Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Relatório de Estágio Supervisionado

Laboratório de Excelência em Microeletrônica do Nordeste (XMEN)

Heriberto Gomes da Fonseca Junior

Campina Grande - PB

Junho de 2023

Heriberto Gomes da Fonseca Junior

Laboratório de Excelência em Microeletrônica do Nordeste (XMEN)

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Marcos Ricardo Alcântara Morais, D.Sc.

(Orientador)

Campina Grande - PB

Junho de 2023

Heriberto Gomes da Fonseca Junior

Laboratório de Excelência em Microeletrônica do Nordeste (XMEN)

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovado em ____ /____ /____

Gutemberg Gonçalves dos Santos Júnior
Universidade Federal de Campina Grande
Avaliador

Marcos Ricardo Alcântara Morais
Universidade Federal de Campina Grande
Orientador

Campina Grande - PB
Junho de 2023

Dedico este trabalho a Deus e a minha família, especialmente a minha mãe Andréa, meu pai Heriberto, minha irmã Gabriela e minha noiva Ester, por todo o apoio, amor e incentivo ao longo de todos esses anos. Certamente vocês são o motivo de eu chegar até aqui.

Agradecimentos

Ao Senhor Deus, autor da fé e criador de todas as coisas, por me dar forças e me sustentar em todas as tribulações e intempéries vividas durante a minha vida.

A minha mãe Andréa, meu pai Heriberto e minha irmã Gabriela que fizeram com que esse sonho fosse possível e estiveram presente durante todas as etapas da minha formação.

A minha noiva e eterna companheira Ester, que sempre acreditou em mim e me amou em todos os momentos, decidiu viver um relacionamento a distância e superou esse desafio ao meu lado, fazendo com que todos esses anos fossem mais felizes e divertidos.

Aos amigos e irmãos que fiz durante a vida, que dividiram momentos alegres e tristes comigo. Em especial Alan Pessoa, Arthur Macena, Ezequiel Brito, Hélder Guimarães, Lucas Dechenier, Lourival Netto e Matheus Petrúcio que dividiram os anos de estudo e formação comigo, bem como suas dificuldades e barreiras. Espero que todos vocês estejam sempre comigo na jornada da vida e que possamos compartilhar muitos momentos.

A todos que fazem parte do laboratório XMEN em especial a Guilherme Martins e André Igor, por me incentivarem a realizar as atividades (inclusive esse trabalho) dentro do prazo e compartilharem conhecimentos em microeletrônica e boas risadas.

Aos professores que tive durante a minha formação, em especial os professores Gutemberg Júnior e Marcos Morais, por todas as portas abertas, pelos anos de ensinamentos e por terem sido meus tutores para minha carreira profissional.

Enfim, agradeço a todos que cruzaram meu caminho nessa graduação e que tive o prazer de ajudar e ser ajudado.

"Tudo o que fizerem, façam de todo o coração, como para o Senhor, e não para os homens, sabendo que receberão do Senhor a recompensa da herança. É a Cristo, o Senhor, que vocês estão servindo."

Colossenses 3:23-24

Resumo

No presente relatório estão descritas as principais atividades desempenhadas pelo aluno Heriberto Gomes da Fonseca Junior, do curso de graduação em Engenharia Elétrica, durante o período de estágio no "Laboratório de Excelência em Microeletrônica do Nordeste" (X-MEN). No decorrer deste, será explanada a elaboração de uma plataforma *web* chamada *XSME* para o ensino de microeletrônica digital para alunos da graduação. A parte designada ao discente foi a elaboração dos exercícios, seus modelos e seus *testbenchs* auto-verificáveis.

Palavras-chave: Desenvolvimento de *hardware*, Microeletrônica, ASIC, *design* digital, *testbench*.

Abstract

This report describes the main activities carried out by student Heriberto Gomes da Fonseca Junior, from the undergraduate course in Electrical Engineering, during the internship period at the "Laboratório de Excelência em Microeletrônica do Nordeste" (X-MEN). Throughout this report, the development of a web platform called XSME for teaching digital microelectronics to undergraduate students will be explained. The student's task was to create exercises, their templates, and self-verifiable testbenches.

Keywords: Development of hardware, Microelectronics, ASIC, digital design, testbench.

Lista de ilustrações

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
UFCG	Universidade Federal de Campina Grande
XMEN	Laboratório de Excelência em Microeletrônica do Nordeste
ASIC	*Application-Specific Integrated Circuit*
PEM	*Programa de Excelência em Microeletrônica*
P&D	*Pesquisa e Desenvolvimento*
SoC	*System-On-Chip*
XSME	*Xavier School for Microelectronics*

Sumário

1 Introdução

O presente trabalho faz referência ao estágio curricular desenvolvido pelo aluno do curso de Engenharia Elétrica da Universidade Federal de Campina Grande (UFCG), Heriberto Gomes da Fonseca Junior no Laboratório de Excelência em Microeletrônica do Nordeste (*X-MEN*), de 21 de março de 2023 a 30 de maio de 2023, totalizando uma carga horária de 304 horas.

Durante o período de estágio o aluno participou de cursos disponibilizados pela Cadence© Design Systems, tais como: "*SystemVerilog for Design and Verification Engineers*" e "*SystemVerilog Accelerated Verification with UVM*". Além disso, o aluno participou do desenvolvimento da plataforma digital *XSME* que é uma plataforma de aprendizado de *design* digital de microeletrônica e *SystemVerilog*. Ela possui conteúdo didático sobre circuitos básicos, lógica combinacional e sequencial, seguido por exercícios para fixar o conhecimento adquirido. O nome é uma referência ao filme X-Men e ao laboratório *XMEN*. O discente ficou encarregado da criação das atividades, concebendo modelos de referência e testes auto-verificáveis.

1.1 Laboratório de Excelência em Microeletrônica do Nordeste (*XMEN*)

Em 2016, pesquisadores da UFCG estabeleceram uma iniciativa com o objetivo de criar um modelo de formação de recursos humanos, pesquisa, desenvolvimento e inovação na área de microeletrônica, previamente conhecido como PEM (Programa de Excelência em Microeletrônica).

Atualmente, o laboratório possui a participação de 15 alunos de graduação. Os desenvolvedores do laboratório são divididos em três equipes que trabalham no desenvolvimento de um chip. A primeira equipe, chamada de *frontend*, é responsável pela construção lógica e arquitetural do circuito a ser projetado. A segunda equipe, denominada verificação, garante a correta funcionalidade do circuito projetado. Por fim, a equipe de *backend* realiza a construção física do circuito projetado.

Além da divisão entre as equipes de desenvolvimento, também existem diferentes frentes de trabalho, onde os desenvolvedores se dedicam a projetos de P&D, projetos internos do laboratório e projetos em parceria com outras instituições.

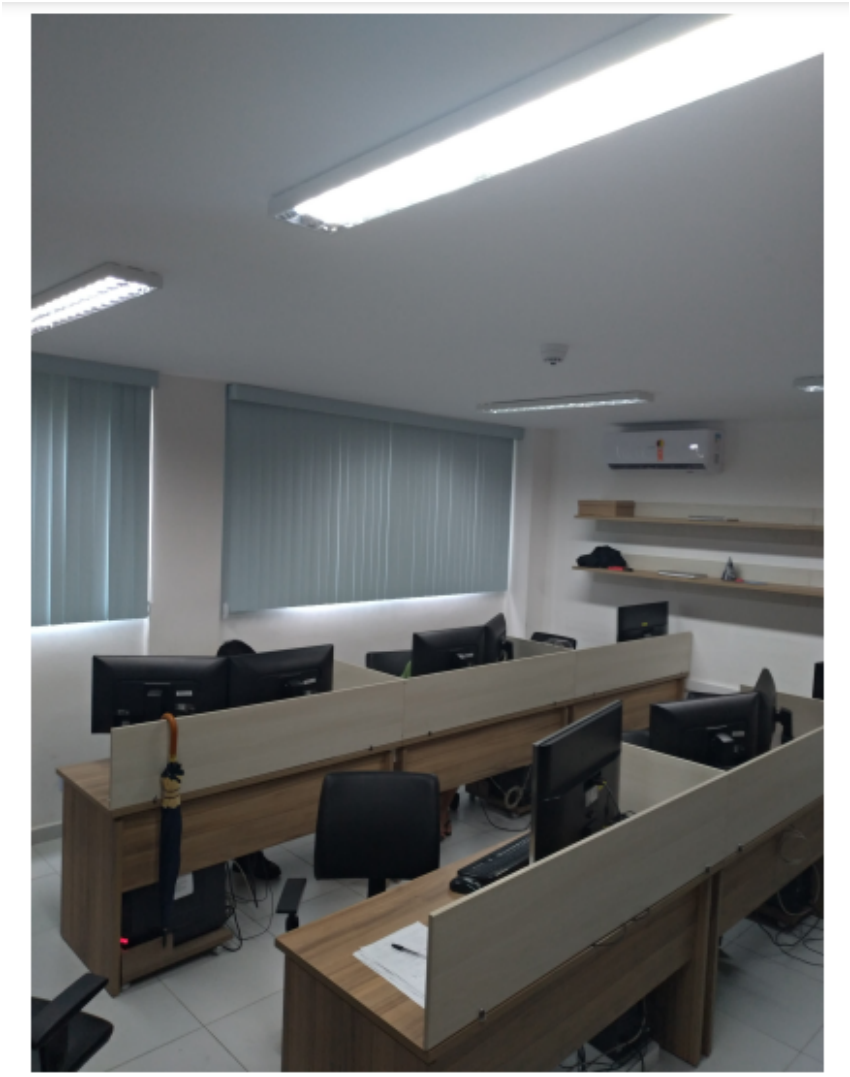


Figura 1 – Laboratório XMEN

Fonte: Autoria própria.

1.2 Objetivos

O objetivo do trabalho é relatar o trabalho desenvolvido durante o estágio no laboratório XMEN, sendo realizado na área de microeletrônica, mais especificamente no desenvolvimento de exercícios, modelos de referência e teste para uma plataforma de aprendizado de *design* digital de microeletrônica e *SystemVerilog*.

Durante o estágio, as seguintes atividades foram realizadas:

- Reuniões acerca da estruturação do site, designação de atividades e escolha das ferramentas a serem utilizadas;
- Desenvolvimento de atividades com modelos de referência e testes na linguagem *SystemVerilog* por conteúdo;
- Criação de *makefiles* para utilização do *Icarus Verilog* e *GTK Wave*.

1.3 Organização do Trabalho

O trabalho está estruturado em 4 capítulos, incluindo este introdutório, conforme a seguir.

Nesse capítulo foi apresentada uma breve introdução e os objetivos do estágio, bem como a estrutura de organização do trabalho.

No **Capítulo 2** será discorrido sobre a fundamentação teórica de *design* digital e da linguagem *SystemVerilog*.

No **Capítulo 3** serão apresentadas as atividades realizadas pelo estagiário.

Por fim, no **Capítulo 4** apresenta-se a conclusão sobre o trabalho.

2 Fundamentação Teórica

Neste capítulo, será apresentada uma fundamentação teórica sobre *design* de circuitos digitais e a linguagem de descrição de *hardware SystemVerilog*.

2.1 *Design* de Circuitos Digitais

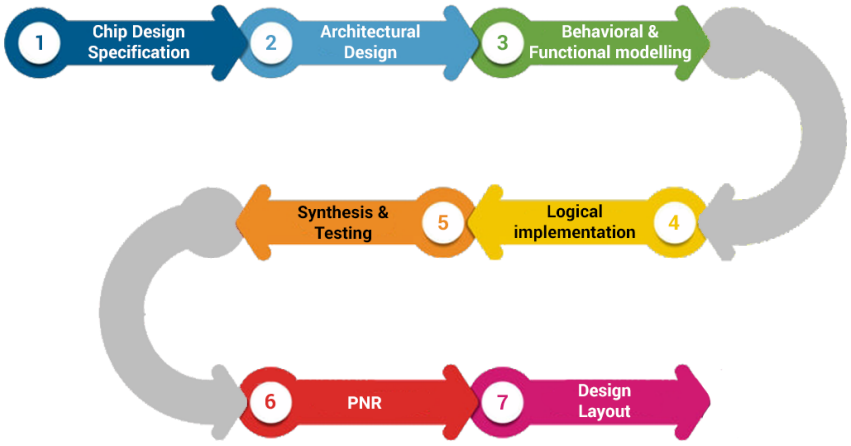
O *design* de circuitos digitais é uma área especializada que lida com a concepção, projeto e implementação de circuitos digitais em níveis muito baixos de integração, como chips e sistemas em um único chip (SoCs). Esses circuitos digitais são essenciais para o funcionamento de dispositivos eletrônicos modernos, como computadores, *smartphones*, sistemas embarcados e muito mais. Eles envolvem a integração de componentes eletrônicos, como transistores, resistores, capacitores, entre outros, em um único chip.

Ao projetar circuitos digitais para microeletrônica, alguns dos principais desafios incluem o aumento da densidade de integração, a redução do consumo de energia, o aumento da velocidade de operação e a garantia de confiabilidade e robustez.

Uma das áreas de desenvolvimento de circuitos digitais mais abrangentes nos dias atuais é a área de *design ASIC*. É uma área especializada e que se concentra na criação de chips personalizados para aplicações específicas. Diferentemente dos chips de propósito geral, como microprocessadores, ASICs são projetados para executar tarefas específicas e são otimizados para desempenho, consumo de energia e custo.

O processo de *design ASIC* envolve várias etapas, como a definição dos requisitos do circuito, a criação da especificação de design, a implementação do circuito em um nível de abstração de alto nível, a simulação e verificação, a síntese lógica, o layout físico, a verificação pós-layout e a fabricação do chip, como pode ser visto em (MARTIN, 2002).

Figura 2 – Fluxo de *design ASIC*



Fonte: (CHAUHAN, 2020).

2.2 *SystemVerilog*

A linguagem de descrição de *hardware SystemVerilog* desempenha um papel fundamental no projeto de circuitos digitais, fornecendo uma linguagem de alto nível para modelagem e simulação do comportamento de sistemas digitais complexos (SUTHERLAND; DAVIDMANN; FLAKE, 2006). É uma extensão poderosa do *Verilog*, contendo recursos adicionais que aprimoram a capacidade de descrever, simular e verificar projetos de *hardware*.

Uma das principais vantagens do *SystemVerilog* é sua capacidade de descrever circuitos digitais em diferentes níveis de abstração. Ele suporta descrições estruturais, onde os componentes do circuito são conectados por portas e sinais, e descrições comportamentais, onde o comportamento do circuito é especificado usando regras e equações. Essa flexibilidade permite que os projetistas escolham o nível de detalhe que melhor se adapta à tarefa em questão. Uma característica importante do *SystemVerilog* é a digitação de dados. Tipos de dados poderosos, como números inteiros, reais, vetores e estruturas definidas pelo usuário, são introduzidos para tornar a modelagem de circuitos mais precisa e robusta. Além disso, a linguagem suporta primitivas de concorrência que permitem a escrita de circuitos assíncronos e paralelos que podem executar múltiplas operações simultaneamente.

Outro aspecto importante do *SystemVerilog* é a verificação do projeto. Ele fornece recursos poderosos para criar asserções (VIJAYARAGHAVAN; RAMANATHAN, 2005), que são instruções lógicas para testar propriedades específicas do circuito. Isso ajuda os *designers* a encontrar erros e garantir que o design esteja correto. Além disso, o *SystemVerilog* oferece suporte à verificação formal, uma técnica avançada que permite a verificação automática das propriedades do projeto sem simulação, contribuindo para uma

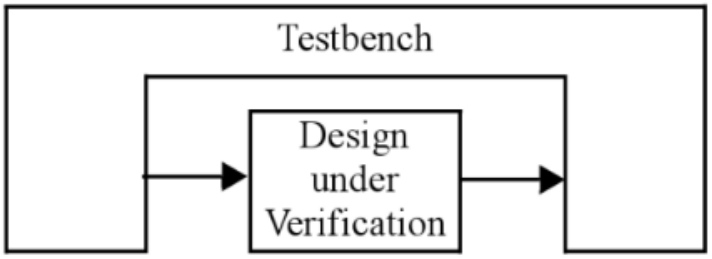
verificação mais abrangente e eficiente.

O SystemVerilog é especificado e mantido pela *IEEE* através do padrão *IEEE Std 1800*. O padrão define a linguagem, incluindo sua sintaxe, semântica e recursos.

2.3 *Testbench*

Um *testbench* em *SystemVerilog* é uma entidade separada que interage com o *design* em teste, fornecendo estímulos e verificando as saídas produzidas pelo *design* (BERGERON, 2007). Existem diferentes estilos de *testbench* que podem ser adotados, dependendo das necessidades específicas do projeto.

Figura 3 – Fluxo de interação entre *testbench* e *design*



Fonte: (BERGERON, 2007).

O primeiro estilo de *testbench* é o estrutural, que reflete a hierarquia do *design* em teste. Nesse estilo, o *testbench* instancia os componentes do *design* e conecta suas entradas e saídas, permitindo a aplicação de estímulos e a verificação dos resultados.

Outro estilo comum é o *testbench* de módulo único, que se concentra na verificação de um único módulo ou bloco do *design* em teste. Esse tipo de *testbench* fornece estímulos específicos para o módulo em questão e verifica se as saídas estão de acordo com as expectativas.

Por fim, temos o *testbench* de sistema, que abrange todo o sistema composto por múltiplos módulos interconectados. Esse estilo de *testbench* simula o comportamento do sistema como um todo, aplicando estímulos abrangentes e verificando as interações e saídas do sistema.

Além dos estilos de *testbench*, *SystemVerilog* oferece recursos avançados para aprimorar a eficiência e a capacidade de verificação. A geração automatizada de estímulos é uma dessas funcionalidades, permitindo a criação de sequências de estímulos ou o uso de algoritmos de geração aleatória para abranger uma ampla gama de cenários de teste.

O estilo de *testbench* escolhido para a atividade foram os *testbenches* de módulo único auto-verificáveis. Os *testbenches* auto-verificáveis são estruturas projetadas para estimular e verificar automaticamente um *design* em teste, proporcionando uma abordagem mais abrangente e completa. Ao empregar essa técnica, é possível gerar estímulos adequados para o *design*, monitorar suas saídas e compará-las com os resultados esperados.

3 Atividades Desenvolvidas

Esse capítulo discute acerca das atividades desenvolvidas durante o estágio que consistem na concepção dos exercícios para a plataforma *XSME*, bem como os seus modelos de referências e testes.

3.1 XSME

Xavier School for Micro Electronics (XSME) é uma plataforma de aprendizado de design digital de microeletrônica e SystemVerilog. Ela possui conteúdo didático sobre circuitos básicos, lógica combinacional e sequencial, seguido por exercícios para fixar o conhecimento adquirido. O nome é uma referência ao filme X-Men e ao laboratório XMEN (Laboratório de Excelência em Microeletrônica do Nordeste), que trabalha com programas de treinamento com estudantes de graduação em microeletrônica digital e analógica.

Para a criação da plataforma, foram dividias três equipes:

- *Frontend*: responsável pela criação das telas e do ambiente visual do site.

  Integrantes: Vitor Freire Bezerra

- Exercícios: responsável por desenvolver atividades sobre os contéudos de microeletrônica digital e *SystemVerilog*.

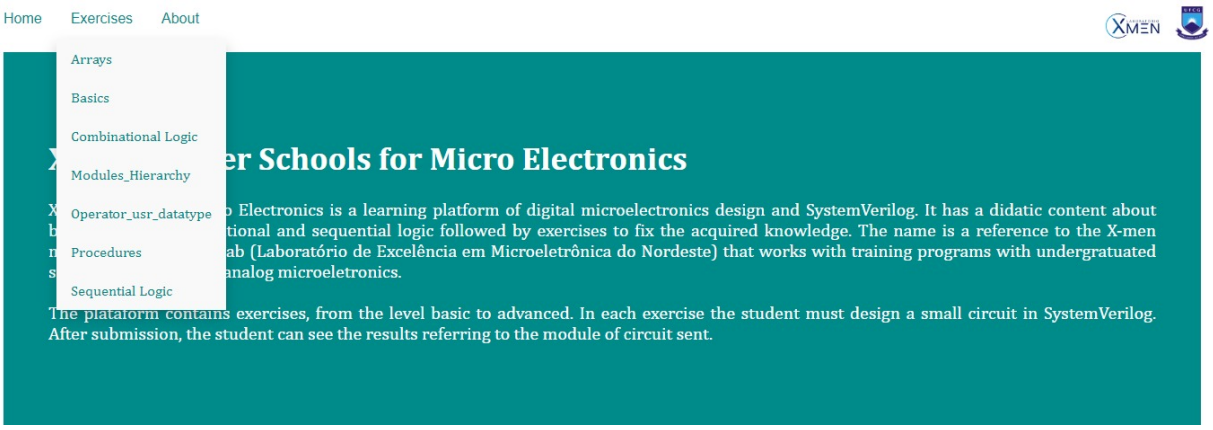
  Integrantes: Guilherme Teixeira Martins e Heriberto Gomes da Fonseca Junior

- *Backend*: responsável pela criação do banco de dados e *scripts* do site.

  Integrantes: Ana Rita Diniz da Cruz

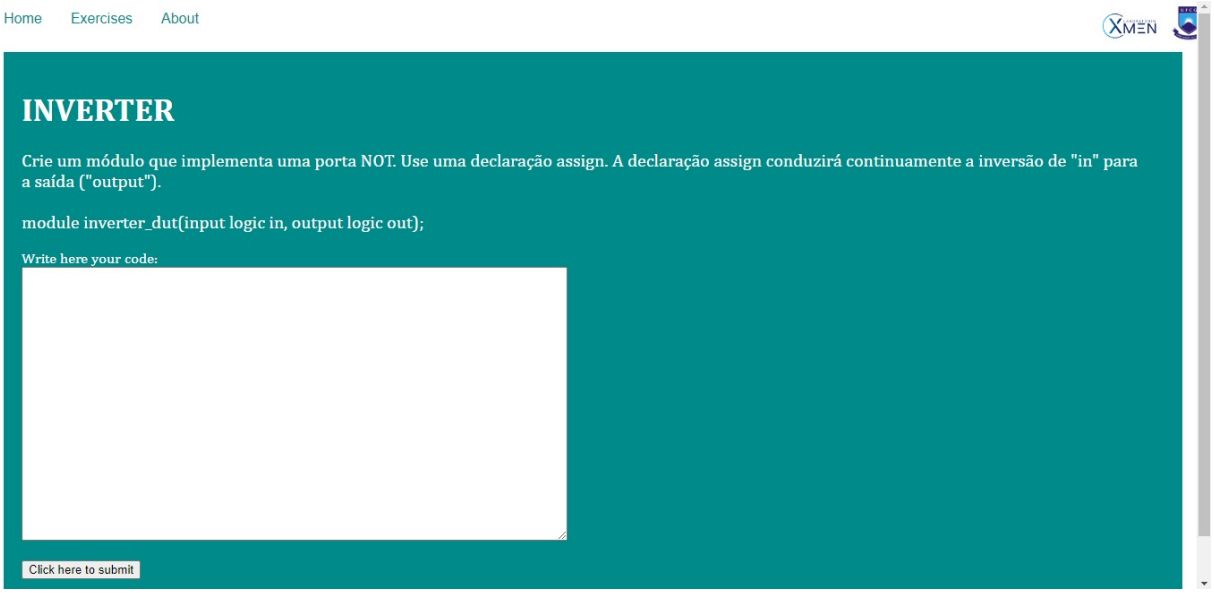
As Figura 4 e Figura 5 ilustra uma das telas do site que corresponde a tela de exercícios.

Figura 4 – Plataforma *XSME*



Fonte: Autoria própria.

Figura 5 – Aba de Atividade *XSME*



Fonte: Autoria própria.

3.2 Exercícios

Nesta seção, serão mostrados os tipos de exercícios desenvolvidos e sua estuturação. Todos os *testbenches* possuem um *'include'* inicial que remete a estrutura do banco de dados da plataforma e a escala de tempo que a simulação irá acontecer, como ilustrado no código abaixo.

- Estrutura do *Testbench*

```
// FILE NAME            : xxx_tb.sv
// AUTHOR               : Heriberto Fonseca

`include "files/golden/xxx_golden.sv"
`include "files/dut/dut_temp.sv"

`timescale 1ns/1ns
```

3.2.1 Básicos

Os exercícios básicos foram pensados com o intuito de relembrar conceitos de circuitos lógicos e trazê-los para a implementação em linguagem de descrição de *hardware*. Foram feitos exercícios acerca das portas lógicas e foi introduzido o conceito da variável *logic* que é uma das *features* trazidas pelo *SystemVerilog*. Para exemplificar, é ilustrado abaixo a descrição, o modelo de referência e o *testbench* do exercício sobre variáveis *logic*.

- Descrição: Crie uma variável do tipo *logic* de 1 bit e verifique sua saída para os diferentes tipos de entrada. Confirme a natureza *four-state* desse tipo de variável e pense em usos para os valores x e z.

```
    module  logic_dut(
      input logic in,
      output logic out
  );

```

- Modelo de Referência - Variável *Logic*:

```
    module  logic_golden(
      input logic in,
      output logic out
  );

     assign out = in;

    endmodule

```

- *Testbench* - Variável *Logic*:

```
// FILE NAME            : logic_tb.sv
```

```
2  // AUTHOR                : Heriberto Fonseca
3  // AUTHOR'S E-MAIL       : heriberto.fonseca@embedded.ufcg.edu.br
4
5  `include "files/golden/logic_golden.sv"
6  `include "files/dut/dut_temp.sv"
7
8  `timescale 1ns/1ns
9
10 module tb;
11
12 logic in_tb, out_golden, out_dut;
13
14 //INSTATIATION
15   logic_golden golden
16   (
17     .in(in_tb),
18     .out(out_golden)
19     );
20
21   logic_dut dut
22   (
23     .in(in_tb),
24     .out(out_dut)
25   );
26
27 // INITIAL BLOCK
28   initial
29   begin
30     #4 in_tb = 0;
31     #1 if(out_dut == out_golden)
32         $display("Passed: out = %d", out_dut);
33     else $error("Error: out = %d", out_dut);
34
35     #4 in_tb = 1;
36     #1 if(out_dut == out_golden)
37         $display("Passed: out = %d", out_dut);
38     else $error("Error: out = %d", out_dut);
39
40     #4 in_tb = 'z;
41     #1 if(out_dut == out_golden)
42         $display("Passed: out = %d", out_dut);
43     else $error("Error: out = %d", out_dut);
44
45     #4 in_tb = 'x;
46     #1 if(out_dut == out_golden)
47         $display("Passed: out = %d", out_dut);
48     else $error("Error: out = %d", out_dut);
```

```
49
50      $finish();
51    end
52
53 // TASKS & FUNCTIONS
54 //ASSERTIONS
55
56
57 endmodule
```

3.2.2 Operadores

Os operadores desempenham um papel crucial na manipulação de dados e na implementação de lógica de *hardware*. Eles permitem realizar operações aritméticas, lógicas, de atribuição, de comparação e de deslocamento, entre outras. A Figura 6 mostra uma lista de alguns operadores da linguagem *SystemVerilog*.

Figura 6 – Operadores *SystemVerilog*

Symbol	Usage	Meaning	Description
+=	a += b	a = a + b	add
-=	a -= b	a = a - b	subtract
*=	a *= b	a = a * b	multiply
/=	a /= b	a = a / b	divide
%=	a %= b	a = a % b	modulus
&=	a &= b	a = a & b	logical and
|=	a |= b	a = a | b	logical or
^=	a ^= b	a = a ^ b	logical xor
<<=	a <<= b	a = a << b	left shift logical
>>=	a >>= b	a = a >> b	right shift logical
>>>=	a >>>= b	a = a >>> b	right shift arithmetic
<<<=	a <<<= b	a = a <<< b	left shift arithmetic

Fonte: (IEEE..., 2018).

Além desses operadores básicos, *SystemVerilog* também inclui operadores bit a bit, operadores de redução, operadores condicionais, operadores ternários e operadores

de concatenação, entre outros, que oferecem maior flexibilidade e poder de expressão na criação de lógica de hardware. Para exemplificar os exercícios desenvolvidos, é ilustrado abaixo a descrição, o modelo de referência e o *testbench* do exercício sobre operador ternário.

- Descrição: Dado quatro números de 8 bits *unsigneds*, encontre o mínimo entre eles. Números *unsigneds* podem ser comparados usando operadores de comparação padrão (a < b). Use o operador ternário para criar circuitos comparativos utilizando valores auxiliares.

```
1    module  ternary_dut(
2      input logic [7:0] a, b, c, d,
3      output logic [7:0] min
4  );
5
```

- Modelo de Referência - Operador Ternário:

```
1    module  ternary_golden(
2      input logic [7:0] a, b, c, d,
3      output logic [7:0] min
4  );
5
6    logic [7:0] aux1, aux2;
7
8    always_comb begin
9     aux1 = (a < b) ? a : b;
10    aux2 = (aux1 < c) ? aux1 : c;
11    min = (aux2 < d) ? aux2 : d;
12   end
13
14   endmodule
15
```

- *Testbench* - Operador Ternário:

```
1 // FILE NAME            : ternary_tb.sv
2 // AUTHOR               : Heriberto Fonseca
3 // AUTHOR'S E-MAIL      : heriberto.fonseca@embedded.ufcg.edu.br
4
5 `include "files/golden/ternary_golden.sv"
6 `include "files/dut/dut_temp.sv"
```

```
7
8  `timescale 1ns/1ns
9
10 module tb;
11
12 //INTERNAL SIGNS
13   logic [7:0] a_tb, b_tb, c_tb, d_tb, min_golden, min_dut;
14
15 //INSTATIATION
16   ternary_golden golden
17   (
18     .a(a_tb),
19     .b(b_tb),
20     .c(c_tb),
21     .d(d_tb),
22     .min(min_golden)
23     );
24
25   ternary_dut dut
26   (
27     .a(a_tb),
28     .b(b_tb),
29     .c(c_tb),
30     .d(d_tb),
31     .min(min_dut)
32   );
33
34 // INITIAL BLOCK
35   initial
36   begin
37     for(int i=0; i<20; i++) begin
38       #4 a_tb = $urandom_range(0,255);
39       b_tb = $urandom_range(0,255);
40       c_tb = $urandom_range(0,255);
41       d_tb = $urandom_range(0,255);
42       #1 if(min_dut == min_golden)
43           $display("Passed(attempt %d): MIN_DUT = %d", i+1, min_dut...
    );
44         else $error("Error(attempt %d): MIN_DUT = %d", i+1, min_dut...
    );
45      end
46      $finish();
47   end
48
49 // TASKS & FUNCTIONS
50 //ASSERTIONS
51
```

```
52 endmodule
```

3.2.3 Vetores

Vetores são estruturas de dados que permitem armazenar e manipular conjuntos de valores do mesmo tipo. Eles desempenham um papel fundamental na descrição e manipulação de dados em *designs* de *hardware*. Em *SystemVerilog*, há duas formas de declarar vetores: vetores *packed* e vetores *unpacked*. Essas formas de declaração afetam o armazenamento e a alocação de memória dos elementos do vetor, e é importante compreender as diferenças entre elas.

Os vetores *packed* são compactados em um único bloco contíguo de memória, onde cada elemento do vetor é armazenado de forma consecutiva. A compactação é feita para otimizar o uso de memória e permitir acesso eficiente aos elementos individuais. Já os vetores *unpackeds* não são compactados em um único bloco de memória contígua e permitem maior flexibilidade na alocação de memória para cada elemento do vetor. Cada elemento do vetor é armazenado em sua própria localização de memória.

Para exemplificar os exercícios de vetores, é mostrado abaixo a descrição, o modelo de referência e *testbench* do exercício *unpacked arrays*.

- Descrição: Dado um vetor de 6 valores de 4 bits, encontre o valor máximo e mínimo comparando suas posições e os retorne na saída.

```
1     module  unpacked_dut(
2       input logic [3:0] unpacked [5:0],
3       output logic [3:0] max,
4       output logic [3:0] min
5   );
6
```

- Modelo de Referência - *Unpacked Array*:

```
1 // FILE NAME              : unpacked_golden.sv
2 // AUTHOR                 : Heriberto Gomes
3 // AUTHOR'S E-MAIL        : heriberto.fonseca@embedded.ufcg.edu.br or...
      heriberto.junior@ee.ufcg.edu.br
4
5
6 module  unpacked_golden(
7       input logic [3:0] unpacked [5:0],
8       output logic [3:0] max,
```

```
9         output logic [3:0] min
10    );
11
12  always_comb begin
13    max = unpacked[0];
14    min = unpacked[0];
15
16    for (int i = 0; i < 6 ; i++) begin
17      if(max < unpacked[i])
18        max = unpacked[i];
19      else
20        max = max;
21
22      if(min > unpacked[i])
23        min = unpacked[i];
24      else
25        min = min;
26    end
27  end
28
29  endmodule
```

- *Testbench - Unpacked Array*:

```
1  // FILE NAME            : unpacked_tb.sv
2  // AUTHOR               : Heriberto Fonseca
3  // AUTHOR'S E-MAIL      : heriberto.fonseca@embedded.ufcg.edu.br
4
5  `include "files/golden/unpacked_golden.sv"
6  `include "files/dut/dut_temp.sv"
7
8  `timescale 1ns/1ns
9
10 module tb;
11
12 //INTERNAL SIGNS
13   logic [3:0] unpacked_tb [5:0];
14   logic [3:0] max_golden, max_dut, min_golden, min_dut;
15
16
17 //INSTATIATION
18   unpacked_golden golden
19   (
20     .unpacked(unpacked_tb),
21     .max(max_golden),
22     .min(min_golden)
```

```
23    );
24
25    unpacked_dut dut
26    (
27      .unpacked(unpacked_tb),
28      .max(max_dut),
29      .min(min_dut)
30    );
31
32  // INITIAL BLOCK
33    initial
34    begin
35      for(int i = 0; i<10; i++) begin
36        #4 for (int j = 0;j<6 ;j++ ) begin
37          unpacked_tb[j] = $urandom_range(0,16);
38          $display("unpacked[%d] = %d", j, unpacked_tb[j]);
39        end
40        #1 if(max_dut == max_golden)
41            $display("Passed: maxDUT = %d / max_GOLDEN = %d", max_dut...
      , max_golden);
42          else $error("Error: max_DUT = %d / max_GOLDEN = %d", ...
      max_dut, max_golden);
43
44        if(min_dut == min_golden)
45          $display("Passed: min_DUT = %d / min_GOLDEN = %d", min_dut,...
       min_golden);
46        else $error("Error: min_DUT = %d / min_GOLDEN = %d", min_dut,...
       min_golden);
47      end
48
49      $finish();
50    end
51
52  // TASKS & FUNCTIONS
53  //ASSERTIONS
54
55
56  endmodule
```

3.2.4 Procedimentos

Um procedimento, em *SystemVerilog*, é uma unidade de código que executa uma sequência de instruções definida. Elas são usadas para encapsular blocos de código reutilizáveis e são semelhantes a funções ou sub-rotinas em outras linguagens de programação. Os prodecimentos são especialmente úteis para modelar o comportamento do bloco em uma descrição de *hardware*.

Para exemplificar os exercícios sobre procedimentos, é mostrado abaixo a descrição, o modelo de referência e *testbench* do exercício *priority encoder with casez*.

- Descrição: Faça um codificador de prioridade para entradas de 8 bits. Dado um vetor de 8 bits, a saída deve indicar o primeiro bit (menos significativo) no vetor que é 1. Reporte zero se o vetor de entrada não tiver bits em nível alto. Por exemplo, a entrada 8'b10010000 deve produzir a saída 3'd4, porque o bit[4] é o primeiro bit em nível alto.

```
1     module  priority_casez_dut(
2       input logic [7:0] in,
3       output logic [2:0] pos
4   );
5
```

- Modelo de Referência - *Priority Encoder with casez*:

```
1  // FILE NAME            : priority_casez_golden.sv
2  // AUTHOR               : Heriberto Gomes
3  // AUTHOR'S E-MAIL      : heriberto.fonseca@embedded.ufcg.edu.br or...
       heriberto.junior@ee.ufcg.edu.br
4
5
6  module  priority_casez_golden(
7        input logic [7:0] in,
8        output logic [2:0] pos
9    );
10
11   always_comb begin
12     priority casez (in)
13       8'bzzzzzzz1: pos = 0;
14       8'bzzzzzz1z: pos = 1;
15       8'bzzzzz1zz: pos = 2;
16       8'bzzzz1zzz: pos = 3;
17       8'bzzz1zzzz: pos = 4;
18       8'bzz1zzzzz: pos = 5;
19       8'bz1zzzzzz: pos = 6;
20       8'b1zzzzzzz: pos = 7;
21       default: pos = 0;
22     endcase
23
24   end
25
26 endmodule
```

- *Testbench - Priority Encoder with casez*:

```
// FILE NAME            : priority_casez_tb.sv
// AUTHOR               : Heriberto Fonseca
// AUTHOR'S E-MAIL      : heriberto.fonseca@embedded.ufcg.edu.br


`include "files/golden/priority_casez_golden.sv"
`include "files/dut/dut_temp.sv"

`timescale 1ns/1ns

module tb;

//INTERNAL SIGNS
  logic [7:0] in_tb;
  logic [2:0] pos_golden, pos_dut;

//INSTATIATION
  priority_casez_golden golden
  (
    .in(in_tb),
    .pos(pos_golden)
  );

  priority_casez_dut dut
  (
    .in(in_tb),
    .pos(pos_dut)
  );

// INITIAL BLOCK
  initial
  begin

    for(int i = 0; i<20; i++) begin
    #4 in_tb = $urandom_range(0,255);
    $display("IN = %b", in_tb);
    #1 if(pos_dut == pos_golden)
        $display("Passed: pos_DUT = %d / pos_GOLDEN = %d", pos_dut,...
    pos_golden);
      else $error("Error: pos_DUT = %d / pos_GOLDEN = %d", pos_dut,...
    pos_golden);
    end

    $finish();
```

```
43    end
44
45  // TASKS & FUNCTIONS
46  //ASSERTIONS
47
48  endmodule
```

3.2.5 Lógica Combinacional

A lógica combinacional em *SystemVerilog* refere-se à implementação de circuitos digitais que realizam operações lógicas sem armazenar estados. Esses circuitos são projetados para produzir uma saída determinada apenas pelos valores de entrada presentes no momento em que a operação é executada.

SystemVerilog oferece várias formas de descrever lógica combinacional. As principais são através de atribuições contínuas (*assign*) e *always blocks*.

Para exemplificar os exercícios sobre lógica combinacional, é mostrado abaixo a descrição, o modelo de referência e *testbench* do exercício *gatesv*.

- Descrição: Você recebe um vetor de entrada de quatro bits in[3:0]. Queremos saber algumas relações entre cada bit e seu vizinho:

  out_both: Cada bit deste vetor de saída deve indicar se tanto o bit de entrada correspondente quanto seu vizinho à esquerda (índice mais alto) são '1'. Por exemplo, *out_both*[2] deve indicar se in[2] e in[3] são ambos 1. Como in[3] não possui vizinho à esquerda, a resposta é óbvia, então não precisamos saber *out_both*[3].

  out_any: Cada bit deste vetor de saída deve indicar se algum dos bits de entrada correspondentes ou seu vizinho à direita é '1'. Por exemplo, *out_any*[2] deve indicar se in[2] ou in[1] são 1. Como in[0] não possui vizinho à direita, a resposta é óbvia, então não precisamos saber *out_any*[0].

  out_different: Cada bit deste vetor de saída deve indicar se o bit de entrada correspondente é diferente de seu vizinho à esquerda. Por exemplo, *out_different*[2] deve indicar se in[2] é diferente de in[3]. Para esta parte, trate o vetor como se fosse circular, ou seja, o vizinho à esquerda de in[3] é in[0].

```
1   module gatesv_dut(
2    input logic [3:0] in,
3    output logic [2:0] out_both,
4    output logic [3:1] out_any,
5    output logic [3:0] out_different );
6
```

- Modelo de Referência - *GateSV*:

```
module gatesv_golden(
    input logic [3:0] in,
    output logic [2:0] out_both,
    output logic [3:1] out_any,
    output logic [3:0] out_different );

    always_comb begin

        for (int i = 0; i < 3; i++) begin
            out_both[i] = in[i] & in[i+1];
        end

        for (int i = 1; i < 4; i++) begin
            out_any[i] = in[i] | in[i-1];
        end

        for (int i = 0; i < 3; i++) begin
            out_different[i] =  in[i] ^ in[i+1];
        end

        out_different[3] = in[3] ^ in[0];

    end

endmodule
```

- *Testbench* - *GateSV*:

```
`include "files/golden/gatesv_golden.sv"
`include "files/dut/dut_temp.sv"


`timescale 1ns/1ns

module tb;

//INTERNAL SIGNS

  logic [3:0] in_tb;
  logic [2:0] out_both_golden, out_both_dut;
    logic [3:1] out_any_golden, out_any_dut;
    logic [3:0] out_different_golden, out_different_dut;

```

```
16
17 //INSTATIATION
18   gatesv_golden golden(
19        .in(in_tb),
20        .out_both(out_both_golden),
21        .out_any(out_any_golden),
22        .out_different(out_different_golden)
23   );
24
25   gatesv_dut dut
26   (
27     .in(in_tb),
28        .out_both(out_both_dut),
29        .out_any(out_any_dut),
30        .out_different(out_different_dut)
31   );
32
33 // INITIAL BLOCK
34   initial
35   begin
36
37     $dumpfile("dumpfile.vcd");
38     $dumpvars(0,tb);
39     for (int i = 0; i < 16 ; i++ ) begin
40       in_tb = i;
41       #5 $display("in_tb = %b", in_tb);
42       #10 if(out_both_dut == out_both_golden) $display("Passed: ...
    out_both_dut = %b / out_both_golden = %b", out_both_dut, ...
    out_both_golden);
43       else $error("Error: out_both_dut = %b / out_both_golden = %b"...
    , out_both_dut, out_both_golden);
44       if(out_any_dut == out_any_golden) $display("Passed: ...
    out_any_dut = %b / out_any_golden = %b", out_any_dut, ...
    out_any_golden);
45       else $error("Error: out_any_dut = %b / out_any_golden = %b", ...
    out_any_dut, out_any_golden);
46       if(out_different_dut == out_different_golden) $display("...
    Passed: out_different_dut = %b / out_different_golden = %b", ...
    out_different_dut, out_different_golden);
47       else $error("Error: out_different_dut = %b / ...
    out_different_golden = %b", out_different_dut, ...
    out_different_golden);
48     end
49     $finish;
50
51   end
52
```

```
53
54 // TASKS & FUNCTIONS
55 //ASSERTIONS
56
57 endmodule
```

3.2.6 Lógica Sequencial

A lógica sequencial em *SystemVerilog* refere-se à implementação de circuitos digitais que possuem elementos de memória e cujo comportamento é determinado tanto pelas entradas presentes quanto pelos estados anteriores do circuito. Esses circuitos são projetados para armazenar informações e realizar operações com base em eventos ou condições específicas.

SystemVerilog oferece recursos para descrever lógica sequencial, sendo o mais comum o uso de always blocks em conjunto com variáveis de estado.

Para exemplificar os exercícios sobre lógica sequencial, é mostrado abaixo a descrição, o modelo de referência e *testbench* do exercício sobre máquina de estados finitos de Mealy.

- Descrição: Implemente uma máquina de estados finitos do tipo Mealy que reconheça a sequência 101 em um sinal de entrada chamado x. Sua FSM deve ter um sinal de saída, z, que é ativado para lógica 1 quando a sequência 101 é detectada. Sua FSM também deve ter um reset assíncrono ativo em nível baixo. Você só pode ter 3 estados na sua máquina de estados. Sua FSM deve reconhecer sequências sobrepostas.

```
1    module mealy_fsm_dut (
2     input logic clk,
3     input logic aresetn,     // Asynchronous active-low reset
4     input logic x,
5     output logic z );
6
```

- Modelo de Referência - *Mealy FSM*:

```
1 module mealy_fsm_golden (
2     input logic clk,
3     input logic aresetn,     // Asynchronous active-low reset
4     input logic x,
5     output logic z );
6
7     typedef enum logic [1:0]
8     {
```

```
9           STATE_1B,       //00
10          STATE_2B,       //01
11          STATE_3B     //02
12      } state;
13
14      state current, next;
15
16      always_ff @(posedge clk or negedge aresetn) begin
17          if(!aresetn) current <= STATE_1B;
18          else current <= next;
19      end
20
21      always_ff @(posedge clk) begin
22          case (current)
23              STATE_1B: begin
24                  if(x) next <= STATE_2B;
25                  else next <= STATE_1B;
26              end
27              STATE_2B: begin
28                  if(!x) next <= STATE_3B;
29                  else next <= STATE_2B;
30              end
31              STATE_3B: begin
32                  if(x) next <= STATE_2B;
33                  else next <= STATE_1B;
34              end
35              default: next <= STATE_1B;
36          endcase
37      end
38
39      always_comb begin
40          case (current)
41              STATE_1B: z = 0;
42              STATE_2B: z = 0;
43              STATE_3B: z = 1;
44              default: z = 0;
45          endcase
46      end
47
48  endmodule
```

- *Testbench - Mealy FSM*:

```
1  // FILE NAME            : asyncReset_tb.sv
2  // AUTHOR               : Heriberto Gomes
3  // AUTHOR'S E-MAIL      : heriberto.fonseca@embedded.ufcg.edu.br or...
```

```
    heriberto.junior@ee.ufcg.edu.br


`include "files/golden/mealy_fsm_golden.sv"
`include "files/dut/dut_temp.sv"


`timescale 1ns/1ns

module tb;

//INTERNAL SIGNS

  logic x_in;
  logic clk;
  logic aresetn;
  int count_golden = 0;
  int count_dut = 0;

  logic z_golden, z_dut;

//INSTATIATION
  mealy_fsm_golden golden
  (
    .clk(clk),
    .aresetn(aresetn),
    .x(x_in),
    .z(z_golden)
  );

  mealy_fsm_dut dut
  (
    .clk(clk),
    .aresetn(aresetn),
    .x(x_in),
    .z(z_dut)
  );

// INITIAL BLOCK
always begin
  #5 clk = ¬clk;
end

  initial
  begin
    $dumpfile("dumpfile.vcd");
    $dumpvars(0,tb);
```

```
50      aresetn = 1;
51      clk = 0;
52        @(posedge clk); aresetn = 0;
53        @(posedge clk); aresetn = 1;
54        count_dut = 0;
55        count_golden = 0;
56        repeat(20) begin
57          @(posedge clk);
58          x_in = $urandom_range(0,1);
59          $display("x = %d", x_in);
60          if(z_golden) count_golden++;
61          if(z_dut) count_dut++;
62        end
63
64            if(count_dut == count_golden) $display("Passed: count_dut...
      = %d / count_golden = %d (number of times that the sequence 101...
      was identified)", count_dut, count_golden);
65        else $error("Error: count_dut = %d / count_golden = %d (...
     number of times that the sequence 101 was identified)", ...
     count_dut, count_golden);
66
67      $finish;
68    end
69
70  // TASKS & FUNCTIONS
71  //ASSERTIONS
72
73  endmodule
```

4 Conclusões

Neste relatório, foram descritas as atividades realizadas pelo aluno durante o estágio no Laboratório de Microeletrônica do Nordeste. O estágio proporcionou uma valiosa oportunidade para aplicar os conhecimentos adquiridos ao longo do curso de Engenharia Elétrica na universidade. A experiência de contribuir para o XMEN resultou em um crescimento profissional significativo, oferecendo maior contato com o mercado de trabalho e a possibilidade de trabalhar em equipe, colaborando com outros alunos e engenheiros. Além disso, o estágio proporcionou acesso a cursos que enriqueceram o conhecimento do estagiário na área de desenvolvimento de *hardware*.

Referências Bibliográficas

BERGERON, J. *Writing testbenches using SystemVerilog*. [S.l.]: Springer Science & Business Media, 2007. Citado na página 6.

CHAUHAN, K. Asic design flow in vlsi engineering services (a quick guide). *eInfoChips*, 2020. Citado na página 5.

IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, p. 1–1315, 2018. Citado na página 12.

MARTIN, K. *Digital Integrated Circuit Design*. second. [S.l.]: Oxford University Press, 2002. Citado na página 4.

SUTHERLAND, S.; DAVIDMANN, S.; FLAKE, P. *SystemVerilog for Design: A Guide to using SystemVerilog for Hardware Design and Modeling*. second. [S.l.]: Springer Science Business Media, 2006. Citado na página 5.

VIJAYARAGHAVAN, S.; RAMANATHAN, M. *A practical guide for SystemVerilog assertions*. first. [S.l.]: Springer Science Business Media, 2005. Citado na página 5.