PROJETO E IMPLEMENTAÇÃO DE UMA UNIDADE LÓGICA E ARITMÉTICA DE 32 BITS EM VERILOG

Design and Implementation of a 32-Bit Logical and Arithmetic Unit in Verilog

MAURO HEMERLY GAZZANI, KÁTIA LOPES SILVA,RAPHAEL BERNARDES BORBA, JOÃO VICTOR LEMOS

RESUMO

A ULA (Unidade Lógica Aritmética) é um bloco de construção fundamental da unidade central de processamento de um computador. Este trabalho envolve a implementação do projeto de uma ULA de 32 bits usando a linguagem Verilog. A ULA é projetada para realizar 11 operações que incluem lógicas e operações aritméticas, além das operações deslocamento e rotação de bits. Os resultados da simulação foram satisfatórios, conforme o esperado e comprovam que a modelagem e hipóteses adotadas estão coerentes com o comportamento das operações implementadas.

Palavras-chave: ULA 32 bits.Circuitos lógicos.HDL

ABSTRACT

The ULA (Arithmetic Logical Unit) is a fundamental building block of a computer's central processing unit. This work involves the design implementation of a 32-bit ALU using the Verilog language. The ALU is designed to perform 11 operations that include logic and arithmetic operations, in addition to shifting and rotating bit operations. The simulation results were satisfactory, as expected and prove that the modeling and assumptions adopted are consistent with the behavior of the implemented operations.

Keywords: *ALU 32 bits.Logic circuits.HDL*

INTRODUÇÃO

Projetos de sistemas digitais usando Linguagens de Descrição de Hardware (HDL-*Hardware DescriptionLanguage*) é um campo com potencial grande 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 HDL permite projetar um hardware digital por meio de software. Uma economia considerável de tempo é constatada ao projetar sistemas usando uma HDL. Isso oferece uma vantagem competitiva ao reduzir o tempo de colocação no mercado de um sistema, outra vantagem é que o projeto pode ser simulado e testado para operação funcional correta antes de implementar o sistema no hardware. Quaisquer erros encontrados durante a simulação podem ser corrigidos antes de confirmar desde o design até a implementação de hardware caro.

O código das HDL descreve o comportamento ou estrutura de um circuito eletrônico, a partir do qual um circuito físico compatível pode ser inferido por um compilador. Suas principais aplicações incluem a síntese de circuitos digitais em CPLD/FPGA (Dispositivos lógicos programáveis complexos / Matriz de portas programáveis em campo) chips e layout / geração de máscara para fabricação de ASIC (Circuito Integrado Específico de Aplicativo).

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 independente da plataforma de desenvolvimento (IDE – *Integrated Development Environment*) 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).

Uma Unidade Lógica Aritmética (ULA) representa o componente fundamental de um CPU do computador. Os processadores modernos contêm muitas ULAs poderosas e complexas. Além de ULAs, CPUs modernas contêm uma unidade de controle. A maioria das operações da CPU são realizadas por uma ou mais ULAs, que carregam os dados dos registradores de entrada. A unidade de controle informa a ULA a operação que será realizada e esta armazena o resultado em registrador de saída. A unidade de controle move os dados entre esses registradores, a ULA e a memória. (KANTAWALA, 2018) .

Uma ULA pode ser descrita como uma função digital de lógica combinatória com múltiplas operações. Ele pode realizar uma série de operações aritméticas básicas e um conjunto de operações lógicas. Ela tem uma série de linhas de seleção para selecionar uma determinada operação na unidade. (SARANGI et al.,2020).

Este trabalho apresenta modelagem e simulação das operações lógicas e aritméticas de uma ULA de 32 bits usando Verilog. O projeto lógico de uma ULA tem por objetivo desenvolver algoritmos adequados para alcançar uma utilização eficiente do hardware disponível. Utilizando o a linguagem Verilog, o código pode ser criado com base no objetivo do projeto.

MATERIAL E MÉTODOS

LaMeres (2019) é um autor que publicou um livro que apresenta uma introdução à linguagem Verilog para modelar, simular e sintetizar a lógica combinacional. O livro então cobre lógica sequencial e máquinas de estado finito no nível estrutural e sistemas síncronos sofisticados a serem modelados. Vários exemplos de modelagem de sistemas sequenciais e os detalhes de lógica programável, memória de semicondutor e circuitos aritméticos são apresentados. O livro culmina com uma discussão sobre design de sistema de computador. Cada componente de um sistema de computador é descrito com a implementação do Verilog, ao mesmo tempo em que reforça continuamente o hardware subjacente à abstração HDL

Atualmente, existem vários IDEs baseados em Verilog com FPGAs Altera e Xilinx que são suportados pelo Altera Quartus II e Xilinx ISE IDEs (XILINX, 2021).

Ratre e Mamta (2016) apresentaram um estudo de revisão sobre diferentes tipos de projeto ULAs existentes com uma análise comparativa em termos de área, potência, atraso e frequência. Além disso, implementaram um novo modelo para ULA de 32 bits usando o Verilog HDL na Ferramenta Xilinx 14.2 e a simulação do projeto foi realizada também no Modelsim.

Swamynathan e Banumathi (2017) propuseram uma ULA de 32 bits usando Verilog HDL com portas lógicas como AND e OR para cada circuito da ULA de um bit. O design foi implementado em Xilinx. A proposta de ULA pode funcionar mais rápido do que o processador ULA de mercado usando menos energia. O projeto de uma ULA e uma memória cache para uso em um processador de alto desempenho foi pesquisado. A lógica reversível tornou-se muito importante nos últimos anos porque tem a capacidade de reduzir a dissipação de energia, que é o principal requisito em projetos de baixa potência. As ULAs onde são projetados usando portas lógicas não reversíveis consomem mais energia. Portanto, há uma necessidade de menor consumo de energia e a lógica reversível tem desempenhado um papel vital nos últimos anos para as técnicas de design de VLSI de baixa energia. Essa técnica ajuda a reduzir o consumo de energia e a dissipação de energia. Desta forma, o artigo publicado apresenta uma implementação de ULA baseada em lógica reversível enquanto a compara a uma arquitetura de ULA com portas lógicas normais. Todos os módulos são simulados em Modelsim SE 6.4c e sintetizados usando Xilinx ISE 14.5. A ULA que é projetada usando portas lógicas não reversíveis consome mais energia de cerca de 0,312 MW e a implementação da ULA baseada em lógica reversível reduz o consumo de energia durante as operações para cerca de 5,1 por cento.

Kantawala (2018) propôs um novo modelo para ULA de 16 Bits que executa as operações aritméticas e lógicas. Esta ULA de 16 bits executa 16 operações para duas entradas de dados e uma seleção. De acordo com a seleção apropriada, a operação é realizada entre 2 entradas. Saída de estado da ULA de 16 bits está conectada entre ROM e RAM. Foi implementada no dispositivo SPARTAN-3E FPGA

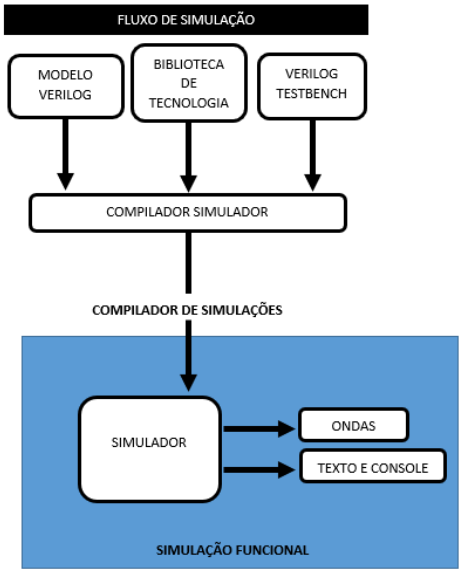
que possui a maior flexibilidade de design. O projeto é compacto e escalável sem qualquer mudança de material. A ferramenta de síntese otimiza o design FPGA arquitetura para ULA. Como resultado, recursos podem ser adicionados ao design existente sem qualquer mudança no equipamento.

A linguagem Verilog descreve um sistema digital como um conjunto de módulos, cada um com uma interface para outros módulos e a descrição de seu conteúdo.

Conforme LaMeres (2019), o padrão Verilog original (IEEE 1364) foi atualizado inúmeras vezes desde sua criação em 1995. A atualização mais significativa ocorreu em 2001, em que foi intitulado IEEE 1394-2001. Em 2005, pequenas correções e melhorias foram adicionadas ao padrão, que resultou em IEEE 1394-2005. As funcionalidades do Verilog (por exemplo, operadores, tipos de sinal, funções, etc.) são definidas no padrão Verilog; desta forma, não é necessário declarar explicitamente que um projeto está usando o Pacote IEEE 1394 pois é inerente ao uso de Verilog.

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

Figura 1- Visão geral da simulação em Verilog



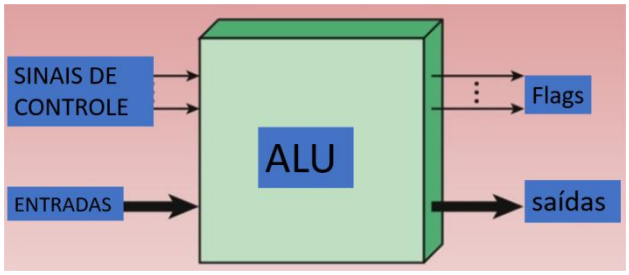
Fonte: Modificado de Altera Corporation (2008)

Os sistemas em Verilog são arquivados dentro de um módulo. Os módulos podem possuir variações de padrão de nível inferior para oferecer auxílio a projetos hierárquicos. O módulo de palavras-chave e módulo final significam o início e o fim da descrição do sistema.

> Um projeto Verilog descreve um único sistema em um único arquivo. O arquivo possui o sufixo *.v. Dentro do arquivo, a descrição do sistema está contida em um módulo. O módulo inclui a interface para o sistema(ou seja, as entradas e saídas) e a descrição do comportamento. A Figura 2.1 mostra uma representação gráfica de um arquivo Verilog. (LAMERES, 2019, p.17, tradução nossa)

Normalmente a ULA tem acesso direto de entrada e saída para o controlador do processador, memória principal (memória de acesso aleatório ou RAM em um computador pessoal) e dispositivos de entrada / saída. As entradas e saídas fluem ao longo de um caminho eletrônico chamado de barramento. A entrada consiste em uma palavra de instrução (às vezes chamada de palavra de instrução de máquina) que contém um código de operação (às vezes chamado de "código op"), um ou mais operandos e às vezes um código de formato. O código de operação informa à ULA qual operação realizar e os operandos usados na operação. (Por exemplo, dois operandos podem ser somados ou comparados logicamente.) O formato pode ser combinado com o “código op” e informa, por exemplo, se esta é uma instrução de ponto fixo ou de ponto flutuante. A saída consiste em um resultado que é colocado em um registro de armazenamento e configurações que indicam se a operação foi executada com sucesso. (Se não for, algum tipo de status será armazenado em um local permanente que às vezes é chamado de palavra de status da máquina.).A figura 2 mostra uma visão geral da ULA.

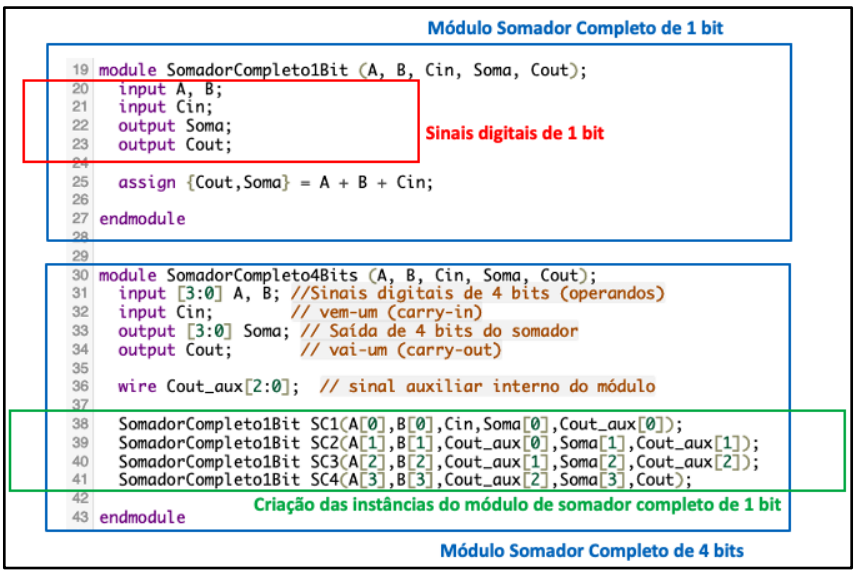
Figura 2- Visão geral da ULA



Fonte: Borba & Lemos, 2021

Todo programa escrito em Verilog deve possuir pelo menos um módulo que implementa um circuito digital. A Figura 3 mostra a anatomia de um módulo Verilog que implementa o projeto de um somador completo de 4 bits, constituído de 4 instâncias de um módulo de somador completo de 1 bit.

Figura 3 - Estrutura de um programa em Verilog

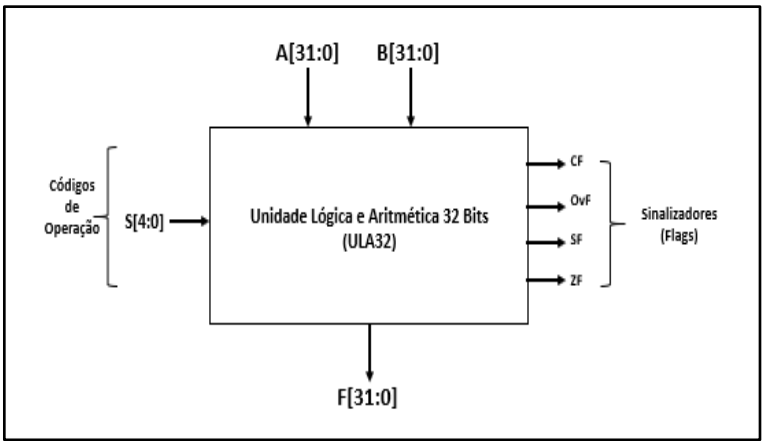



```
19 module SomadorCompleto1Bit (A, B, Cin, Soma, Cout);
20   input A, B;
21   input Cin;
22   output Soma;
23   output Cout;
24
25   assign {Cout,Soma} = A + B + Cin;
26
27 endmodule
28
29
30 module SomadorCompleto4Bits (A, B, Cin, Soma, Cout);
31   input [3:0] A, B; //Sinais digitais de 4 bits (operandos)
32   input Cin;        // vem-um (carry-in)
33   output [3:0] Soma; // Saída de 4 bits do somador
34   output Cout;       // vai-um (carry-out)
35
36   wire Cout_aux[2:0];  // sinal auxiliar interno do módulo
37
38   SomadorCompleto1Bit SC1(A[0],B[0],Cin,Soma[0],Cout_aux[0]);
39   SomadorCompleto1Bit SC2(A[1],B[1],Cout_aux[0],Soma[1],Cout_aux[1]);
40   SomadorCompleto1Bit SC3(A[2],B[2],Cout_aux[1],Soma[2],Cout_aux[2]);
41   SomadorCompleto1Bit SC4(A[3],B[3],Cout_aux[2],Soma[3],Cout);
42
43 endmodule
```

Fonte: Autores

A Figura 4 ilustra a visão geral de uma ULA de 32 bits, onde as entradas A e B de 32 bits representam os operandos, e a entrada de 5 bits representa os códigos de operação. Destes 5 bits, os dois bits mais superiores (S4 e S3) identificam a unidade funcional da ULA, e os demais bits (S2 a S0), identificam a operação a ser realizada pela unidade funcional identificada pelos bits S4 e S3. O sinal de saída F representa o resultado da última operação realizada pela ULA. Os outros 4 bits de saída sinalizam o estado do resultado da ULA.

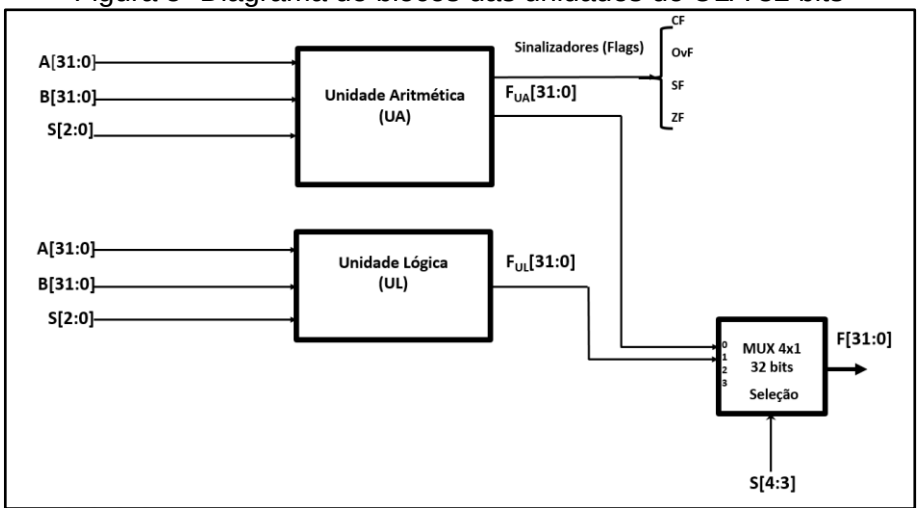
Figura 4- Visão geral de uma ULA de 32 bits



Fonte: Borba & Lemos, 2021

A ULA de 32 bits é modularizada em unidade funcionais e ilustrada na figura 5. As 3 unidades funcionais são denominadas como seguem: Unidade Aritmética (UA), e a Unidade Lógica (UL). As 2 entradas de 32 bits representam os operandos das respectivas unidades funcionais, e os 3 bits de entrada (S2 a S0) identificam a operação a ser realizada. Cada uma das unidades funcionais possui uma saída de 32 bits, que representa o resultado da operação realizada, sendo que a UA possui uma saída a mais que representa os sinalizadores (flags) que sinalizam o estado do resultado desta unidade. O MUX 4:1 utiliza os bits S4 a S3 para selecionar a saída de qual unidade será aquela da ULA.

Figura 5- Diagrama de blocos das unidades do ULA 32 bits



Fonte: Borba & Lemos, 2021

A tabela 1 elenca as operações que a ULA de 32 bits poderá realizar, de acordo com os 5 bits do código de operação (S4 a S0). A tabela relaciona a operação a ser realizada com o código de operação já descrito anteriormente.

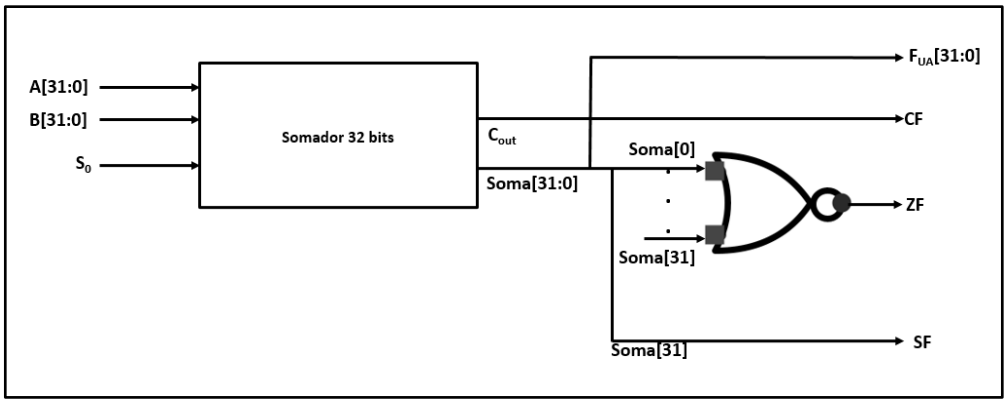
Tabela 1- Tabela das Funções da ULA 32 bits

Código de Operação					Saída	Função	Unidade Funcional
S_4	S_3	S_2	S_1	S_0			
0	0	0	0	0	$F = A$	Transfere A	Aritmética
0	0	0	0	1	$F = A + 1$	Incrementa A	
0	0	0	1	0	$F = A + B$	Adição	
0	0	0	1	1	$F = A + B + 1$	Adição com Vai-um	
0	0	1	0	0	$F = A - B - 1$	Subtração com Vem-um	
0	0	1	0	1	$F = A - B$	Subtração	
0	0	1	1	0	$F = A - 1$	Decrementa A	
0	1	0	0	0	$F = A \vee B$	Ou (Or)	Lógica
0	1	0	0	1	$F = A \oplus B$	Ou-exclusivo (Xor)	
0	1	0	1	0	$F = A \wedge B$	E (And)	
0	1	0	1	1	$F = \bar{A}$	Não (Not)	

Fonte: Borba & Lemos, 2021

A figura 6mostra a Unidade Aritmética. Neste caso tem-se os operandos de 32 bits e S0 como o bit de vem-um (*carry*-in) como suas entradas, e suas saídas, representando os 32 bits Soma como o resultado da operação realizada por esta unidade, e o C_{out} como o bit de vai-um (*carry*-out) do resultado. Os outros 4 bits de saída são os sinalizadores que representam o estado do resultado da saída da unidade. Além disso, a figura 11 mostra como cada um dos sinalizadores são determinados, sendo que a descrição de cada um deles será explicado mais adiante.

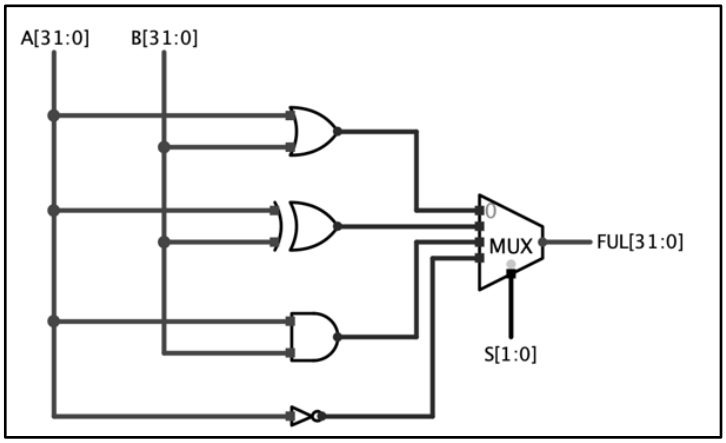
Figura 6 - Diagrama da Unidade Aritmética (UA) de 32 Bits



Fonte: Borba & Lemos, 2021

A Unidade Lógica (UL) é mostrada na figura 7com os dois operandos de 32 bits como suas entradas e os 2 bits S1 e S0 como as entradas de seleção do multiplexador (MUX) que determina a operação lógica que será disponibilizada na saída da unidade, conforme a tabela 3 que destaca as operações lógicas disponíveis neste projeto.

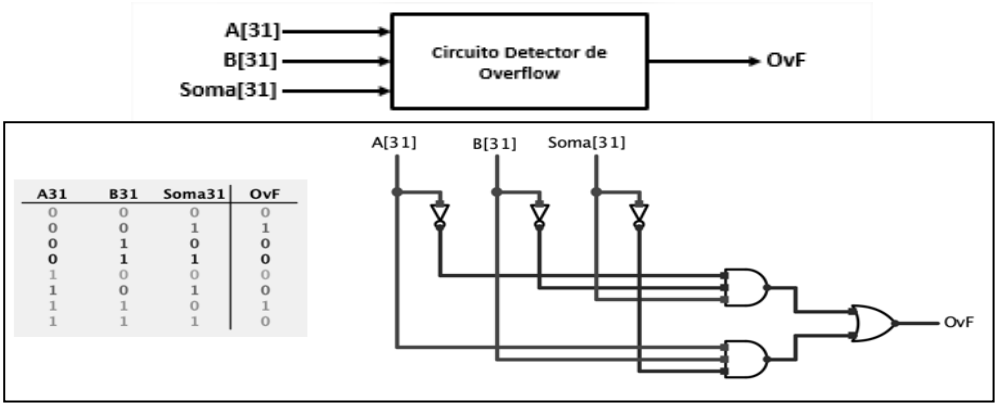
Figura 7- Circuito da Unidade Lógica (UL)



Fonte: Borba & Lemos, 2021

A figura 8 ilustra também o diagrama do circuito detector de *overflow* projetado de acordo com a tabela verdade correspondente.O projeto do circuito detector de overflow baseia-se na análise dos sinais de operandos e do resultado da operação realizada. A tabela verdade e o circuito mostrados na figura 8 relacionam os bits dos sinais (mais significativos) de A, B e a Soma. Quando os operandos possuem sinais distintos (o primeiro operando positivo e o segundo negativo, ou vice-versa) a saída do resultado está correta. Por outro lado, se os operandos possuem sinais iguais (o primeiro e o segundo operando positivos, ou ambos negativo) existe a possibilidade que a saída do resultado esteja incorreta. Para este último cenário, se o sinal da saída do resultado diferir daquele dos operandos então pode-se concluir que o resultado está incorreto. Em resumo, o bit da saída do circuito detector de overflow será 0 caso o resultado esteja correto, e igual a 1, caso o resultado esteja incorreto.

Figura 8- Diagrama do Circuito detector de *overflow* de 32 bits



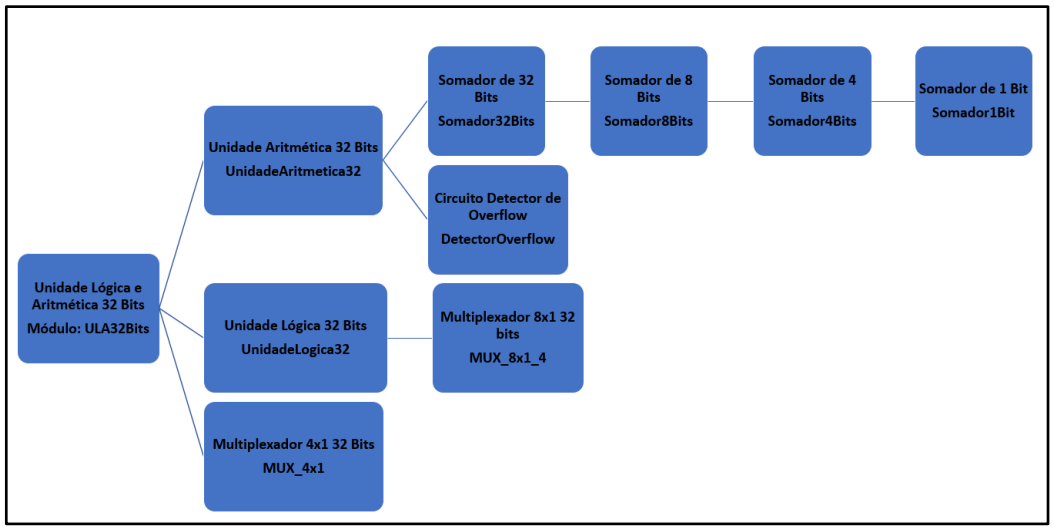
Fonte: Borba & Lemos, 2021

A implementação em linguagem Verilog pode ser realizada com vários IDEs baseados em Verilog com FPGAs Altera e Xilinx que são suportados pelo Altera Quartus II e Xilinx ISE IDEs disponíveis no mercado. No caso deste trabalho foi utilizado o EDA Playground (2021).

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 System Verilog, Verilog, VHDL, C ++ / System C 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 *TestBench*.

A implementação da ULA 32 bits está organizada em dez módulos Verilog conforme o diagrama Hierárquico dos Módulos Verilog da ULA de 32 Bits mostrado na Figura 9. São eles: ULA32Bits é o módulo que contém a ULA; UnidadeAritmetica32, UnidadeLógica32, MUX 4x1: que contém respectivamente os módulos Unidade Aritmética (UA), Unidade Lógica (UL) e o Multiplexador 4:1; Detector Overflow, MUX_8x1_4: que contém os módulos Circuito Detector de Over Flow e o Multiplexador 8x1 de 32 Bits; Somador32Bits, Somador8Bits, Somador4Bits, Somador1Bits: que contém os módulos Somador de 32 Bits, Somador de 8 Bits, Somador de 4 Bits e Somador de 1Bit.

Figura 9- Diagrama Hierárquico dos Módulos Verilog da ULA de 32 Bits



Fonte: Borba & Lemos, 2021

Os códigos dos módulos em Verilog descritos na figura 9 são mostrados nas figuras 10e 11 para Unidade Lógica e Aritmética 32 Bits.

Figura 10- Código Verilog da Unidade Lógica e Aritmética 32 Bits – Parte 1

```
7  // Unidade Lógica e Aritmética de 32 bits
8
9  module ULA32Bits(A,B,S,OvF,SF,ZF,CF,F);
10   input [31:0] A,B; // Operandos da ULA
11   input [4:0] S;  // Códigos de operação: S4 S3 S2 S1 S0
12   output OvF,SF,ZF,CF;  // sinalizadores (Flags)
13   output [31:0] F; // saída de 32 bits da ULA
14
15   wire [31:0] F_UA,F_UL;
16   wire [31:0] F2,F1;
17
18
19   // Cria instância da Unidade Aritmética de 32 Bits
20   UnidadeAritmetica32 UA32(A,B,S(2:0),Cin,F_UA,OvF,SF,ZF,CF);
21
22   // Cria instância da Unidade Lógica de 32 Bits
23   UnidadeLogica32 UL32(A,B,S(2:0),F_UL);
24
25   // Multiplexador seleciona qual unidade será a saída da ULA
26   MUX4x1 MUX41(S(4:3),F2,F1,F_UA,F_UL,F);
27
28   endmodule
```

Fonte: Borba & Lemos, 2021

Figura 11- Código Verilog da Unidade Lógica e Aritmética 32 Bits – Parte 2

```
32 // Multiplexador 4:1 32 bits
33 module MUX_4x1_4(I3,I2,I1,I0,S,Saida);
34   // 4 entradas de 32 bits cada
35   input [31:0] I3,I2,I1,I0;
36
37   // 2 bits de seleção
38   input [1:0] S;
39
40   // Saída do MUX
41   output reg [31:0] Saida;
42
43
44   always @(S)
45    begin
46      case(S)
47          3'b000: Saida = I0;
48          3'b001: Saida = I1;
49          3'b010: Saida = I2;
50          3'b011: Saida = I3;
51        default: Saida = 3'b000;
52      endcase
53    end
54 endmodule
```

Fonte: Borba & Lemos, 2021

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 também verificar automaticamente se as saídas estão corretas.

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).

A tabela 2 mostra os casos de testes que serão simulados para a unidade aritmética.

Tabela 2 - Casos de testes da Unidade Aritmética (UA)

Caso de teste	Tipo	Entradas	Saída esperada
Transfere A	Teste de Funções F = A	S2 = 0 S1 = 0 S0 = 0 A = 0xABCDABCD B = 0x11112345	F_UA = 0xABCDABCD
Incrementa A	Teste de Funções F = A+1	S2 = 0 S1 = 0 S0 = 1 A = 0xABCDABCD B = 0x11112345	F_UA = 0xABCDABCE
Adição	Teste de Funções F = A+B	S2 = 0 S1 = 1 S0 = 0 A = 0xABCDABCD B = 0x11112345	F_UA = 0xBCDECF12
Adição comVai-um	Teste de Funções F = A+B+1	S2 = 0 S1 = 1 S0 = 1 A = 0xABCDABCD B = 0x11112345	F_UA = 0xBCDECF13
Subtração comVem-um	Teste de Funções F = A-B-1	S2 = 1 S1 = 0 S0 = 0 A = 0xABCDABCD B = 0x11112345	F_UA = 0X9ABC8887
Subtração	Teste de Funções F = A-B	S2 = 1 S1 = 0 S0 = 1 A = 0xABCDABCD B = 0x11112345	F_UA = 0X9ABC8888
Decrementa A	Teste de Funções F = A-1	S2 = 1 S1 = 1 S0 = 0 A = 0xABCDABCD B = 0x11112345	F_UA = 0xABCDABCC

Fonte: Borba & Lemos, 2021

A tabela 3 mostra os casos de testes que serão simulados para a unidade Lógica.

Tabela 3- Casos de testes da Unidade Lógica (UL)

Caso de teste	Tipo	Entradas	Saída esperada
Ou (Or)	Teste de Funções F = A or B	S2 = 0 S1 = 0 S0 = 0 A = 0xABCDABCD B = 0x1FFAFFA1	F_UL = 0xBFFFFED
Ou-exclusivo (Xor)	Teste de Funções F = A xor B	S2 = 0 S1 = 0 S0 = 1 A = 0xABCDABCD B = 0x1FFAFFA1	F_UL = 0xB437546C
E (And)	Teste de Funções F = A and B	S2 = 0 S1 = 1 S0 = 0 A = 0xABCDABCD B = 0x1FFAFFA1	F_UL = 0x0BC8AB81
Não (Not)	Teste de Funções F = not(A)	S2 = 0 S1 = 1 S0 = 1 A = 0xABCDABCD B = 0x1FFAFFA1	F_UL = 0x54325432

Fonte: Borba & Lemos, 2021

A Tabela 4mostra os casos de testes para os *flags* da unidade aritmética.

Tabela 4- Casos de Testes para os *Flags*

Caso de teste	Tipo	Entradas	Saídas esperadas
Flag de Carry CF = 0	Teste de Flags F = A + B	S2 = 0 S1= 1 S0 = 0 A = 0xA115FBB4 B = 0x1000AFF4	F_UA = 0xB116ABA8 CF = 0
Flag de Carry CF= 1	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0xA115FBB4 B = 0x7000AFF4	F_UA = 0x1116ABA8 CF = 1
Flag de Overflow OvF = 0	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0xA115FBB4 B = 0x1000AFF4	F_UA = 0xB116ABA8 OvF = 0
Flag de Overflow OvF = 1	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0x9115FBB4 B = 0x8000AFF4	F_UA = 0x1116AFA8 OvF = 1
Flag de Sinal SF =0	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0	F_UA = 0x2116ABA8 SF = 0

Caso de teste	Tipo	Entradas	Saídas esperadas
		A = 0xA115FBB4 B = 0x8000AFF4	
Flag de Sinal **SF =1**	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0xA115FBB4 B = 0x1000AFF4	F_UA = 0xB116ABA8 SF = 1
Flag de Zero **ZF = 0**	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0xA115FBB4 B = 0x1000AFF4	F_UA = 0xB116ABA8 ZF = 0
Flag de Zero **ZF = 1**	Teste de Flags F = A + B	S2 = 0 S1 = 1 S0 = 0 A = 0xA115FBB4 B = 0x5EEA044C	F_UA = 0x00000000 ZF = 1

Fonte: Borba & Lemos, 2021

As bancadas de teste para a simulação para as unidades aritmética são mostradas na figura 12 e 13.

Figura 12- Código dos *Testbenches* – Unidade Aritmética – Funções – Parte 1

```
4  `timescale 1ns/1ps
5
6  // Test bench para a Unidade Aritmética de 32 bits
7  module UAritmetica32_tb();
8    reg [31:0] A,B;
9    reg [2:0] S;
10   wire [31:0] F_UA;
11   wire CF, ZF,SF,OvF;
12
13
14   // Cria instância da Unidade Aritmética de 32 bits
15   UAritmetica32 ua32(A,B,S,CF,ZF,SF,OvF,F_UA);
16
17   initial
18     begin
19       $dumpfile("ua32.vcd");
20       $dumpvars;
21
22       // 1. F = A
23       S = 3'b000; // S2S1S0
24       A = 32'hABCDABCD;
25       B = 32'h11112345; #1
26
27       // 2. F = A + 1
28       S = 3'b001; // S2S1S0
29       A = 32'hABCDABCD;
30       B = 32'h11112345; #1;
```

Fonte: Borba & Lemos, 2021

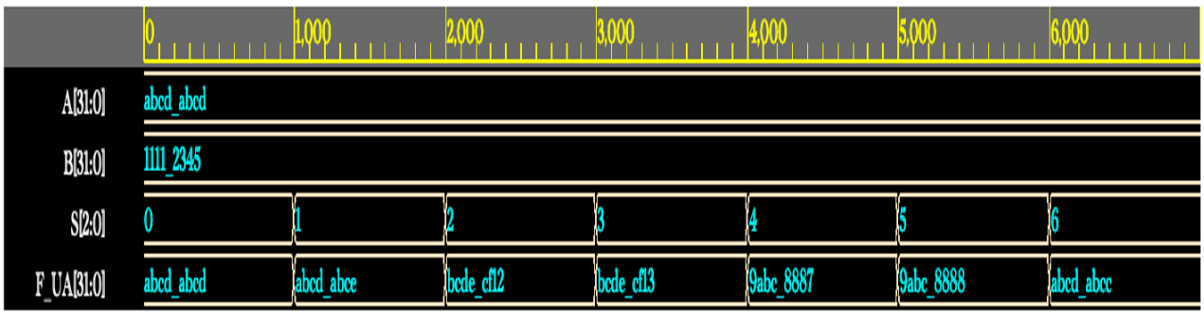
Figura 13 - Código dos *Testbenches* – Unidade Aritmética – Funções – Parte 2

```
32        // 3. F = A + B
33        S = 3'b010; // S2S1S0
34        A = 32'hABCDABCD;
35        B = 32'h11112345; #1;
36
37        // 4. F = A + B + 1
38        S = 3'b011; // S2S1S0
39        A = 32'hABCDABCD;
40        B = 32'h11112345; #1;
41
42        // 5. F = A - B - 1
43        S = 3'b100; // S2S1S0
44        A = 32'hABCDABCD;
45        B = 32'h11112345; #1;
46
47        // 6. F = A - B
48        S = 3'b101; // S2S1S0
49        A = 32'hABCDABCD;
50        B = 32'h11112345; #1;
51
52         // 7. F = A - 1
53        S = 3'b110; // S2S1S0
54        A = 32'hABCDABCD;
55        B = 32'h11112345; #1;
56
57        $finish;
58
59    end;
60 endmodule
```

Fonte: Borba & Lemos, 2021

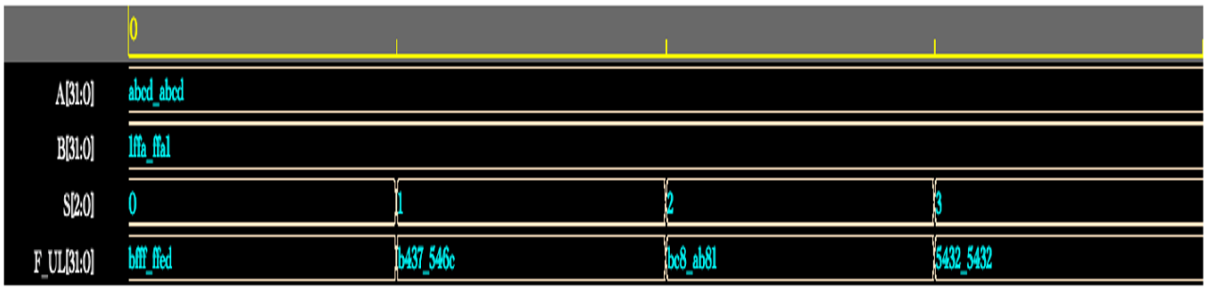
A figura 14 mostra os resultados da simulação para os casos de testes descritos na tabela 4 para a Unidade Aritmética.

Figura 14 - Resultado da Simulação para os casos de testes da Unidade Aritmética
Fonte: Borba & Lemos, 2021



A figura 15 mostra os resultados da simulação para os casos de testes descritos na tabela 5 para a Unidade Lógica.

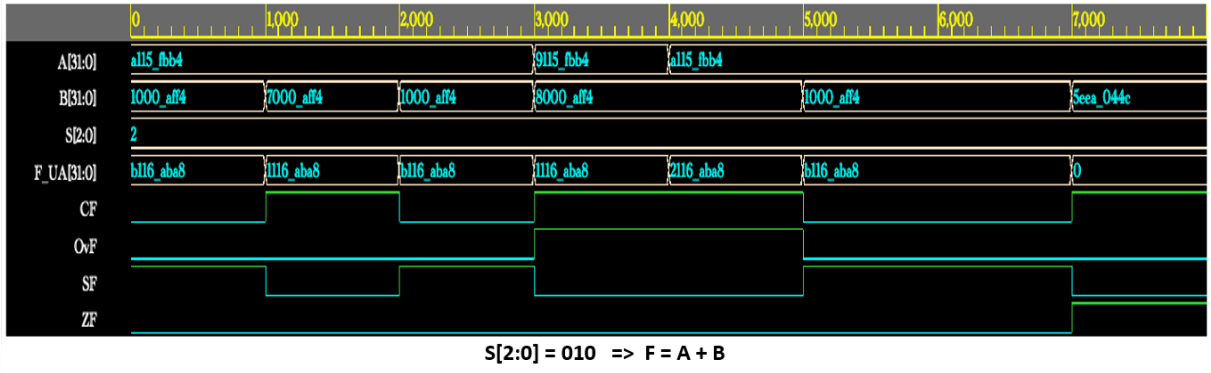
Figura 15 - Resultado da Simulação para os casos de testes da Unidade Lógica



Fonte: Borba & Lemos, 2021

A figura 16 mostra os resultados da simulação para os casos de testes descritos na tabela 6 para os *flags*.

Figura 16- Resultado da Simulação para os casos de testes dos *Flags*



Fonte: Borba & Lemos, 2021

CONSIDERAÇÕES FINAIS

Neste trabalho realizou-se a implementação do projeto de uma ULA 32 bits usando a linguagem Verilog. Primeiramente elaborou-se uma revisão geral da linguagem Verilog, sua sintaxe, organização de código, tipos de dados, e características gerais sobre a linguagem, mostrando por meio de simulações com as operações lógicas clássicas, abordados no capítulo 3. No capítulo 4, foi realizado a revisão teórica da ULA, modelagem, projeto e implementação em Verilog; os resultados foram gerados gráficos em formas de ondas por meio do *testbenches* no capítulo 5.

A ULA pode realizar 7 operações aritméticas, e 4 lógicas. A implementação em linguagem Verilog utilizou a ferramenta EDA Playground, que é uma ferramenta possui um ambiente online para simular e sintetizar implementações nas linguagens VHDL e outras. Considerando os resultados da simulação pode-se observar que a

ULA de 32 bits implementada pelo método e código descritos apresentou um desempenho adequado para todas as combinações de entrada e os códigos de operação.

Na implementação em linguagem VHDL foi utilizado a plataforma EDA Playground, que possui um ambiente online para fazer as simulações e resumir implementações nas linguagens Verilog, System Verilog, VHDL, C++/System C e outros HDLs. Os resultados das simulações podem ser visualizados na forma de ondas usando o visualizador de ondas baseado em navegadores EPWave e por meio de definições de casos de teste no *TestBench*.

Observou-se a partir dos resultados da simulação que a ULA de 32 bits implementada pelo método e código descritos funcionou adequadamente para todas as combinações de entrada de operações.

Como trabalhos a serem desenvolvidos a partir deste, podem-se sugerir os seguintes:Implementar a unidade de rotação e deslocamento de bits;Implementar a unidade de multiplicação e divisão de inteiros não sinalizados e sinalizados;Implementar a etapa final com ferramentas de sintetização do código da ULA em dispositivo lógico programável como por exemplo FPGA;Considerar o estudo e projeto de uma unidade de ponto flutuante para operações numéricas em números fracionários;Abordar a implementação da unidade lógica e aritmética por meio de portas lógicas reversíveis para fins de redução no consumo de energia. Nos últimos anos, os circuitos lógicos reversíveis ganharam um interesse notável à luz dos avanços feitos na computação quântica.

REFERÊNCIAS

BORBA, R. B.; LEMOS, J. V. **Projeto e implementação de uma unidade lógica e aritmética de 32 bits em Verilog**. 2021. Monografia (Bacharelado em Engenharia Elétrica), Universidade do Estado de Minas Gerais – UEMG, Campus Ituiutaba, 2021.

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 ago. 2021.

KANTAWALA, M. Design and implementation of 8 bit and 16 bitalu using Verilog language. **International Journal of Engineering Applied Sciences and**

Technology, 2018, Vol. 3, Issue 2, ISSN No. 2455-2143, Pages 30-34, Published Online June 2018 in IJEAST. Disponível em:http://www.ijeast.com/papers/30-34,TESMA302,IJEAST.pdf. Acesso em: 19 jun. 2021.

LAMERES, Brock J. **Quick Start Guide to Verilog**. Cham, Switzerland: Springer
RATRE, Mamta; SINGH, Jitendra. A Review Paper on Arithmetic and Logical Unit for Graphics Processor. **International Journal of Emerging Trends in Science and Technology**. Impact Factor: 2.838 DOI: http://dx.doi.org/10.18535/ijetst/v3i02.08. IJETST- Vol.||03||Issue||02||Pages 3541-3547||February||ISSN 2348-9480. 2016. Disponívelem: https://ieeexplore.ieee.org/document/8191959. Acessoem: 06 jun. 2021.

SARANGI, Saumyakanta et al. VHDL Implementation of Arithmetic Logic Unit. **International Journal of Engineering Research & Technology (IJERT).** Vol. 3 Issue 4, April – 2014. ISSN: 2278-0181. Disponívelem: ijert.org/vhdl-implementation-of-arithmetic-logic-unit. Acesso em: 22 mai. 2021.

SWAMYNATHAN, S. M.; BANUMATHI, V. Design and analysis of FPGA based 32 bit ALU using reversible gates. **2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE).** Date of Conference: 27-28 April 2017. ISBN Information:INSPEC Accession Number: 17431013DOI: 10.1109/ICEICE.2017.8191959. Publisher: IEEE.ConferenceLocation: Karur, India. Disponível em: https://ieeexplore.ieee.org/document/8191959. Acesso em: 22 jun. 2021.

XILINX. **Verilog Reference Guide**. 1999. Disponível em: http: in.ncu.edu.tw / ncume_ee/digilogi/vhdl/Verilog_Reference_Guide.pdf. Acesso em: 22 djun. 2021.

AUTORES

RAPHAEL BERNARDES BORBA, graduando do Curso de Engenharia Elétrica na Universidade do Estado de Minas Gerais –UEMG, Unidade Ituiutaba. E-mail: raphael.1554408@discente.uemg.br .

JOÃO VICTOR LEMOS, graduando do Curso de Engenharia Elétrica na Universidade do Estado de Minas Gerais –UEMG, Unidade Ituiutaba. E-mail: joao.1554394@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 Universidadedo Estado de Minas Gerais –UEMG, Unidade Ituiutaba. E-mail:maur.gazzani@uemg.br .

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