UNIDADE ITUIUTABA | UEMG
Intercursos Revista Científica
Ciências Exatas

IMPLEMENTAÇÃO DAS OPERAÇÕES DE UMA UNIDADE DE PONTO FLUTUANTE DE 32 BITS BASEADA NO PADRÃO IEEE 754 EM VERILOG

IMPLEMENTATION OF THE OPERATIONS OF A 32-BIT FLOATING POINT UNIT BASED ON THE IEEE 754 STANDARD IN VERILOG

ERIKC JOSÉ FERREIRA SANTOS, KÁTIA LOPES SILVA, MAURO HEMERLY GAZZANI

RESUMO
Uma UPF (Unidade de Ponto Flutuante) de forma geral é um coprocessador matemático, o qual faz parte de um sistema computacional especialmente projetado para realizar operações em números de ponto flutuante. As operações típicas que são tratadas pela UPF são adição, subtração, multiplicação e divisão. Este trabalho apresenta modelagem e simulação de uma UPF de 32 bits usando a linguagem Verilog no ambiente da plataforma EDA Playground. A UPF de 32 bits foi modularizada em quatro unidades funcionais, sendo uma para cada tipo de operação (adição, subtração, multiplicação e divisão). As unidades funcionais possuem uma saída de 32 bits, que representa o resultado da operação realizada, e outra saída que representa os sinalizadores (flags) que sinalizam o estado do resultado das unidades. O principal foco deste trabalho é a análise, projeto e implementação da UFM (Unidade Funcional de Multiplicação). A verificação funcional foi realizada em uma bancada de teste (*TestBench*), onde vários casos de testes foram simulados envolvendo situações que podem acontecer no resultado da operação tais como: *overflow, underflow* e exceções (NAN, infinito). Os resultados apresentados pela UFM que a UPF de 32 bits implementa, demostram que ela funcionou adequadamente para os casos de testes gerados.

Palavras chave: UPF 32 bits. Verilog. UFM, IEEE 754.

ABSTRACT
A FPU (Floating Point Unit) in general is a math coprocessor, which is part of a computer system specially designed to perform operations on floating point numbers. Typical operations that are handled by the FPU are addition, subtraction, multiplication, and division. This work presents modeling and simulation of a 32-bit FPU using the Verilog language in the EDA Playground platform environment. The 32-bit FPU was modularized into four functional units, one for each type of operation (addition, subtraction, multiplication and division). The functional units have a 32-bit output, which represents the result of the operation performed, and another output that represents the flags (flags) that signal the state of the units' result. The main focus of this work is the analysis, design and implementation of the FMU (Functional Multiplication

Unit). The functional verification was carried out in a test bench (TestBench), where several test cases were simulated involving situations that can happen in the result of the operation such as: overflow, underflow and exceptions (NAN, infinity). The results presented by the FMU that the 32-bit FPU implements worked properly for the generated test cases.

Keywords: FPU 32 bits. Verilog. FMU, IEEE 754.

INTRODUÇÃO

Projetos de sistemas digitais usando Linguagens de Descrição de Hardware (*Hardware Description Language* - HDL) é um campo com grande potencial de crescimento na atualidade. As aplicações dos projetos digitais estão presentes em nosso dia a dia, incluindo computadores, calculadoras e câmeras de vídeo etc.

A linguagem HDL permite projetar hardware digital utilizando um software. Inegável que uma grande economia de tempo é constatada ao projetar sistemas usando uma HDL. Como ponto relevante pode-se citar o ganho em vantagem competitiva, com a redução do tempo de colocação no mercado de um sistema. Outra vantagem é a possibilidade de realizar variadas simulações operacionais que possibilitam ter um projeto otimizado antes da implantação do sistema no hardware. Desta forma, vários erros podem ser encontrados durante a simulação e suas correções implementadas, reduzindo os custos com hardware.

A linguagem Verilog, cuja padronização atual é a IEEE 1364-2005, é uma HDL usada para modelar sistemas eletrônicos. Um subgrupo da linguagem é tipicamente utilizado para propósitos de síntese e a linguagem completa pode ser utilizada para modelamento e simulação. Verilog suporta a projeção, verificação e implementação de projetos analógicos, digitais e híbridos em vários níveis de abstração. Um dos principais atributos da modelagem de circuitos por linguagem descritiva frente à modelagem por captura de esquemático, está ligado ao fato de que desta maneira o projeto torna-se independentemente da plataforma de desenvolvimento (IDE) na qual se está trabalhando. Além disso, adotando-se as boas práticas na descrição dos circuitos, o compilador é inclusive capaz de contornar a ausência de determinado recurso na tecnologia onde o circuito será sintetizado, conferindo uma portabilidade desse modelo para qualquer dispositivo (target) onde pode ser sintetizado. (CAVANAGH, 2010).

Segundo Sahu e Dev (2012), uma Unidade de Ponto Flutuante (UPF) de forma geral é um coprocessador matemático, o qual faz parte de um sistema computacional especialmente projetado para realizar operações em números de ponto flutuante. As operações típicas que são tratadas pela UPF são adição, subtração, multiplicação e divisão.

Do ponto de vista da arquitetura, a UPF é um coprocessador que opera em paralelo com a unidade inteira do processador. A UPF obtém suas instruções da mesma instrução decodificador e sequenciador e compartilha o barramento do sistema. Além disso, unidade lógica aritmética (ULA) e a UPF operam independentemente e em paralelo. No caso da Intel, a microarquitetura de um processador Intel varia entre as várias famílias de processadores. Por exemplo, o processador Pentium Pro tem duas unidades inteiras e duas UPFs; enquanto, o

processador Pentium tem duas unidades inteiras e uma UPF, e o processador Intel486 tem uma unidade inteira e uma UPF. (INTEL,1999).

Este trabalho apresenta modelagem e simulação de uma UPF que executa as funções básicas com especial foco na operação de multiplicação. As atividades envolvidas na implementação são: a manipulação de dados de ponto flutuante, a conversão de dados para o formato IEEE 754, a execução de qualquer uma das operações aritméticas como adição, subtração, multiplicação.

ESTADO DA ARTE

DSPs (do inglês *Digital Signal Processor*) ou quaisquer outros processadores que envolvam operações complexas como multiplicação e/ou acumulação operações com alta precisão necessitam de Unidade de Ponto Flutuante (UPF) para garantir um desempenho conveniente.

Uma unidade de ponto flutuante contém sequência de dígitos em três partes que é sinal, mantissa e expoente. O sinal pode ser positivo ou negativo, mantissa é sequência de dígitos e expoente é a potência de magnitude. A operação principal de uma unidade de ponto flutuante inclui adição, subtração, multiplicação, divisão e raiz quadrada. A unidade de ponto flutuante pode ser de precisão simples ou dupla.

O principal marco para teoria de ponto flutuante aconteceu em 1985 quando o Padrão para Aritmética de Ponto Flutuante Binário IEEE Std754 foi apresentado pelo *Institute of Electrical and Electronics Engineers* (IEEE) e depois atualizado em 1990 pelo mesmo instituto. Neste padrão, atualmente recomendado também pelo ANSI *American National Standard Institute* (ANSI), tem-se as normas a serem seguidas pelos fabricantes de computadores e construtores de compiladores de linguagens científicas, ou de bibliotecas de funções matemáticas, na utilização da aritmética binária para números de ponto flutuante. (IEEE,1985)

Cavanagh (2010) apresentou em seu livro a aritmética computacional para pontos fixos, decimais, e representações de números de ponto flutuante para as operações de adição, subtração, multiplicação e divisão, e para implementar essas operações usando Verilog. As diferentes construções de modelagem suportadas pelo Verilog são descritas em detalhes. No caso de ponto flutuante, forma apresentados os algoritmos para implementação das operações de adição, subtração, multiplicação e divisão, todas baseadas no padrão do Instituto de Engenheiros Elétricos e Eletrônicos (IEEE) para Aritmética de Ponto Flutuante Binário IEEE Std 754-1985. Todos os algoritmos foram implementados utilizando a linguagem Verilog.

Sahu e Dev (2012) publicaram em seu trabalho de graduação em Ciência da computação do *National Institute Of Technology Rourkela* na Índia, a modelagem e implementação de uma Unidade de ponto flutuante baseada no padrão IEEE 754. A implantação mostrou-se bastante eficiente e executa as funções básicas e transcendentais com uma reduzida complexidade quando comparada com as implementações da família x87 do fabricante Intel. As velocidades de *clock* ficaram próximas, porém a utilização de memória foi bastante reduzida.

Ziaullah e Munaff (2015) implementaram operações típicas de uma UPF. As funções executadas foram a manipulação de dados de ponto flutuante, conversão de dados para o formato IEEE 754, execução de qualquer uma das seguintes operações

aritméticas como adição, subtração, multiplicação, divisão. Todos os algoritmos foram avaliados no ambiente Modelsim. Segundo os autores, todas as funções foram construídas por algoritmos com diversas mudanças incorporadas. Consequentemente, todas as funções da unidade são únicas em certos aspectos, e essas funções tenderão a mostrar eficiência e velocidade comparáveis e, se canalizadas, maior taxa de transferência. Na verdade, os autores não deixam muito claro as mudanças realizadas, mostrando apenas os resultados atingidos.

Upendar (2018) apresentou uma implementação ASIC (*Aplication Specifi Integrated Circuit*) de alta velocidade de uma unidade de ponto flutuante que pode executar funções de adição, subtração, multiplicação e divisão em operandos de 32 bits que usam o padrão IEEE 754. As Unidades de pré-normalização e pós-normalização também são discutidas juntamente com sua manipulação. Todas as funções são construídas por algoritmos eficientes viáveis com diversas mudanças incorporadas que podem melhorar a latência geral e, se for canalizado, maior taxa de transferência. No caso da multiplicação, o algoritmo Booth foi utilizado, pois este oferece uma forma mais eficiente de multiplicar inteiros binários com muito menos operações de adição/subtração. Os algoritmos são modelados em Verilog HDL e o código RTL para somador, subtrator, multiplicador, são sintetizados usando HDL designer series e XILINX.

Savaliya e Rudani (2020) implementaram um UPF, baseada no padrão IEEE 754, para valores de ponto flutuante de precisão simples de 32 bits. A principal aplicação desta UPF está no processador DSP, para o processamento de sinais, onde é necessário um valor com alta precisão e por se tratar de um processo iterativo, o cálculo deve ser o mais rápido possível. Este projeto apresenta a implementação de uma unidade aritmética de vírgula flutuante eficiente de 32 bits usando Verilog com o objetivo de analisar o problema durante a implementação e entender a forma de contornar o problema a fim de melhorar o desempenho do sistema. Os resultados mostraram-se satisfatórios e os autores sugeriram como trabalho futuro a implementação de um conversor, para realizar a conversão da saída em forma IEEE 754 em representação decimal e fornecer a saída como um sistema numérico decimal.

Maladkar e Aradhya (2021) desenvolveram uma unidade de ponto flutuante otimizada para que o atraso fosse reduzido e a eficiência fosse aumentada. A unidade de ponto flutuante foi escrita de acordo com o padrão IEEE 754 e todo o projeto foi codificado em Verilog HDL e simulado com Xilinx 14.7 (2022). Na proposta, a eficiência do projeto é aumentada com menos atraso computacional em comparação com um método tradicional. Os resultados são melhorados em 12% com o uso do multiplicador védico que é um atraso de 4.450ns em comparação com 5.123ns com um multiplicador de matriz. O projeto pode ser usado em computação matemática, processamento de sinais, gráficos e outros que necessitam de melhor velocidade de cálculos e operações que envolvem ponto flutuante.

FUNDAMENTAÇÃO TEÓRICA

Linguagem Verilog

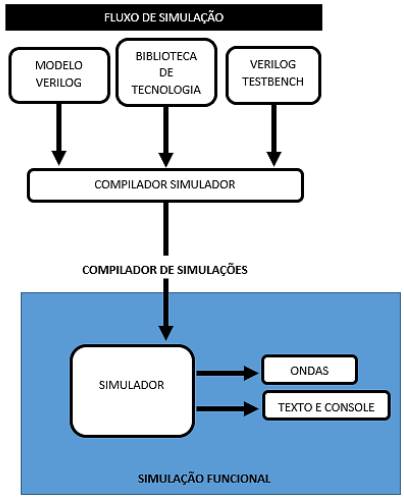
A linguagem Verilog é uma linguagem de descrição de hardware (HDL) que fornece um meio de especificar um sistema digital em uma ampla gama de níveis de

abstração. A linguagem suporta os estágios conceituais de projeto com sua fase comportamental de abstração, e o estágio posterior de implementação com suas abstrações estruturais. A linguagem inclui construções hierárquicas, permitindo ao desenvolvedor controlar a complexidade de uma descrição.

Foi originalmente projetado entre 1983 e 1984 como um produto proprietário de verificação e simulação. Mais tarde, várias outras ferramentas de análise foram desenvolvidas em torno da linguagem, incluindo um simulador de falhas e um analisador de tempo. Mais recentemente, a Verilog também forneceu a especificação de entrada para ferramentas de síntese lógica e comportamental. A linguagem Verilog tem sido fundamental para fornecer consistência entre essas ferramentas. A linguagem foi padronizada primeiramente como padrão IEEE 1364-1995. Atualmente a padronização vigente é o IEEE 1364-2005.

As etapas de uma simulação são mostradas na figura 1 na qual pode-se notar o fluxo de uma formação de um código em Verilog.

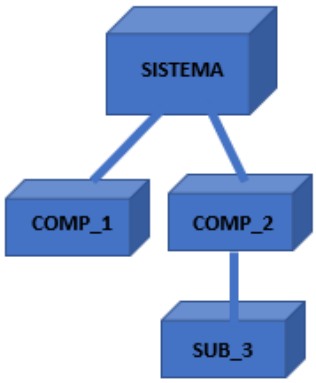
Figura 1- Visão geral da simulação em Verilog



Fonte: Modificado de Altera Corporation (2008)

As descrições estruturais do Verilog são compostas de vários blocos de código e permitem a introdução de hierarquia em um projeto (Figura 2). Os elementos da estrutura do programa são: o módulo, a porta e o sinal. Um modelo (sistema) em Verilog é composto de módulos (componentes). Desta forma, como mostrado na figura 2 tem-se que um sistema pode ser composto de vários componentes os quais podem ser compostos de subsistemas. Os sistemas instanciam os componentes 1 e 2 e este último instância o subsistema 3.

Figura 2 - Estrutura de um programa em Verilog



Fonte: Modificado de LaMeres (2019)

Em Verilog, um componente é representado por um módulo, que é a unidade básica. A declaração do módulo fornece a visão "externa" do componente; ele descreve o que pode ser visto de fora, incluindo as portas dos componentes. O corpo do módulo fornece uma visão "interna"; descreve o comportamento ou a estrutura do componente.

Um módulo representa um texto de declaração que detalha a função do módulo utilizando as construções Verilog. Ele representa através de comandos e estruturas da linguagem a estrutura física do hardware e seu comportamento. Desta forma, o módulo em Verilog manipula as entradas e produz as saídas do circuito lógico.

Conforme Cavanagh (2010), Verilog possui elementos lógicos predefinidos chamados primitivos. Esta lógica embutida primitivos são elementos estruturais que podem ser instanciados em um projeto maior para formar uma estrutura mais complexa. Exemplos de primitivas lógicas integradas são as operações lógicas de AND, OR, XOR e NOT.

Ponto Flutuante e o Padrão IEEE 754

Em, 1985, o padrão IEEE 754 para aritmética de ponto flutuante foi estabelecido e, desde a década de 1990, as representações mais comumente encontradas são aquelas definidas pelo IEEE.

A UPF da arquitetura Intel (IA) fornece recursos de processamento de ponto flutuante de alto desempenho. Ele suporta os tipos de dados real, integer e BCD-integer e os algoritmos de processamento de ponto flutuante e arquitetura de tratamento de exceção definidos no IEEE padrões 754 e 854 para aritmética de ponto flutuante. A UPF executa as instruções do fluxo de instruções normal do processador e melhora muito a eficiência dos processadores em lidar com os tipos de operações

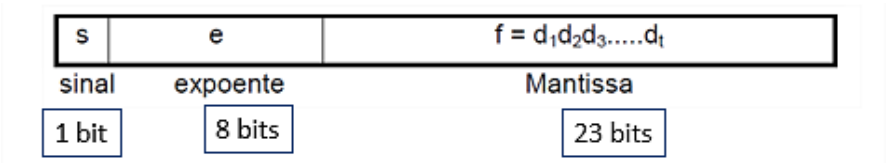
de processamento do ponto flutuante de alta precisão comumente encontradas em aplicações científicas, de engenharia e de negócios. (INTEL,1999).

Cada geração de UPFs das arquiteturas de computadores atualmente são projetadas para fornecer estabilidade, resultados precisos quando programados usando algoritmos simples de “lápis e papel”, trazendo a funcionalidade e o poder da computação numérica precisa para o usuário final. O padrão IEEE 754 aborda especificamente essa questão, reconhecendo a importância fundamental de tornar os cálculos numéricos fáceis e seguros de usar. Viana (2022) descreve o padrão IEEE 754:

> O padrão IEEE 754, recomendado pelos institutos ANSI (*American National Standard Institute*) e IEEE (*Institute of Electrical and Eletronic Engineers*), refere-se às normas a serem seguidas pelos fabricantes de computadores e construtores de compiladores de linguagens científicas, ou de bibliotecas de funções matemáticas, na utilização da aritmética binária para números de ponto flutuante. As recomendações são relativas ao armazenamento de dados numéricos, métodos de arredondamento, tratamento de casos de *underflow* e overflow, formas de realização das quatro operações aritméticas básicas e implementação de funções nas linguagens de programação. (VIANA,2022)

Para aumentar a velocidade e a eficiência dos cálculos de números reais, os computadores ou UPFs normalmente representam números reais em um formato binário de ponto flutuante. Neste formato, um número real tem três partes: um sinal, um significativo e um expoente. A figura 3 mostra o formato binário de ponto flutuante em precisão simples (32 bits). Desta forma, N = s + e + f, corresponde ao tamanho da palavra em bits. No caso da precisão simples: s = 1, e =8 e f = 23 bits, totalizando N = 32 bits.

Figura 3 – Formato binário de ponto flutuante



Fonte: INTEL (1999)

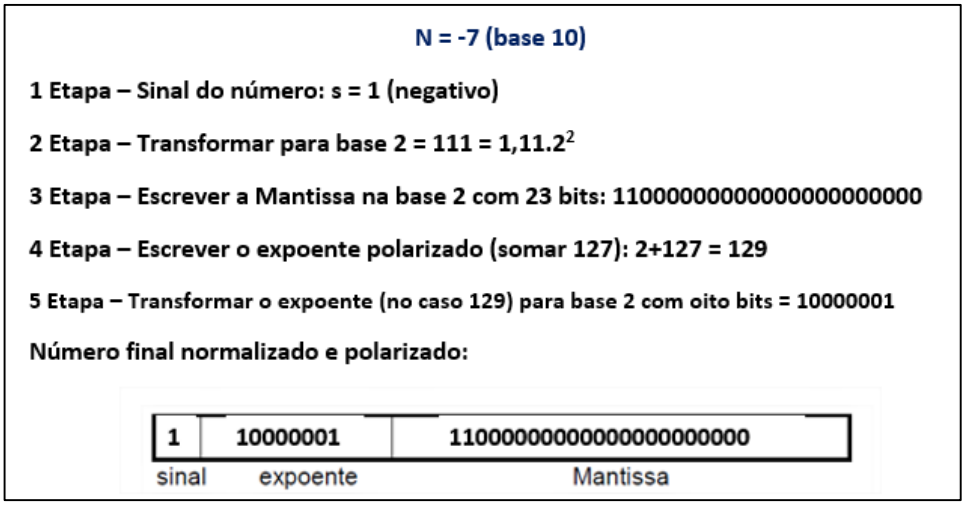
Um ponto muito importante do padrão IEEE 754 está relacionado com o expoente. Para facilitar as implementações, o expoente é polarizado (*bias* em inglês), ou seja, trabalha-se sempre com expoente positivo. Deste modo, para o caso de 32 bits o padrão recomenda somar 127 ao expoente.

O padrão IEEE 754 trabalha com o número em ponto flutuante normalizado, ou seja, o primeiro dígito (d1) deve ser diferente de zero para assegurar a unicidade de

representação, e manter sempre a precisão máxima suportada pela mantissa (Vianna, 2022).

A figura 4 mostra um exemplo de um número decimal em formato IEEE 754 na base 2:

Figura 4 – Exemplo de um número no formato IEEE 754 normalizado e polarizado



Fonte: Autor

O padrão IEEE 754 possui algumas classes especiais de representação para atender os resultados de operações aritméticas que não são suportadas pelos computadores; como por exemplo: divisão por zero, overflow. Nestes casos tem-se as representações mostradas no quadro 1.

Quadro 1 – Classes especiais de representação do Formato IEEE754

Caso	Sinal	Expoente Não Polarizado	Expoente Polarizado	Mantissa	Significado
Zeros	0	-127	0	0	+0
Zeros	1	-127	0	0	-0
Infinitos	0	128	255	0	$+\infty$
Infinitos	1	128	255	0	$-\infty$
NaN (Not a Number)	1 ou 0	128	255	$\neq 0$	exceções, invariavelmente, intratáveis
Não Normalizado	1 ou 0	-126 (não vale o 127)	0	$0.f_1 f_2 f_3 ... f_{23}$	Útil para números pequenos

Fonte: Autor

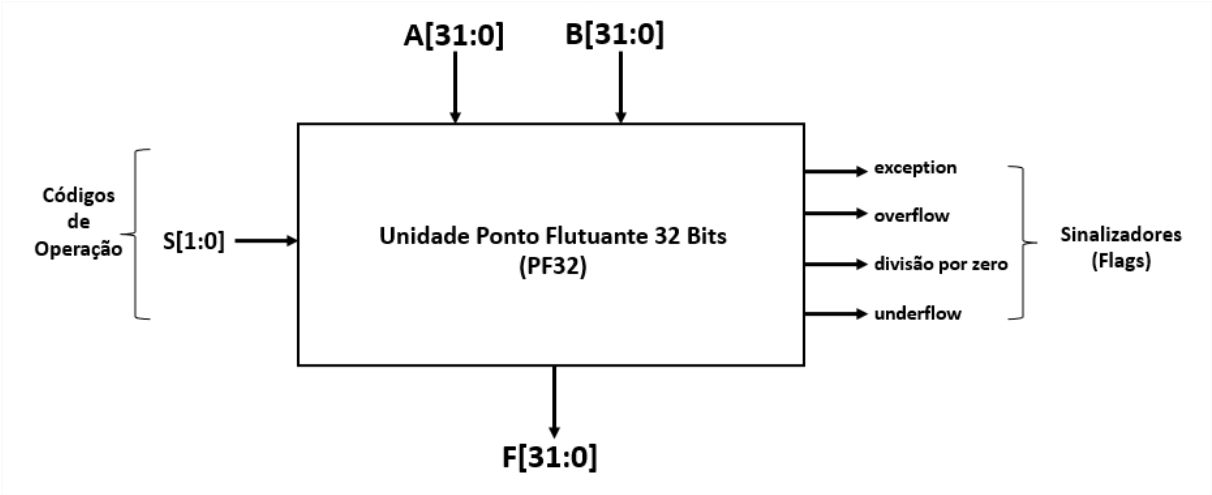
Unidade de Ponto Flutuante (UFP)

Uma unidade de ponto flutuante (UPF, coloquialmente um coprocessador matemático) é uma parte de um sistema de computador especialmente projetado para realizar operações em números de ponto flutuante. As operações típicas são adição, subtração, multiplicação, divisão e outras, conforme a implementação.

Qualquer operação matemática, como adição, subtração, multiplicação ou divisão, pode ser realizada pela unidade de processamento de inteiros ou pela UPF. Quando uma CPU recebe uma instrução, ela a envia automaticamente para o processador correspondente. Por exemplo, 20 + 3 seria processado como um cálculo inteiro pela Unidade Lógica Aritmética (ULA), enquanto 20,3245 + 3,789 seria enviado para a UPF.

Na computação, uma UPF é usada para representação de fórmulas de números reais como uma aproximação para oferecer suporte a uma compensação entre intervalo e precisão. Frequentemente usada em sistemas com números reais muito pequeno ou grande que requerem tempos de processamento rápido. Em geral, um número de ponto flutuante é representado com um número fixo de dígitos significativos e escalado usando um expoente em alguma base fixa; a base para a escala é normalmente dois, dez ou dezesseis.

Figura 5 - Visão geral da UPF



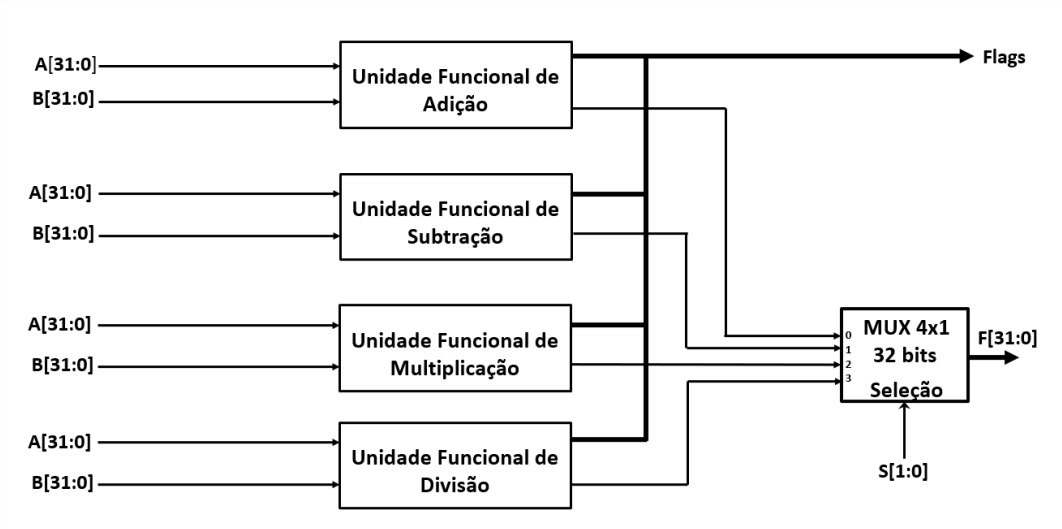
Fonte: Autor

A Figura 5 ilustra a visão geral da UPF de 32 bits, onde as entradas A e B de 32 bits representam os operandos, e a entrada de 2 bits representa os códigos de operação (adição, subtração, multiplicação e divisão). Os dois bits combinados S1 e S0 (00: adição, 01: subtração, 10: multiplicação e 11: divisão) identificam a unidade funcional da UPF que realiza a operação correspondente. O sinal de saída F de 32 bits representa o resultado da última operação realizada pela UPF, e os outros 3 bits de saída sinalizam o estado do resultado da UPF.

PROJETO DA UPF DE 32 BITS

A UPF de 32 bits é modularizada em unidade funcionais e ilustrada na figura 6. As quatro unidades funcionais são denominadas como seguem: Unidade Funcional de Adição (UFA), e a Unidade Funcional de Subtração (UFS), Unidade Funcional de Multiplicação (UFM) e Unidade Funcional de Divisão (UFD). As 2 entradas de 32 bits representam os operandos das respectivas unidades funcionais. Cada uma das unidades funcionais possui uma saída de 32 bits, que representa o resultado da operação realizada, e outra saída que representa os sinalizadores (flags) que sinalizam o estado do resultado das unidades. O MUX 4:1 utiliza os bits S1 a S0 para selecionar a saída de qual unidade será aquela da UFP.

Figura 6 - Unidades Funcionais da UPF de 32 Bits

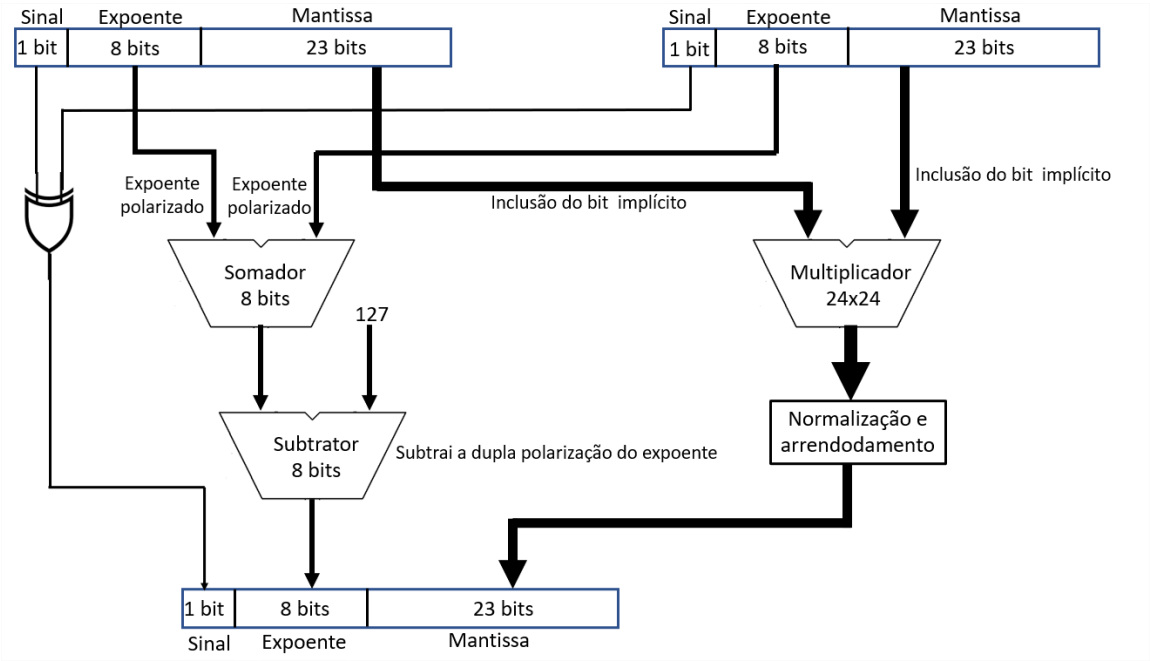


Fonte: Autor

O principal foco deste trabalho é a análise, projeto e implementação da Unidade Funcional de Multiplicação. Para multiplicar dois números em ponto flutuante, 1.f1 x 2e1 * 1.f2 x 2e2 (onde e1 e e2 são expoentes já polarizados), os seguintes passos são necessários: [1] Verificar se todos os bits de e1 ou e2 sejam iguais a zero, e, portanto, o número está desnormalizado e o bit implícito da mantissa correspondente é definido como 0 (0.f x 2e) formando uma mantissa de 24 bits. Caso contrário, o número está normalizado e bit implícito é definido como 1 (1.f x 2e);[2] Multiplicar as Mantissas incluindo o bit implícito; [3] Somar os expoentes: somar e1 com e2 e subtrair 127 (01111111) para eliminar a dupla polarização no resultado, $e1+e2-01111111_2$; [4] Obter o sinal do produto: s1 XOR s2; [5] Normalizar o resultado, se necessário (como por exemplo o resultado da multiplicação seja igual a 10.xxxx..x. Neste caso desloca-se o ponto decimal uma vez para a esquerda e soma-se 1 ao expoente determinado no passo 3); [6] Arredondar o resultado para caber nos bits disponíveis da mantissa, no caso 23 bits; [7] Determinar os flags de ocorrência de *underflow*/*overflow*, ou se a operação é inválida.

A Figura 7 mostra um diagrama de blocos desta unidade, não inclusa a determinação dos flags citados no passo 7 no parágrafo anterior.

Figura 7 – Visão geral da Unidade Funcional de Multiplicação (UFM)



Fonte: Autor

Os códigos dos módulos em Verilog descritos são mostrados nas figuras 8 e 9 para Unidade funcional de multiplicação 32 Bits.

A implementação em linguagem Verilog pode ser realizada com vários IDEs baseados em Verilog com FPGAs Altera e Xilinx (2022) que são suportados pelo Altera Quartus II e Xilinx ISE IDEs disponíveis no mercado. No caso deste trabalho foi utilizado o EDA Playground (2022).

O EDA Playground possui um ambiente online para simular (utilizando vários simuladores disponíveis: o usuário pode escolher qual deseja utilizar) e sintetizar implementações nas linguagens SystemVerilog, Verilog, VHDL, C ++ / SystemC e outros HDLs. Os resultados das simulações podem ser visualizados na forma de ondas usando o visualizador de ondas baseado em navegador EPWave e por meio de definições de casos de teste no *Test Bench*.

Figura 8 – Código Verilog da Unidade Funcional de Multiplicação – Parte 1

```
// O módulo de multplicação em formato IEEE 754 com os expoentes polarizados.
// sinal = opd[31]    positivo: 0   negativo: 1
// expoente = opd[30:23] polarizado com excesso 127
// mantissa = opd[22:0]

module mult_pf(a,b,mult,exception,underflow,overflow);

input [31:0] a, b; // operandos
output overflow,underflow,exception;  // flags
output [31:0] mult;  // resultado da multiplicação dos operandos

  wire sinal,round,normalisado,zero,infinity;
  wire [8:0] expoente,soma_expoente;
  wire [22:0] produto_mantissa;
  wire [23:0] opd_a,opd_b;
  wire [47:0] produto,produto_normalisado;


  assign sinal = a[31] ^ b[31];                       // XOR do 32o bit
  // Operação inválida
  assign exception = (&a[30:23]) | (&b[30:23]);       // exception = 1 : expoente de a ou b igual a 255


  // Para reduzir a perda de precisão quando ocorre um underflow, o formato IEEE 754 inclui a capacidade
  // de representar mantissas menores do que são possíveis na representação normalizada, tornando
  // o dígito inicial implícito um 0.
  // Se todos os bits de E1 ou E2 forem 0 ===> O número é desnormalizado e o bit implícito da mantissa
  // correspondente é definido como 0.
  assign opd_a = {|a[30:23],a[22:0]};  // mantissa de 24 bits c/ bit implícito
  assign opd_b = {|b[30:23],b[22:0]};

  // multiplicando a e b com o bit implícito da mantissa correspondente
  assign produto = opd_a * opd_b;
```

Fonte: Autor

Figura 9 - Código Verilog da Unidade Funcional de Multiplicação – Parte 2

```
  // multiplicando a e b com o bit implícito da mantissa correspondente
  assign produto = opd_a * opd_b;

  // OR para todos os bits dos últimos 22 bits para propósito de arredondamento
  assign round = |produto_normalisado[22:0];
  assign normalisado = produto[47];

  // normalização baseado no 48o bit
  assign produto_normalisado = normalisado ? produto : produto << 1;
  assign produto_mantissa = produto_normalisado[46:24] + (produto_normalisado[23] & round);
  assign zero = exception ? 1'b0 : ((produto_mantissa == 23'd0) ? 1'b1 : 1'b0);
  assign soma_expoente = a[30:23] + b[30:23];

  //remove uma polarização - polarização dobrada devido à soma
  assign expoente = soma_expoente - 8'd127 + normalisado;

  // Se expoente = 255 e mantissa = 0, então infinito. O bit de sinal define o +/-.

  // o overflow ocorre quando uma operação aritmética resulta
  // em uma magnitude maior do que pode ser expressa com um expoente de 128
  // exemplo: 2**120 * 2**100 = 2**220
  // Se soma dos expoentes polarizados maior que 255 então overflow
  assign overflow = ((expoente[8] & !expoente[7]) & !zero);

// o underflow ocorre quando a magnitude fracionária é muito pequena com um expoente abaixo de -127
// exemplo: 2**-120 * 2**-100 = 2**-220
// soma dos expoentes polarizados menor que 255 então underflow
  assign underflow = ((expoente[8] & expoente[7]) & !zero) ? 1'b1 : 1'b0;
  assign mult = exception ? 32'd0 : (zero ? {sinal,31'd0} : (overflow ? {sinal,8'hFF,23'd0} :
                (underflow ? {sinal,31'd0} : {sinal,expoente[7:0],produto_mantissa})));

endmodule
```

Fonte: Autor

RESULTADOS E DISCUSSÃO

Conforme LaMeres (2019) um dos componentes essenciais do fluxo do projeto digital moderno é verificar a funcionalidade por meio simulação. Essa verificação funcional é realizada em uma bancada de teste (*TestBench*). Uma bancada de teste é um modelo Verilog que instancia o sistema a ser testado como um subsistema, gera os padrões de entrada para conduzir o subsistema e observa as saídas. Bancadas de teste são usadas apenas para simulação, para que se possa usar técnicas de modelagem abstratas que não são sintetizáveis para gerar os padrões de estímulo.

As construções sintáticas em Verilog podem ser usadas para relatar o status de um teste e verificar automaticamente se as saídas estão corretas.

Quadro 2 – Casos de Testes da UFM de 32 Bits

Caso de teste	Entradas	Saída esperada
Multiplicação a * b #1	a = 40C9999A (+6,3) b = +infinity[4]	mult = 00000000 exception[1] = 1
Multiplicação a * b #2	a = 7B86D0AA (+1.39999997806e+36) b = 71834436 (+1.299999944e+30)	mult = 00000000 overflow[2] = 1
Multiplicação a * b #3	a = 4234851F (+45,13) b = 427C851F (+63,13)	mult = 453210E9
Multiplicação a * b #4	a = 4049999A (+3,15) b = C1663D71 (-14,39)	mult = C2355062
Multiplicação a * b #5	a = C1526666 (-3.15) b = C240A3D7 (-48,16)	mult = 441E5375
Multiplicação a * b #6	a = 41C80000 (+25,0) b = 42480000 (+50,0)	mult = 449C4000
Multiplicação a * b #7	a = 3ACA62C1 (+0.00154408081) b = 3ACA62C1 (+0.00154408081)	mult = 361FFFE7
Multiplicação a * b #8	a = 037F3637 (+7.50000004534e-37) b = 0D7D1FDD (+7.79999981785e-31)	mult = 00000000 underflow[3] = 1
Multiplicação a * b #9	a = 00000000 (0) b = 00000000 (0)	mult = 00000000
Multiplicação a * b #10	a = 7F800000 (+infinity[4]) b = 00000000 (0)	mult = 00000000 exception[1] = 1
Multiplicação a * b #11	a = 7F800000 (+infinity[4]) b = 7F800000 (+infinity[4])	mult = 00000000 exception[1] = 1

[1]**exception**: operação inválida. [2]**overflow**: ocorre quando uma operação aritmética resulta em uma magnitude maior do que pode ser expressa com um expoente de 128. Exemplo: 2^120 * 2^100 = 2^220.

[3]**underflow**: ocorre quando a magnitude fracionária é muito pequena com um expoente abaixo de -127. Exemplo: 2^-120 * 2^-100 = 2^-220. [4]**+infinity**: infinito positivo.

Fonte: Autor

O quadro 2 mostra 11 casos de testes que serão simulados para a Unidade Funcional de Multiplicação (UFM). Deve-se observar que todos os números hexadecimais estão em formato IEEE 754 32 bits com expoentes polarizados.

Os casos de teste projetados no quadro 2 servirão como bancada de testes para a simulação dos resultados para casos típicos de multiplicação, casos que resultam em operações inválidas, e os casos para simular *overflow* e *underflow*.

Uma bancada de teste é um arquivo em Verilog que não possui entradas ou saídas. A bancada de testes instancia o sistema a ser testado como um módulo de nível inferior. O sistema que está sendo testado é frequentemente chamado de dispositivo em teste (DUT) ou unidade em teste (UUT).

O código Verilog da bancada de teste para a simulação para a unidade funcional de multiplicação é mostrado nas figuras 10 e 11.

Figura 10 - Código da Bancada de Teste – UFM - Parte 1

```
// módulo de testbench para simulação da unidade de multiplicação em ponto flutuante de 32 bits
`timescale 1ns / 1ns

module mult_tb;
 reg [31:0] a,b;
 wire overflow,underflow,exception;
 wire [31:0] mult;

 mult_pf dut(a,b,mult,exception,underflow,overflow); // dut: design under test

initial
begin
  $dumpfile("upf32.vcd");
  $dumpvars;

  // Caso de Teste #1
  a = 32'h40C9_999A;
  b = 32'h7F80_0000;  // 6.3 * +infinity = exception
  #1;
  // resultado: 0x40C9_999A * +infinity = exception

  // Caso de Teste #2
  a = 32'h7B86_D0AA;
  b = 32'h7183_4436;
  // resultado: overflow
  #1;

  // Caso de Teste #3
  a = 32'h4234_851F;
  b = 32'h427C_851F; // 45.13 * 63.13 = 2849.0569
  #1;
  // resultado:0x4234_851F * 0x427C_851F
```

Fonte: Autor

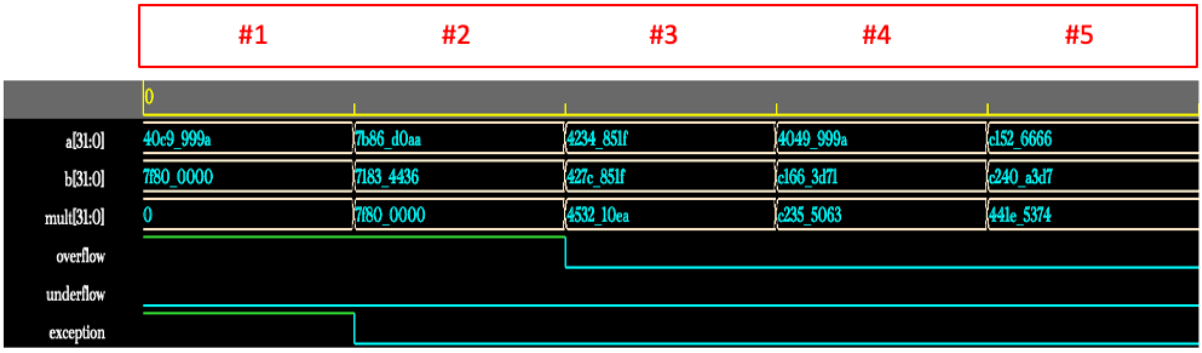
Figura 11 - Código da Bancada de Teste – UFM - Parte 2

```
    // Caso de Teste #4
    a = 32'h4049_999A;
    b = 32'hC166_3D71; // 3.15 * -14.39 = -45.3285
    // resultado: 0x4049_999A * 0xC166_3D71 = 0xC235_5063
    #1;

    // Caso de Teste #5
    a = 32'hC152_6666;
    b = 32'hC240_A3D7; // -13.15 * -48.16 = 633.304
    // resultado: 0xC152_6666 * 0xC240_A3D7 = 0x441E_5374
    #1;

    // Caso de Teste #6
    a = 32'h41C8_0000;
    b = 32'h4248_0000; // 25 * 50 = 1250
    // resultado: 0x41C8_0000 * 0x4248_0000 = 0x449C_4000
    #1;

    // Caso de Teste #7
    a = 32'h3ACA_62C1;
    b = 32'h3ACA_62C1; // 0.00154408081 * 0.00154408081 = 0.00000238418
    // resultado: 0x3ACA_62C1 * 0x3ACA_62C1 = 0x361F_FFFF
    #1;

    // Caso de Teste #8
    a = 32'h037F_3637;
    b = 32'h0D7D_1FDD; // +7.50000004534e-37 * +7.79999981785e-31
    // resultado: 0x037F_3637 * 0x0D7D_1FDD = underflow
    #1;

    // Caso de Teste #9
    a = 32'h0000_0000;
    b = 32'h0000_0000; // 0 * 0 = 0;
    // resultado: 0.0
    #1;

     // Caso de Teste #10
     a = 32'h7F80_0000;
     b = 32'h0000_0000; //+infinity * 0.0
     // resultado: +infinity * 0x0000_0000 = exception
     #1;

     // Caso de Teste #11
     a = 32'h7F80_0000;
     b = 32'h7F80_0000; // +infinity * +infinity
     // resultado: 0x7F80_0000 * 0x7F80_0000 = exception
     #1;

     $finish;
  end
  endmodule
```

Fonte: Autor

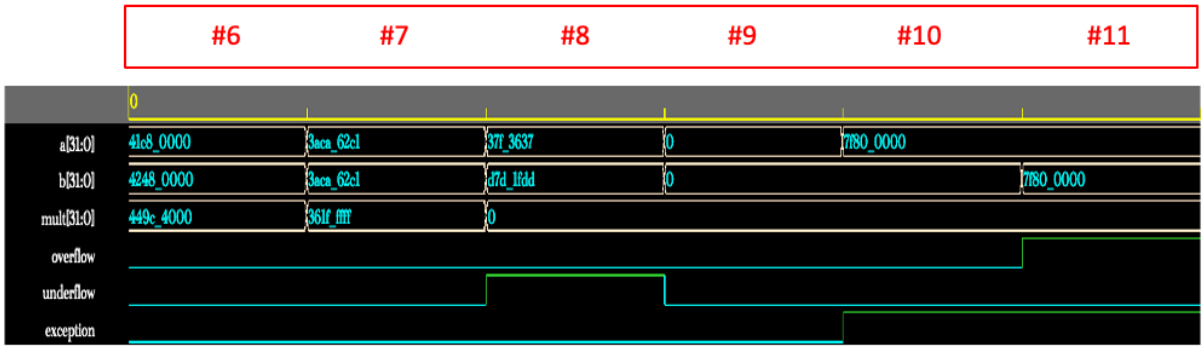
As figuras 12 e 13 mostram os resultados da simulação para a banca de testes descritos no quadro 2 para a UFM.

Figura 12 - Resultados da Simulação dos casos de teste – Parte 1



Fonte: Autor

Figura 13 - Resultados da Simulação dos casos de teste – Parte 2



Fonte: Autor

O tempo total de simulação foi de 11 ns, sendo simulado 1 ns para cada um dos testes. A simulação da UFM teve o comportamento esperado de seus resultados em todos os seus testes conforme previsto na bancada de testes do quadro 2, e como pode ser visto tanto no resultado para os casos típicos (#3 a #7 e #9), os casos de underflow (#8), e os casos de operações inválidas (#1, #10 e #11).

CONSIDERAÇÕES FINAIS

Este trabalho apresenta modelagem e simulação de uma UPF que executa as funções básicas com especial foco na operação de multiplicação. As atividades envolvidas na implementação são: a manipulação de dados de ponto flutuante, a conversão de dados para o formato IEEE 754, a execução de qualquer uma das operações aritméticas como adição, subtração, multiplicação.

A unidade funcional de multiplicação foi implementada em linguagem Verilog utilizando a plataforma EDA Playground. Vários casos de testes envolvendo situações que podem acontecer no resultado da operação tais como: *overflow, underflow* e exceções (NAN, infinito).

Os resultados apresentados pela Unidade funcional de multiplicação que a UPF de 32 bits implementada pelo método e código descritos funcionou adequadamente para os casos de testes gerados.

Como sugestão de trabalhos futuros a partir desta implementação são sugeridos: implementação de um algoritmo védico na operação de multiplicação e a implementação das operações de adição, subtração e divisão.

REFERÊNCIAS

ALTERA CORPORATION. **Verilog HDL Basis**. 2008. Disponível em <http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/Altera%20Tutorial%20-%20Verilog%20HDL%20Basic.pdf >. Acesso em: 03 mar. 2022.

CAVANAGH, Joseph. **Computer Arithmetic and Verilog HDL Fundamentals**. Boca Raton: CRC Press Taylor & Francis Group, 2010.

EDA Playground. Disponível em: https://edaplayground.com. Acesso em: 28 de mai. de 2022.

INTEL. **Intel Architecture Software Developer's Manual**. 1999. Volume 1: Basic Architecture. Disponível em <https://www.cs.cmu.edu/~410/doc/intel-arch.pdf>. Acesso em: 24 de mai. 2022.

LAMERES, Brock J. **Quick Start Guide to Verilog**. Cham, Switzerland: Springer

MALADKAR , Kishan; ARADHYA , Ravish. Design and Implementation of a 32-bit Floating Point Unit. **International Journal for Research in Applied Science & Engineering Technology (IJRASET)**, 2021, Vol. 9, Issue IV. e Published Online Dec. 2021 in RASET Journals. Disponível em < https://www.ijraset.com/fileserve.php?FID=35052 >. Acesso em: 02 de jun. 2022.

SAHU, L.; DEV, R. **An efficient IEEE 754 compliant floating point unit using Verilog**. 2012. A Thesis Submitted for The Partial Fulfilment of Requirements for Degree of Bachelor of Technology IN Computer Science and Engineering- Department of Computer Science and Engineering National Institute of Technology Rourkela Rourkela - 769008, India, 2012. Disponível em < http://ethesis.nitrkl.ac.in/3638/1/thesis_final.pdf >. Acesso em: 24 de mai. 2022.

SAVALIYA , Yagnesh; RUDANI , Jenish. Design and Simulation of 32-Bit Floating Point Arithmetic Logic Unit using Verilog HDL. **International Research Journal of Engineering and Technology (IRJET))**, 2020, Vol. 7, Issue 12. E-Published Online Dec. 2020 in IRJET Journals. Disponível em < https://www.irjet.net/archives/V7/i12/IRJET-V7I12262.pdf >. Acesso em: 25 de mai. 2022.

UPENDAR, S. Design and implementation of floating point Unit using VERILOG. **Journal of Advanced Research in Technology and Management Sciences**, 2018, Vol. 00, Issue 1. e Published Online Dec. 2018 in Artms Journals. Disponível em < http://jartms.org/view_issue.php?title=DESIGN-AND-IMPLEMENTATION-OF-FLOATING-POINT-UNIT-USING-VERILOG >. Acesso em: 20 de mai. 2022.

VIANA, G. V. R. **Padrão IEEE 754 para aritmética binária de ponto flutuante**. Universidade Estadual do Ceará-Departamento de Estatística e Computação. 2022.UFCE. (Apostila). Disponível em < https://www.lia.ufc.br/~valdisio/download/ieee.pdf >. Acesso em: 24 de mai. 2022.

XILINX. **Verilog Reference Guide**. 1999. Disponível em: http://in.ncu.edu.tw/ncume_ee/digilogi/vhdl/Verilog_Reference_Guide.pdf. Acesso em: 22 de jun. 2022.

ZIAULLAH, M.; MUNAFF, A. Design and Implementation of Floating Point ALU with Parity Generator Using Verilog HDL. **IOSR Journal of VLSI and Signal Processing**, 2015, Vol. 5, Issue 1, Ver. 1. e-ISSN: 2319 – 4200, p-ISSN No.: 2319 – 4197 Published Online Sep. 2015 in Iosr Journals. Disponível em < https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue5/Version-1/I05515459.pdf >. Acesso em: 19 de mar. 2022.

AUTORES:

ERIKC JOSÉ FERREIRA SANTOS, graduando do Curso de Engenharia Elétrica na Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail: erikc.1536059@discente.uemg.br.

MAURO HEMERLY GAZZANI, doutor em Engenharia Elétrica pela Universidade Federal de Uberlândia. Bacharel em Engenharia Elétrica pela Universidade Federal de Uberlândia. Professor do Curso de Graduação em Engenharia Elétrica da Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail : mauro.gazzani@uemg.br.

KÁTIA LOPES SILVA, Docteur en Sciences Appliquées pela Université de Liège. Bacharel em Engenharia Química pela Universidade Federal de Uberlândia. Professor do Curso de Graduação em Engenharia Elétrica da Universidade do Estado de Minas Gerais – UEMG, Unidade Ituiutaba. E-mail: katia.lopes@uemg.br.