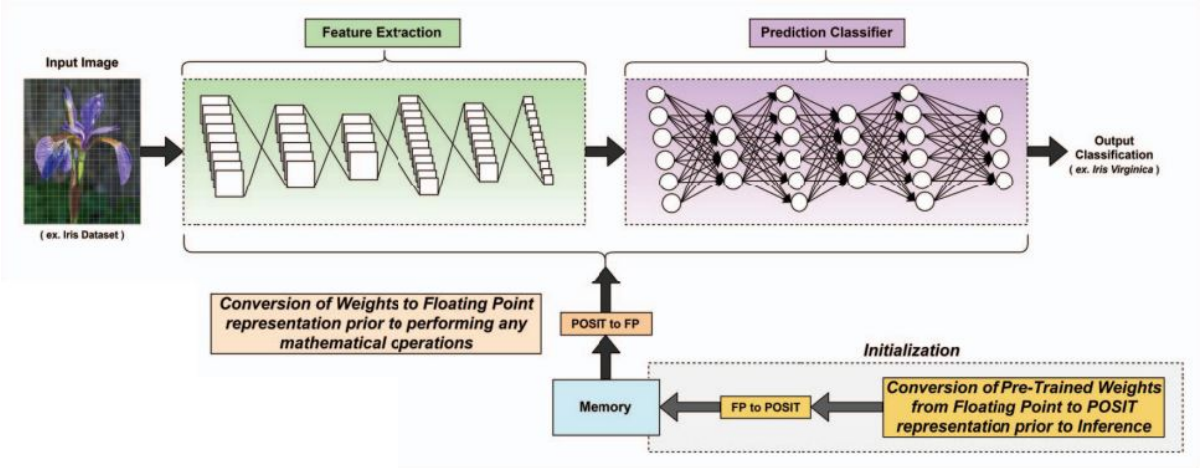
Também comparando com quantidades de bits diferentes, o IEEE de 16 bits com o Posit de 8 bits e $ES = 1$, vemos que a razão de elementos lógicos sobe para 66,46%, a razão de frequência máxima desce para 1,67, e na razão de potências 99,09%. Agora, comparando o IEEE de 32 bits com o Posit de 16 bits e $ES = 1$, vemos que a razão de elementos lógicos sobe para 71,93%, a razão de frequência máxima desce para 1,76, enquanto a razão de potência fica em 95,21%.

4.3 REDES NEURAIS

No último *design*, foi avaliada uma aplicação maior, mais robusta e em um maior nível de complexidade. Como dito anteriormente, aprendizagem de máquina é uma área que tem crescido bastante, e isso se deve muito ao poder de predição com o uso de redes neurais, que foram explicadas na seção 2.3.2. Para este caso, foram utilizados os conversores propostos por Fatemi Langroudi et al. (2018), já que a rede utilizada foi treinada em ponto fixo, se fez necessário converter os pesos das redes neurais de ponto fixo (1 bit de sinal, 3 bits de parte inteira e 12 bits de parte fracionária) para a aritmética Posit e IEEE-754.

O módulo utilizado como *design* para a rede neural trabalhada foi o de Vipin (2020), com código aberto, disponível também na plataforma GitHub. A representação que essa rede neural utiliza é ponto fixo, portanto utilizando os conversores citados, é possível realizar o armazenamento de valores na memória em uma outra representação (IEEE ou Posit), e para realizar os cálculos na rede os valores são convertidos de volta para ponto fixo ao serem lidos da memória. O diagrama da Figura 22 ilustra o processo explicado.

Figura 22: Visualização de alto nível da arquitetura de conversão entre diferentes representações de pesos em aplicação de rede neural



Fonte: Fatemi Langroudi et al. (2018)

A tarefa principal dessa rede neural é identificar digitos em imagens 28x28 *pixels*, que resulta em 784 entradas na primeira camada da rede. As camadas ocultas são 3, as duas primeiras possuem 30 neurônios cada e a mais próxima a saída possui 10 neurônios. A rede produz 10 saídas na sua última camada, que representam os dígitos de 0 a 9, que a rede tenta acertar de acordo com o treinamento realizado com os pesos. Por fim, os valores são comparados, como um apredizado supervisionado e calculada uma acurácia total da rede neural.

Como conjunto de dados utilizados, foi usado o popular conjunto de dados MNIST, para realizar a comparação desejada através de simulação e validação de *hardware*. O conjunto de dados MNIST para reconhecimento de dígitos manuscritos possui 30000 imagens para treino e 10000 imagens para testes. Os pesos e os bias para a rede foram encontrados em pré-treinamento a partir da implementação de *software* e utilizados para a implementação de *hardware* (Vipin, 2019).

Em relação a quantidade de bits, foi escolhido como representante Posit, 16 bits de palavra binária, sem bit de expoente. Essa escolha foi realizada considerando que o ponto fixo padrão da rede é de 16 bits, portanto valores menores levariam a acurácias muito baixas devido a grande perda de informação pela redução de casas decimais. Essa representação foi comparada com os 16, 32 e 64 bits da representação da norma IEEE-754.

Na Tabela 8, podemos comparar os *floats* de 16, 32 e 64 bits com um Posit base de 16 bits. Então, temos que, a quantidade de elementos lógicos do IEEE em relação ao Posit correspondem a 44%, 56,1% e 58,85%, para as respectivas quantidades de bits supracitadas. Já para a frequência máxima obtemos os valores de 1,57, 1,47 e 1,46 vezes maior, para os *floats*, em contraste com o valor do Posit. Em relação a potência, os valores encontrados foram 57,4%, 60,55% e 62,79%, também para os respectivos casos em relação ao Posit utilizado.

Tabela 8: Comparação de IEEE-754 com 16, 32 e 64 bits e Posit(16,0) na rede neural

Precisão	**Área (ALMs)**	**Freq. Máx. (MHz)**	**Potência (mW)**	**Acurácia (%)**
IEEE (16 bits)	12891	36,97	421,22	74
IEEE (32 bits)	16438	34,68	444,37	78
IEEE (64 bits)	17244	34,44	460,77	91
Posit(16,0)	29301	23,52	733,86	91

Fonte: Autoria Própria

Entretanto, a principal métrica levada em consideração, neste caso, é a acurácia, pois a aplicação para qual se destina a rede geralmente necessita de elevada taxa de acerta para justificar o uso de métodos de aprendizagem de máquina. Isso acontece porque, principalmente em aplicações de nível crítico, são esperadas baixas taxas de erro para evitar situações de risco indesejável.

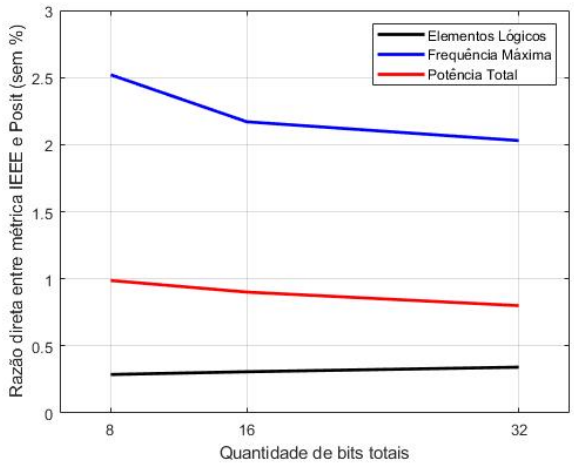
5 DISCUSSÃO DOS RESULTADOS

Foram apresentados neste trabalho três *designs* diferentes, para avaliação de *hardware* em variadas aplicações em aprendizagem de máquina. Sendo eles: uma multiplicação de matrizes quadradas simples, a função sigmoide e uma rede neural. Essa análise foi realizada fazendo a comparação entre a aritmética Posit e o padrão apresentado pela norma IEEE-754.

Para o *design* de multiplicação de matrizes, observamos na Figura 23, que para duas métricas avaliadas (elementos lógicos, frequência máxima) são melhores para a representação IEEE, porém a medida que o número de bits cresce, os Posits melhoram e tendem a superar o concorrente. Isso acontece pelo aumento de elementos lógicos ao utilizar a norma do IEEE, de 28,69% para 34,14%, e na queda de frequência máxima alcançada de 2,62 para 2,03 vezes. Esse resultado comprova o que Forget et al. (2021) encontrou, ao analisar uma comparação similar dessas representações com elementos isolados de soma e multiplicação . Apenas a potência não segue a tendência indicada, mostrando que, em relação a potência consumida, os Posits tendem a consumir mais energia para uma maior quantidade de bits usada.

Ainda na multiplicação de matrizes, um aspecto interessante, é fazer uma comparação com valores de bits diferentes. Nesses casos, valores ainda mais promissores para os Posits são alcançados em relação a todas as métricas analisadas, pois as razões de elementos lógicos supera os 70%, de frequências não fica acima de 1,7 vezes, e a razão de potência (fator de desvantagem para o Posit, mostrado anteriormente) fica ligeiramente maior que a unidade, mostrando que os Posits passam a gastar menos energia.

Figura 23: Razão das métricas avaliadas no *design* de multiplicação de matrizes



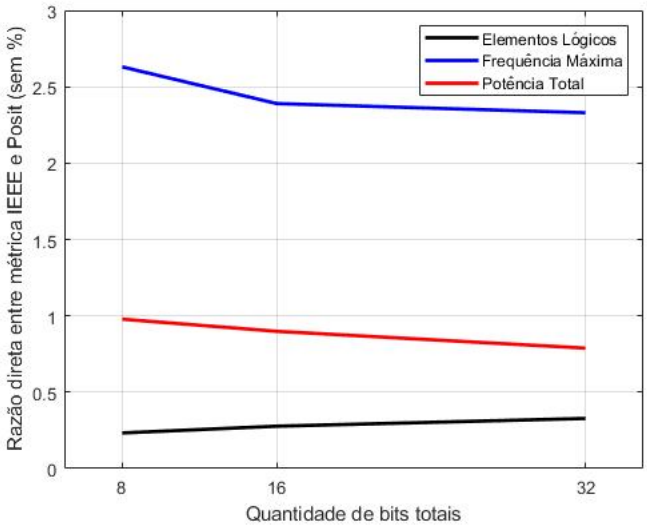
Fonte: Autoria Própria

Para a função sigmoide, subimos um pouco o nível de complexidade, dado que é uma operação mais complexa, que matematicamente envolve exponencial e divisão (ao contrário da multiplicação de matrizes, que usa apenas somas e multiplicações), mas modelada em um modelo de alto nível.

Portanto, para este caso, observamos tendências semelhantes aos observados para o *design* anterior (Figura 24), embora sejam números de menor magnitude, dada a necessidade de um *hardware* mais robusto. Verificamos esse resultado pelo crescimento da razão de elementos lógicos de 23,31% para 32,81%, em termos de frequência máxima, há uma queda de 2,63 para 2,33 vezes. Então, mais uma vez, obtemos valores que começam melhores para a representação IEEE e tendem a melhorar, conforme o tamanho da palavra binária aumenta. A desvantagem dos Posits em relação a potência também se confirma na queda de 97,81% para 78,86%. Mais uma vez, o resultado de Forget et al. (2021) é validado, e além disso também corrobora com a magnitude inferior dos resultados obtidos, mantendo as tendências de crescimento e decrescimento das razões mostradas.

Comparando agora quantidades de bits diferentes, com a quantidade do Posit menor, também obtemos resultados muito mais otimistas, em que a razão de elementos lógicos do IEEE fica sempre acima de 65% em comparação com o Posit, a frequência máxima acima de 1,6 vezes, e a potência consumida acima de 95%. Mais uma vez, os Posits se mostram melhores quando comparados com representações de quantidades maiores de bits.

Figura 24: Razão das métricas avaliadas no *design* da função sigmoide



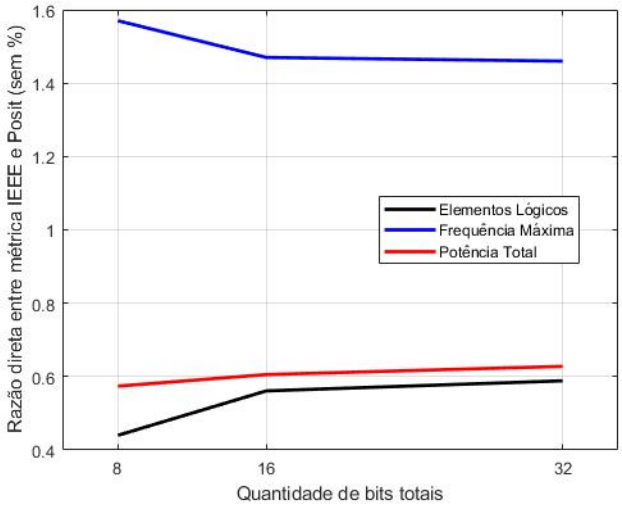
Fonte: Autoria Própria

Por fim, analisando a aplicação de maior complexidade, as redes neurais são *designs* que produzem resultados diretos para tomada de decisão, diferentemente dos casos anteriores, que eram operações intermediárias que fariam parte de um cálculo maior. Desse ponto de vista, a depender do porte do sistema, o intuito de construir um acelerador de *hardware* deve ser avaliado cuidadosamente, para que o sistema tenha um ganho de desempenho ao fazer com que um módulo seja executado de modo dedicado.

As quantidades de bits analisadas foram 16, 32 e 64 bits, na representação IEEE, enquanto o Posit de 16 bits foi utilizado como base, pois como a rede, por padrão, estava sendo treinada com 16 bits, caso 8 bits fossem utilizados, uma grande quantidade de informação seria perdida, prejudicando muito a acurácia final. Entretanto, uma análise semelhante a de Fatemi Langroudi et al. (2018) foi realizada, e um dos primeiros resultados obtidos foi que, de fato, para 16 bits, os Posit se equiparam ao ponto fixo no treino de redes neurais.

Na Figura 25, podemos observar as métricas da rede neural analisada. Para os elementos lógicos e frequência máxima, observamos a mesma tendência de melhora para os Posits, de acordo com que a quantidade de bits aumenta. A diferença se dá por conta da potência consumida, que também se mostra crescente para o IEEE, indicando que, em todas as métricas, o Posit tem uma forte tendência de melhora.

Figura 25: Razão das métricas avaliadas no *design* na rede neural



Fonte: Autoria Própria

Vale destacar que a análise realizada para esta aplicação de redes neurais foi diferente das anteriores, já que nesse caso houve conversões entre representações numéricas, como também a comparação não foi realizada sempre com a mesma quantidade de bits entre *floats* e Posits.

Portanto, para ao se aumentar a quantidade de bits da mantissa dos *floats* de 10 para 23 bits (de meia precisão para precisão simples), há a necessidade de mais elementos lógicos, já que a mantissa na saída do conversor para a rede em ponto fixo deve possuir 12 bits. Entretanto, ao se aumentar a quantidade de bits da mantissa de 23 para 52 bits (precisão simples para dupla precisão), a quantidade de elementos lógicos não aumenta muito, pois apenas 12 bits serão utilizados no ponto fixo que irá para a rede neural, enquanto os demais serão descartados porque a configuração de ponto fixo utilizada não suporta tal quantidade de bits de fração. Isso justifica a maior variação encontrada entre 16 e 32 bits em comparação com a variação existente entre 32 e 64 bits.

6 CONSIDERAÇÕES FINAIS

Foram apresentadas comparações de representações numéricas de computadores aplicadas a *designs* referentes a área de aprendizagem de máquina. Por meio do desenvolvimento de alguns IPs e utilização de alguns repositórios de código aberto, validações de algumas comparações já existentes na literatura poderam ser realizadas, bem como, produção de comparações diretas em aplicações mais complexas.

As validações realizadas ocorreram: primeiro, ao comprovar pela razão de elementos lógicos e frequência máxima, que blocos de soma e multiplicação necessitam de mais recursos quando comparados com a norma padrão IEEE-754; e segundo, ao comprovar que o treino de redes neurais para Posit de 16 bits e ponto fixo de 16 bits em armazenamento de pesos alcança a mesma acurácia final no resultado de saída da rede.

Das comparações realizadas, um resultado bastante importante, foi mostrar que, assim como dito por Uguen et al. (2019), o *hardware* para os *floats* são mais baratos para os padrões de quantidades de bits usados atualmente. Isso é reforçado por Gustafson e Yonemoto (2017), que mostrou que, por conta do processo mais serial que acontece nos Posits, devido principalmente a variação do tamanho dos campos na palavra binária, o *hardware* produzido é, em geral, mais complexo. Já que o padrão IEEE possui campos com comprimento fixo, podendo executar operações de forma paralela.

Desse modo, com a tecnologia que atualmente está no mercado, os Posits podem ainda enfrentar dificuldades relacionadas a um custo de *hardware* mais elevado, sobretudo quando nos referimos a sistemas embarcados. Porém, a tendência é que a tecnologia não só sofra otimizações, mas que evolua, e assim com as tendências de melhora dos resultados de Posit mostradas nesse trabalho, pode tornar viável diversas aplicações que necessitem de alta precisão e melhor variação de faixa dinâmica.

Como sugestão para trabalhos futuros, é utilizar de *softwares* mais robustos que os que aqui foram utilizados, que possibilitem utilizam uma maior quantidade de bits de entrada e saída em FPGA, bem como possuam dispositivos no catálogo que disponibilizem maior quantidade de elementos lógicos. Além disso, pode-se realizar um estudo semelhante em termos de circuitos integrados totalmente customizados, e avaliar qual seria a diferença entre a perspectiva de um sintetizador completo para um projeto realizado com um processo de uma fábrica de chips real. Já em termos de aplicação mais direta na inteligência artificial, uma maior variedade de redes neurais pode ser analisada, e um paralelo pode ser traçado entre a quantidade de camadas e a perda de informação em cada representação utilizada.

7 REFERÊNCIAS

E. Ternovoy, Mikhail G. Popov, Dmitrii V. Kaleev, Yurii V. Savchenko, and Alexey L. Pereverzev. Comparative Analysis of Floating-Point Accuracy of IEEE 754 and Posit Standards. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 1883–186, 2020. doi: 10.1109/EIConRus49466.2020.9039521.

Zoran Peric, Milan Savic, Milan Dincic, Nikola Vucic, Danijel Djosic, and Srdjan Milosavljevic. Floating point and fixed point 32-bits quantizers for quantization of weights of neural networks. In *2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 1–4, 2021. doi: 10.1109/ATEE52255.2021.9425265.

Seyed Hamed Fatemi Langroudi, Tej Pandit, and Dhireesha Kudithipudi. Deep learning inference on embedded devices: Fixed-point vs posit. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 19–23, 2018. doi: 10.1109/EMC2.2018.00012.

John L. Gustafson. *The end of error: Unum computing*. Chapman and Hall/CRC, 2017.

John Gustafson and I. Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4:71–86, 06 2017. doi: 10.14529/jsfi170206.

Melvin Johnson, Mike Schuster, Quoc Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, G.s Corrado, Macduff Hughes, and Jeffrey Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5, 11 2016. doi: 10.1162/tacl_a_00065.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. In *33rd International Conference on MachineLearning, New York, NY, USA, 2016. JMLR: W&CP volume48.*, 12 2015.

Manish Kumar Jaiswal and Hayden K.-H. So. Pacogen: A hardware posit arithmetic core generator. *IEEE Access*, 7:74586–74601, 2019. doi: 10.1109/ACCESS.2019.2920936.

Yohann Uguen, Luc Forget, and Florent de Dinechin. Evaluating the hardware cost of the posit number system. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 106–113, 2019. doi: 10.1109/FPL.2019.00026.

João Ranhel. *Eletrônica Digital, Verilog e FPGA*. Câmara Brasileira do Livro, 2021.

IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229.

Margareth Mee. Análise comparativa da precisão em ponto flutuante dos padrões IEEE 754 e Posit. 2019. Trabalho de Conclusão de Curso - UFCG.

Varun Gohil, Sumit Walia, Joycee Mekie, and Manu Awasthi. Fixed-posit: a floating-point representation for error-resilient applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(10):3341–3345, 2021.

Manish Kumar Jaiswal and Hayden K.-H So. Architecture generator for type-3 unum posit adder/subtractor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018. doi: 10.1109/ISCAS.2018.8351142.

Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: an overview. In *Journal of physics: conference series*, volume 1142, page 012012. IOP Publishing, 2018.

Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):1–21, 2021.

Jianyu Chen, Zaid Al-Ars, and H. Peter Hofstee. A matrix-multiply unit for posits in reconfigurable logic leveraging (open) capi. In *Proceedings of the Conference for Next Generation Arithmetic*, pages 1–5, 2018.

Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.

Inácio Guimarães. Modelos de regressão logística oculto e de componentes principais para reconhecimento e classificação de padrões com variável resposta politômica. *oai:ufpr.br:228716*, 03 2022.

Clive Maxfield. *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier, 2004.

Abdulmajeed Adil Yazdeen, Subhi R.M. Zeebaree, Mohammed Mohammed Sadeeq, Shakir Fattah Kak, Omar M. Ahmed, and Rizgar R. Zebari. FPGA implementations for data encryption and decryption via concurrent and parallel computation: A review. *Qubahan Academic Journal*, 1(2):8–16, 2021.

S.R. Zebari and Numan O. Yaseen. Effects of parallel processing implementation on balanced load-division depending on distributed memory systems. *J. Univ. Anbar Pure Sci*, 5(3):50–56, 2011.

S.R. Zeebaree, Lailan M. Haji, Imad Rashid, Rizgar R. Zebari, Omar M. Ahmed, Karwan Jacksi, and Hanan M. Shukur. Multicomputer multicore system influence on maximum multi-processes execution time. *TEST Engineering & Management*, 83(03):14921–14931, 2020.

Intel. Intel® Quartus® Prime Software Suite: The intuitive high-performance design environment. Disponível em: `https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html`, 2018. Online. Acesso em: Março de 2022.

Intel. Intel® Quartus® Prime Design Software Brochure. Disponível em: `https://www.intel.com/content/dam/www/central-libraries/us/en/documents/quartus-prime-software-brochure.pdf`, 2021. Online. Acesso em: Março de 2022.

D Alonso, J Suardiaz, PJ Navarro, PM Alcover, and JA Lopez. Automatic generation of VHDL code from traditional ladder diagrams applying a model-driven engineering approach. In *2009 35th Annual Conference of IEEE Industrial Electronics*, pages 2416–2421. IEEE, 2009.

Saurabh Sinha. 32bit Floating Point ALU using Verilog. Disponível em: `https://github.com/sks9691901/32bit-Floating-Point-ALU-using-Verilog`, 2021. Online. Acesso em: Novembro de 2021.

Manish Kumar Jaiswal. PACoGen. Disponível em: `https://github.com/manish-kj/PACoGen`, 2019. Online. Acesso em: Novembro de 2021.

Aniket Dukare. Sigmoid function. Disponível em: `https://github.com/aniket0511/Sigmoid-Function`, 2019. Online. Acesso em: Novembro de 2021.

Kizheppatt Vipin. Neural network. Disponível em: `https://github.com/vipinkmenon/neuralNetwork`, 2020. Online. Acesso em: Dezembro de 2022.

Kizheppatt Vipin. ZyNet: Automating Deep Neural Network Implementation on Low-Cost Reconfigurable Edge Computing Platforms. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 323–326, 2019. doi: 10.1109/ICFPT47387.2019.00058.

Luc Forget, Yohann Uguen, and Florent de Dinechin. Comparing posit and ieee-754 hardware cost. 2021.