Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Vinicius Bonfim Araujo

Relatório de Estágio Integrado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Campina Grande
Agosto de 2021

Vinicius Bonfim Araujo

Relatório de Estágio Integrado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Relatório de Estágio Integrado submetido à Coordenação de Graduação em 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.

Orientador:
Marcos Ricardo Alcântara Morais, D. Sc.

Campina Grande
Agosto de 2021

Vinicius Bonfim Araujo

Relatório de Estágio Integrado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Relatório de Estágio Integrado submetido à Coordenação de Graduação em 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 / /

Marcos Ricardo Alcântara Morais, D. Sc.
UFCG

Gutemberg Gonçalves dos Santos Júnior, D. Sc.
UFCG

Campina Grande

Agosto de 2021

Dedico esse trabalho aos meus pais, Roseli Rocha Bonfim e João Francisco de Araújo, no qual sem toda a ajuda e suporte deles jamais teria chegado onde cheguei.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus, pois sem Ele nada seria possível.

Em segundo, aos meus pais, Roseli Rocha Bonfim e João Francisco de Araújo, que nunca mediram esforços para me ajudar e me apoiar e fazer com que essa formação se tornasse possível.

Ao laboratório XMEN e em especial a Kaline que sempre me motivou e ajudou desde que entrei no laboratório, a Pedro e Klynger que me deram suporte durante todo o período de estágio, a Hugo que de inúmeras maneiras se prontificou a tornar o estágio uma ótima experiência e a todos que de alguma forma me ajudaram nesse período.

Ao meu orientador, Marcos Morais, pelos conselhos durante este período de estágio e ao professor, Gutemberg Júnior, que foi determinante para que o estágio nesse período de pandemia fosse concretizado.

RESUMO

Esse relatório apresenta descritivamente as atividades realizadas pelo estagiário, Vinicius Bonfim Araujo, durante o período de estágio no Laboratório de Excelência em Microeletrônica do Nordeste. Consta também dos temas estudados e utilizados pelo mesmo durante o desenvolvimento da sua principal atividade, a verificação funcional em UVM de blocos digitais. Além disso, expõe a metodologia utilizada para gerenciamento das atividades realizadas.

Palavras-chave: CRC, Verificação funcional, Descrição de *Hardware*, *Hardware*, UVM, *testbench*, Estágio.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CI	Circuito Impresso
CRC	*Cyclic Redundancy Check*
DEE	Departamento de Engenharia Elétrica
DUT	*Design Under Test*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IP	*Intellectual Property*
JTAG	*Joint Test Action Group*
RTL	*Register Transfer Level*
SoC	*System on Chip*
UFCG	Universidade Federal de Campina Grande
UVM	*Universal Verification Methodology*
VIP	*Verification Intellectual Property*
XMEN	Laboratório de Excelência em Microeletrônica do Nordeste

SUMÁRIO

1 INTRODUÇÃO

O estágio tem como proposta pôr em prática os conhecimentos obtidos durante a formação, assim como promover crescimento profissional e pessoal ao estagiário garantindo-lhe mais experiência, tornando-o assim mais apto para os desafios advindos do mercado de trabalho.

Este relatório em questão tem como objetivo descrever as atividades realizadas pelo graduando, Vinicius Bonfim Araujo, durante o período de estágio no Laboratório de Excelência em Microeletrônica do Nordeste - XMEN, tendo parte de seu desdobramento de forma presencial no laboratório em Campina Grande e parte de forma remota na residência do mesmo devidos as condições de pandemia.

Com a carga horária de 788 horas totais, sendo assim 40 horas semanais, o estágio se enquadrou como estágio integrado e foi iniciado em 04 de fevereiro de 2021 e finalizado em 30 de julho de 2021. O estagiário foi orientado pelo professor Marcos Ricardo Alcântara Morais e supervisionado pelo professor Gutemberg Gonçalves dos Santos Júnior.

As atividades a serem desenvolvidas seguiam o plano de estágio programado, que constava de três macro atividades:

1. Planejar atividades de Verificação Funcional;
2. Desenvolver ambientes de verificação funcional utilizando metodologia UVM;
3. Executar testes e regressões para verificação de blocos digitais.

2 LOCAL DO ESTÁGIO

O estágio foi realizado no Laboratório de Excelência em Microeletrônica no Nordeste (XMEN) que atua na capacitação e pesquisa aplicada na área de microeletrônica digital, incluindo SoC design e Digital IPs para sistemas embarcados, realizadas por alunos da graduação dos cursos de engenharia elétrica e ciências da computação e profissionais graduados em engenharia na mesma área.

(a) (b)

Figura 1 – Instalações do XMEN

Fonte – Própria

Originada do antigo projeto PEM - Programa para Excelência em Microeletrônica - que teve sua iniciativa dada em 2016 vinculada ao Laboratório EMBEDDED, hoje conta com o VIRTUS como sua parceira, e desenvolve trabalho e pesquisa para diversos parceiros clientes do VIRTUS.

O laboratório conta com o desenvolvimento dividido em três equipes, sendo elas: equipe de *frontend*, responsável pela projeção lógica e arquitetural do projeto a ser desenvolvido; a equipe de verificação, responsável pela verificação das funcionalidades especificadas no projeto dada a descrição de hardware (RTL) desenvolvida pela equipe de *frontend*; e por fim a equipe de *backend* que tem como função o desenvolvimento do

projeto de construção física do circuito projetado. A ordem de apresentação das equipes configura também um fluxo de trabalho para o desenvolvimento de um *chip*.

3 FUNDAMENTAÇÃO TEÓRICA

Com o objetivo de facilitar o entendimento sobre as atividades realizadas durante o estágio esta seção descreve conceitos a respeito das funcionalidades presentes nos blocos propostos a serem verificados, assim como teorias presentes na verificação.

3.1 VERIFICAÇÃO CÍCLICA DE REDUNDÂNCIA - CRC

Em modernos sistemas computacionais, a transmissão de dados ocorre continuamente, tanto entre o processador e seus periféricos, quanto entre os próprios periféricos. Através de tantas transferências e devido a presença de ruídos durante o processo, erros podem ser gerados e propagados ao receptor da mensagem. Dentre tantas formas de verificação de erros de transmissão estão: o método de paridade, *checksum*, o método de repetição, e o método utilizado no bloco verificado, o método de CRC.

A verificação cíclica de redundância, ou da sigla inglesa, CRC (*cyclic redundancy check*), é um código de verificação de erros que é amplamente utilizado em sistemas de comunicação de dados e outros sistemas de transmissão serial de dados. O CRC é baseado na manipulação polinomial usando módulos aritméticos. Alguns dos mais comuns padrões de CRC são CRC-8, CRC-12, CRC-16, CRC-32 e CRC-CCIT (BORRELLI, 2001).

O cálculo de CRC é composto de uma multiplicação baseada no grau de CRC a ser aplicado e uma divisão entre a mensagem a ser enviada e o gerador. Entretanto, ao invés de se utilizar de subtrações durante a divisão, são utilizadas operações XOR. Ao final da operação haverá um resto que, baseado no grau do CRC, será o valor de CRC calculado.

Comumente o método funciona da seguinte forma: dada uma informação a ser transmitida, um valor de CRC é calculado considerando o padrão utilizado. Posteriormente, este valor é indexado à mensagem que é enviada. No receptor a mensagem com o valor de CRC é separada pelo gerador e de acordo com o resto da operação, sendo seu valor igual a zero para uma transmissão sem erros. A Figura 2 demonstra como é realizado o cálculo. O gerador pode ser considerado como um código polinomial, uma vez que é possível representa-lo como um sequencial de *bits*. Esses *bits* são comumente utilizados para identificação do padrão do CRC.

Os padrões de CRC são definidos por parâmetros, são eles: o grau do CRC, o código polinomial, o valor inicial de CRC e se o resultado final do CRC é inverso ou não. Alguns exemplos desses CRC são: CRC-16 genibus, CRC-16 buypass, CRC-8, CRC-32, CRC-32/MPEG-2.

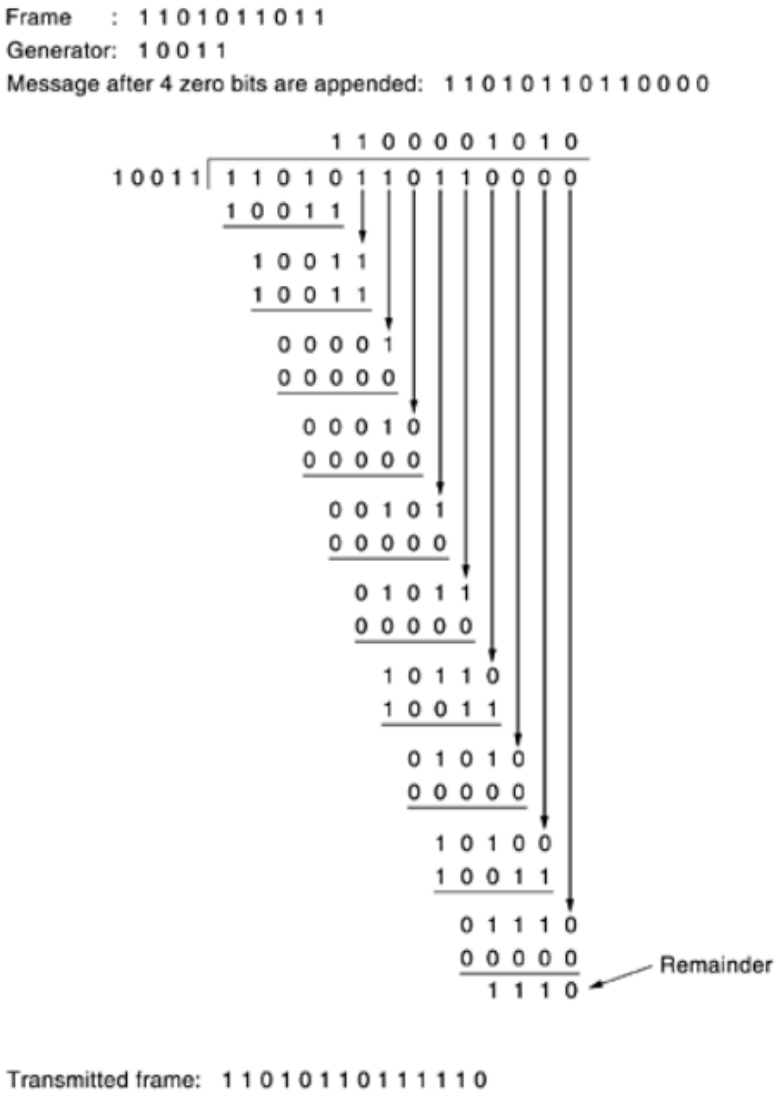

Figura 2 – Cálculo padrão de CRC de 4 bits

Fonte – (TANENBAUM, 2003)

3.2 *BOUNDARY SCAN* - JTAG

O acrônimo JTAG advém de *Joint Test Action Group*, este grupo na década de 1980 trabalhou para desenvolver um padrão no processo de *boundary scan* (varredura de limite). Como resultado desse trabalho e após aceitação de propostas ao comitê de testabilidade padrão de barramento do IEEE foi criado o projeto IEEE P1149.1 e decidido que a proposta do JTAG se tornaria um padrão dentro da família de testabilidade de barramentos (IEEE. . . , 2001). O padrão ficou amplamente conhecido como padrão JTAG, ou simplesmente JTAG. A técnica de *boundary scan* também ficou conhecida e popularmente chamada de JTAG.

Utilizada para a depuração da placa, o *boundary scan* obtém acesso as informações do CI (circuito impresso) de forma serial (*daisy chain*). A técnica amplamente empregada

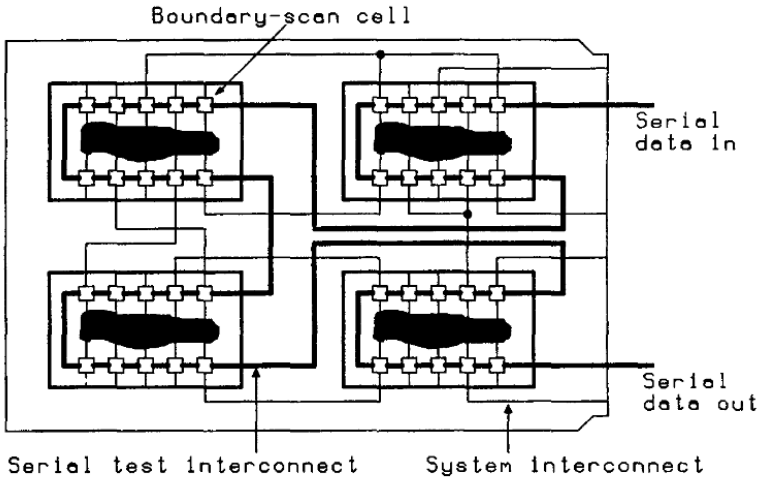

Figura 3 – Exemplo de PCBA com *boundary scan*

Fonte – (IEEE. . . , 2001)

em PCBA's *(Printed Circuit Board Assemblies - Placa Assembly de circuito impresso)*, que são placas com todos os componentes soldados e instalados na placa de circuito impresso, utiliza da ligação serial entre os pinos de cada CI que continuarmente se conecta aos demais componentes da entrada do sinal de teste da interface do JTAG até sua saída, formando assim uma cadeia de varredura ou no inglês *scan chain*. Um exemplo com o esquema de quatro componentes pode ser visto na Figura 3.

3.3 VERIFICAÇÃO FUNCIONAL

Na implementação de uma Propriedade Intelectual (IP), um fluxo de desenvolvimento é seguido, passando por diversas fases e englobando diversas equipes durante o processo. Como dito anteriormente, um código RTL é criado pela equipe de *frontend* e posteriormente tem sua funcionalidade verificada pela equipe de verificação, validando o RTL. Por fim, é executado o projeto físico do IP pela equipe de *backend* e enviado para fabricação. O IP consiste de implementação de hardware que possui funcionalidades, estas são documentadas e chamadas de especificação. A especificação rege todo o fluxo de desenvolvimento do IP.

A verificação funcional é um dos três tipos de verificação de hardware, esta procura assegurar que as funcionalidades postas na especificação do hardware sejam condizentes com as funcionalidades implementadas no projeto lógico. Segundo (BERGERON, 2003), a verificação funcional é definida como um processo usado para demostrar que o objetivo do projeto é preservado em sua implementação.

A garantia de funcionalidade é dada a partir da comparação entre as respostas do *Design Under Test* (DUT) e do modelo ideal - construído em uma linguagem de abstração

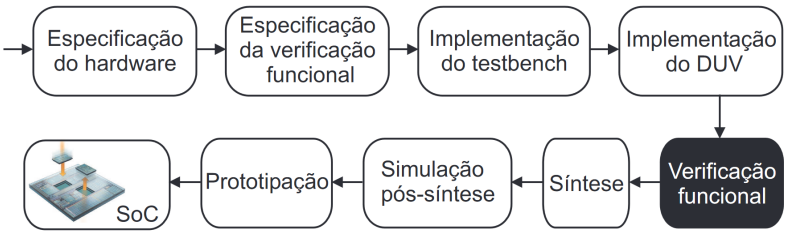

Figura 4 – Fluxo do desenvolvimento de hardware

Fonte – (SILVA, 2007)

maior para ter as respostas de acordo com as especificações - a partir de estímulos capazes de excitar as funcionalidades especificadas.

Um bloco é dado como verificado quando o plano de cobertura obtém todos os seus requisitos cumpridos. Para dar início à verificação de um bloco, primeiramente é feito um plano de verificação (PV) que consta de todos os sinais do bloco, os testes a serem realizados para estimulação do bloco, os parâmetros que regem os testes e como eles vão funcionar para cada teste. Também é criado um documento chamado de plano de cobertura, sendo este o documento mais importante durante a verificação, pois é ele quem irá garantir que um DUT foi dado como verificado uma vez que teve todas as suas métricas de cobertura atendidas.

Uma vertente importante a ser decidida é a granularidade do projeto a ser verificado. Quando um projeto é pensado para ser projetado, o mesmo é dividido em partes menores para que se maximize a possibilidade de reutilização de IPs, assim como para que a equipe de *frontend* consiga dividir de melhor forma o desenvolvimento do projeto. O mesmo se propaga para a equipe de verificação, quando um projeto é posto para ser verificado ele é normalmente dividido em partes menores, partes estas que já podem ter sido divididas na fase de design. A divisão do projeto faz com que a detecção de erros se torne mais fácil, visto que a área de investigação de erros será menor. O contraponto dessa divisão é a criação de uma maior quantidade de modelos e *testbenches* (SILVA, 2007).

Para o desenvolvimento do *testbench* (bancada de testes) se faz necessário seguir uma metodologia de verificação. Na verificação funcional podemos citar: UVM - *Universal Verification Methodology*, OVM - *Open Verification Methodology*, VMM - *Verification Methodology Manual*, amplamente utilizada e difundida tanto no meio acadêmico como no ramo industrial. O UVM é a metodologia foco desse relatório.

3.3.1 UVM

O UVM se tornou uma metodologia referência na verificação funcional de circuitos digitais. Por meio da biblioteca de classe do UVM é possível o desenvolvimento de um

testbench de forma ágil.

A metodologia é composta de componentes derivados da classe UVM, cada um desses componentes possui uma função essencial que torna o ambiente de *testbench* capaz de verificar o DUT.

Cada classe derivada possui fases em sua estrutura que é executada simultaneamente entre todos os componentes. As fases de maior escopo são (CAMPOS, 2016):

- A *build_phase* é responsável pela criação e configuração da estrutura do *testbench* construindo os componentes da hierarquia;
- A *connect_phase* é a fase em que as conexões entre os componentes do ambiente é realizada;
- A *run_phase* é a principal fase onde é executada a simulação;
- A *report_phase* pode ser utilizada para mostrar os resultados da simulação.

Uma estrutura de ambiente de verificação funcional em UVM é exemplificada abaixo. É válido ressaltar que não necessariamente alguns dos componentes do ambiente são estruturados de tal forma, podendo de acordo com a preferência do verificador ou da metodologia de trabalho da empresa serem estruturadas de forma diferente.

O objetivo do *testbench* é estimular o DUT de modo a obter suas respostas e comparar essas respostas com as respostas obtidas pelo modelo de referência diante dos mesmos estímulos. A comparação entre essas respostas produzem resultados que são classificados como: *match*, resultado que comprova que não houve diferença entre a resposta do DUT e a resposta do modelo de referência; e *mismatch*, contrário ao *match*, é o resultado que prova diferença entre as respostas.

Componente responsável pela geração de estímulos (*sequences*) do UVM, o *sequencer* gera-os e os envia ao *driver* que por sua vez transforma esses pacotes de estímulos em sinais de alimentação do DUT através da interface.

O DUT, por sua vez, responde aos estímulos gerando sinais de saída que são transformados em pacotes para compreensão do ambiente pelo *monitor*. Os pacotes de resposta do DUT juntamente com os estímulos que perpassam pela interface são lidos pelo *monitor* e enviados ao scoreboard, componente que engloba o modelo de referência (refmod) e o comparador de respostas (*comparator*). O refmod composto do modelo de referência e classes herdadas do UVM, recebe os mesmos estímulos que o DUT e fornece uma resposta ideal que concorda com a especificação do projeto. As respostas do DUT, recebida pelo scoreboard e a resposta do refmod são enviadas ao comparador de modo a verificar se há presença de *mismatches*.

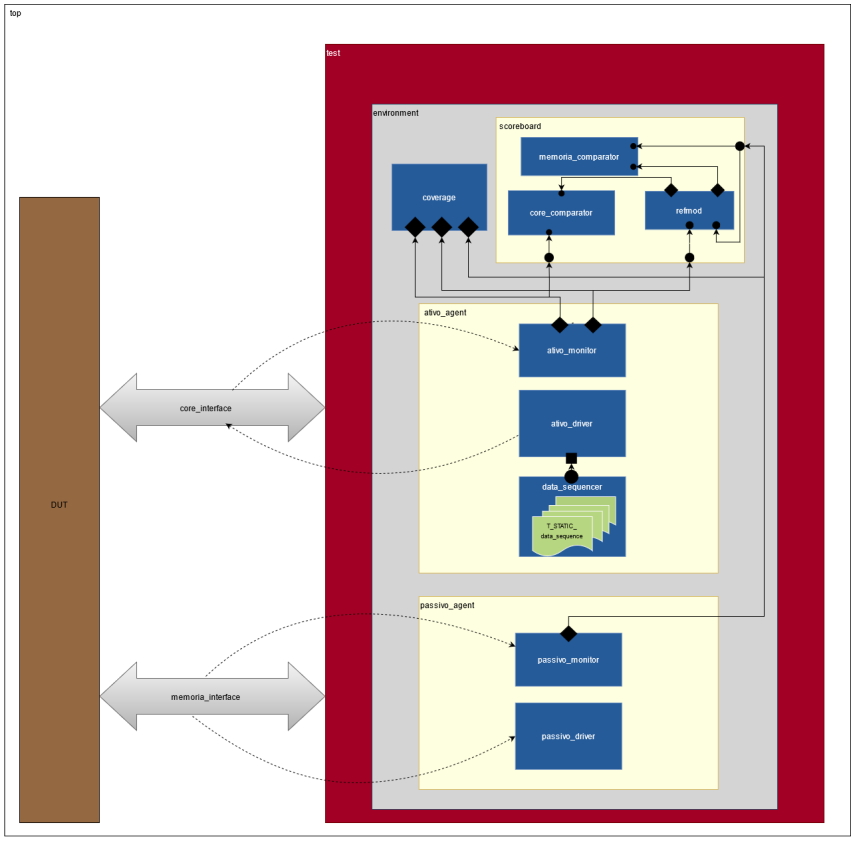

Figura 5 – UVM *testbench*

Fonte – Própria

```
UVM_ERROR @ 58540: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4903 vs. xcrc_tr@3881
UVM_WARNING @ 58540: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_INFO @ 58580: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58620: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58660: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58700: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58740: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58780: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58820: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_INFO @ 58860: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Match]
UVM_ERROR @ 58900: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h9474 : rhs = 'hff5b
UVM_ERROR @ 58900: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4812 vs. xcrc_tr@3881
UVM_WARNING @ 58900: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 58940: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'hd181 : rhs = 'h450b
UVM_ERROR @ 58940: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4826 vs. xcrc_tr@3881
UVM_WARNING @ 58940: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 58980: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h81be : rhs = 'hfe31
UVM_ERROR @ 58980: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4923 vs. xcrc_tr@3881
UVM_WARNING @ 58980: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59020: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'heba7 : rhs = 'ha772
UVM_ERROR @ 59020: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4811 vs. xcrc_tr@3881
UVM_WARNING @ 59020: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59060: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h6a3a : rhs = 'h330
UVM_ERROR @ 59060: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4907 vs. xcrc_tr@3881
UVM_WARNING @ 59060: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59100: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h8ff6 : rhs = 'h22f2
UVM_ERROR @ 59100: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4851 vs. xcrc_tr@3881
UVM_WARNING @ 59100: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59140: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h7cc3 : rhs = 'h4d49
UVM_ERROR @ 59140: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4812 vs. xcrc_tr@3881
UVM_WARNING @ 59140: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59180: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'hcc7c : rhs = 'h14ae
UVM_ERROR @ 59180: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4826 vs. xcrc_tr@3881
UVM_WARNING @ 59180: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59220: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'ha2ec : rhs = 'h3dc2
UVM_ERROR @ 59220: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4923 vs. xcrc_tr@3881
UVM_WARNING @ 59220: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59260: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h6c85 : rhs = 'h51ca
UVM_ERROR @ 59260: reporter [MISCMP] 1 Miscompare(s) for object xcrc_tr_out@4811 vs. xcrc_tr@3881
UVM_WARNING @ 59260: uvm_test_top.xcrc_env.xcrc_score.xcrc_comp [Comparator Mismatch]  differs from
UVM_ERROR @ 59300: reporter [MISCMP] Miscompare for xcrc_tr.xcrc_dto_o: lhs = 'h54b1 : rhs = 'h531
```

Figura 6 – *Matches* e *mismatches* vistos no terminal linux

Fonte – Própria

A cobertura (*coverage*) é responsável pelas métricas que serão utilizadas para definir se um bloco está ou não verificado ao término de todos os testes e é feita a partir do plano de cobertura previamente realizado. A classe *test* é responsável por realizar os testes criando o ambiente e conectando o *sequence* ao *sequencer*. O *driver*, o *monitor* e o *sequencer* formam o *agent*. O *agent* pode ser ativo ou passivo, e assim conter todos os três componentes ou apenas um deles, a depender da configuração planejada pelo verificador na verificação do bloco. A classe *environment* é formada pela junção do(s) *agent(s)*, do scoreboard e da cobertura. E o topo (*top*) é o componente onde se instancia o DUT e o *testbench* (CAMPOS, 2016).

3.4 METODOLOGIA SCRUM

A metodologia Scrum pode ser definida como um framework leve que ajuda pessoas, times e organizações a gerar valor por meio de soluções adaptativas para problemas complexos (SUTHERLAND; SCHWABER, 2020).

Com origem de sua utilização no desenvolvimento de produtos de *software*, hoje em dia a metodologia já é utilizadas em diversos âmbitos, e a mesma não tem limitação para sua utilização. O *scrum* é exercido dentro de uma equipe (*Scrum Team*), de modo a agilizar o desenvolvimento do produto. Ele foi criado por Jeff Sutherland, Ken Schwaber e Mike Beedle (CARVALHO; HENRIQUE; MELLO, 2011), com base em características:

- Flexibilidade de resultados;
- *Deadline* flexível;
- Equipes pequenas;
- Reuniões frequentes;
- Cooperação;
- Orientação a objeto.

O *Scrum Team*, definida como a unidade fundamental do *scrum* (SUTHERLAND; SCHWABER, 2020), é composta por uma pequena quantidade de pessoas e que se dividem em três responsabilidades específicas: os *Developers*, o *Product Owner* e o *Scrum Master*.

Começando pela parte desenvolvedora dos produtos, os *developers* tem como responsabilidade a criação dos planos a serem seguidos durante a *sprint* assim como a execução dos planos e adaptação deles para a meta da *sprint*. Responsável pelo gerenciamento eficaz do *product backlog*, o *product owner*, também tem como objetivo tornar claro a meta do produto, criar os items do *product backlog* e garantir que tais itens sejam transparentes

e compreensíveis. Estes objetivos ainda podem ser delegados ao *scrum master*, entretanto ele ainda responde como responsável da unidade. Capaz de exercer não apenas atividades de sua responsabilidade, o *scrum master* é peça fundamental para manter a eficácia do *framework*, ele serve ao *scrum team* antes, durante e ao fim de um projeto. Além disso, facilita a comunicação entre diferentes unidades de *scrum team* e auxilia os *developers* na adaptação do plano de trabalho diante de problemas que venham a se configurar durante a *sprint*. Liderança, planejamento e prestatividade são qualidades necessárias a *scrum master*.

O *scrum* no desenvolvimento de processos de sistemas é dividido em grupos de fases (SCHWABER, 1997), são elas:

- *Pregame*;
- *Game*;
- *Postgame*.

A fase de *pregame* consiste no planejamento dos itens de *backlog*, juntamente com a estimativa do cronograma do projeto e seus custos, e reunião de planejamento de *sprints*. Dentro do planejamento de *backlog*, uma variante que define o nível de dificuldade/tempo/grandeza dos itens precisa ser definida. Uma forma de se definir é utilizando da prática de *planning poker*, que nada mais é que a votação do nível de dificuldade de cada item do *backlog* pelos *developers* segundo a sequência de Fibonacci após a descrição clara do item a ser avaliado.

No *game* ocorre o desenvolvimento de trabalho dos *developers* segundo as *sprints* planejadas, respeitando os tempos e requisitos previstos. Existem várias *sprints* de desenvolvimento iterativo usados para a evolução do produto. Durante o *game* existe também a prática de *daily scrum*, que são reuniões diárias de curto período de tempo entre a equipe para definir quais serão as atividades a serem desenvolvidas no dia e quais foram os resultados obtidos no dia anterior. Três perguntas básicas sobre suas responsabilidades devem ser respondidas pelos *developers* durante a reunião diária, são elas: "o que foi feito ontem?", "o que vai ser feito hoje?"e "há algum obstáculo que impediu/impeça de realizar as atividades?".

Por fim, o *postgame* que conta com a preparação de documentação do projeto, incluindo documentação final, testes de pré-lançamento e lançamento.

4 ATIVIDADES DESENVOLVIDAS

Focado nas atividades realizadas durante o período de estágio no XMEN, essa seção consta de detalhamentos acerca das atividade desenvolvidas, podendo serem elas divididas em duas principais atividades: a verificação funcional de dois IPs (blocos) e as práticas de *scrum* realizadas para gerir o desenvolvimento da verificação.

4.1 VERIFICAÇÃO DE BLOCOS

Como dito na fundamentação teórica, um fluxo é necessário no desenvolvimento de hardware, assim sendo, após a especificação do bloco, códigos RTLs são criados pela equipe de *frontend* e após dados como finalizados, passados a equipe de verificação para que os mesmos sejam verificados e validados, caso contrário volta-se a equipe anterior e um ciclo é formado até que seja validado. Dois blocos foram propostos ao estagiário para que realizasse a verificação funcional: um bloco JTAG e um CRC.

4.1.1 VERIFICAÇÃO CÍCLICA DE REDUNDÂNCIA - CRC

Um dos blocos dispostos para a realização de verificação funcional do estagiário foi um bloco de CRC. Em blocos deste tipo, procuram-se realizar o cálculo de CRC de acordo com dado(s) de entrada, podendo serem de modo serial ou paralelo. O RTL verificado tem entrada paralela e realiza cálculo de CRC progressivo, ou seja, considerando por exemplo uma entrada de 8 bits e portanto um valor de CRC de 8 bits. Com palavra a ser calculada de 32 bits, quatro cálculos de CRC seriam realizados dependentes um do outro, para ao final obter um resultado de CRC de 8 bits. Dessa forma funcionava o bloco a ser verificado.

Para a realização da atividade em questão, foi necessário o estudo prévio sobre o cálculo de CRC, os tipos, os parâmetros e as diferentes formas de se realizar o cálculo do mesmo. A partir daí, realizou-se por meio de referências de outros VIPs o ambiente de verificação funcional do RTL em questão, utilizando-se da metodologia UVM e escrita em *SystemVerilog*.

Após o estudo sobre a verificação cíclica de redundância e a especificação do bloco desenvolvido, foi implementada sua arquitetura junto a criação dos planos de verificação e cobertura. Todos os componentes do ambiente em UVM foram idealizados e implementados pelo estagiário, aprimorados posteriormente com auxílio do líder da equipe.

Estimulado pela vontade de criar o seu próprio *testbench* sem a reutilização de VIP em busca de pôr em prática o seu conhecimento teórico adquirido quanto ao UVM, o

estagiário criou o *driver* capaz de estimular de forma correta o DUT, através da interface que conecta o RTL ao *testbench*. Criou também o *monitor*, capaz de transferir ao modelo de referência, tanto os dados utilizados para excitar o DUT quanto as respostas obtidas por esses estímulos. Contudo, o maior esforço empregado se deu no desenvolvimento do refmod.

Inicialmente foi utilizado da linguagem de programação C para criação do modelo de referência do bloco, contudo durante a fase de *debug*(depuração) do ambiente foi identificado que o mesmo não dava a resposta ideal que deveria, pois quando um valor em hexadecimal era passado do *SystemVerilog* para o código em C, ele era transformado em código ASCII para realização do cálculo ocasionando assim um cálculo errado. Sem conhecimento de que a conversão de hexadecimal para ASCII era um problema, outras formas de calcular o CRC foram exploradas para criação de um novo modelo até ser decidido que seria melhor trabalhar no modelo em C já existente. Após a realização de pesquisas e testes do modelo, foi identificado o problema que havia na conversão. Uma adaptação do modelo em C foi desenvolvido para *SystemVerilog* e após retomado a depuração do ambiente, constatou-se que a adaptação se adequava à verificação.

Finalizado o *testbench*, foi simulado o RTL dado os testes criados, verificando assim a presença de *matches* e *mismatches* entre o ambiente e o RTL. Com a presença de *mismatches* entre ambos, se fez necessária uma conversa entre a equipe de *design* e de verificação para descobrir se a causa era um *bug*(falha de código) na descrição de hardware ou se era no modelo criado para verificação do bloco. Após reunião entre as equipes, foi identificado que houve falta de informações quanto a forma que o bloco se comportaria, devido a ausência de documentação do mesmo, ocasionando assim, a geração de um modelo de referência errôneo. Uma vez que o modelo de referência foi consertado, novas simulações foram geradas, constatando que não havia mais *mismatches*.

Por fim, foi realizado as regressões do bloco - que nada mais é que uma bateria de simulações do RTL dado os testes do *testbench* criado - uma das fases mais problemáticas para o verificador devido a quantidade de erros que surgiram na execução das regressões. Contudo, com a colaboração da equipe de verificação o estagiário conseguiu solucionar todos os impasses e foi finalizada as regressões do bloco com 100% em todas as métricas de cobertura estipuladas, finalizando assim a verificação do bloco proposto. Na Figura 7 é possível ver a janela da ferramenta utilizada para realização das regressões com as métricas e percentagens atingidas ao fim delas.

4.1.2 *BOUNDARY SCAN* - JTAG

Assim como no caso do CRC, também foi proposto que o estagiário fizesse a verificação de um bloco JTAG. Seguindo as etapas da verificação, foi realizado o estudo aprofundado sobre o que era o JTAG e a técnica de *boundary scan*, também foi realizado

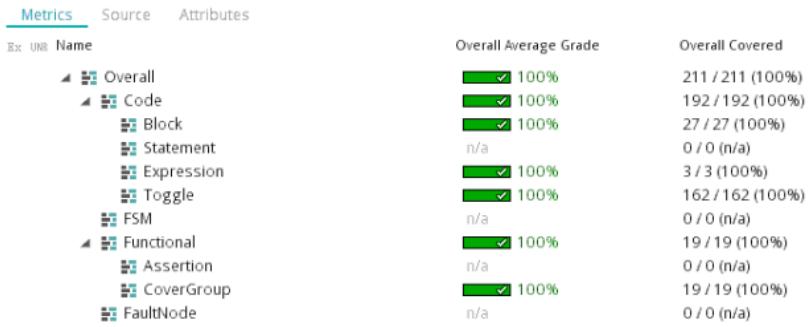

Figura 7 – Métricas de cobertura atingidas na ferramenta vManager

Fonte – Própria

um estudo sobre a interface AXI pelo qual se dava a comunicação entre o JTAG e os demais IPs do projeto ao qual ele estava inserido, para que a verificação do mesmo pudesse acontecer.

Seguindo o fluxo de atividades para a verificação de um bloco, foi estudada então a especificação do bloco através do *block guide* (documento que consta de todas as informações arquiteturais e funcionais do bloco) e posteriormente foi realizado o plano de verificação e plano de cobertura. Durante a fase de desenvolvimento do ambiente de *testbench* do bloco JTAG, o estagiário encontrou lacunas na documentação do RTL e solicitou ao *designer* responsável as informações necessárias para saná-las. Contudo, o *designer* não foi capaz de entregar as informações durante o período que aconteceu o estágio, tornando assim inviável a continuação do desenvolvimento da verificação do bloco.

4.2 PRÁTICAS DE *SCRUM*

Durante o período do estágio e para gerenciamento do projeto ao qual o bloco de CRC estava incluso, foi utilizado da metodologia *scrum*. A equipe de *scrum* contava com cinco integrantes, dos quais, três formavam o conjunto de *developers* (o estagiário se encontrava incluso nesse grupo), o *scrum master*, sendo este Klynger Renan e o *product owner*, Hugo Gayoso.

Os *product backlogs* foram criados pelos *developers* em conjunto com o *scrum master*, durante a criação dos *products*, foi adaptado e utilizado o *planning poker* para avaliar o nível das atividades. Uma vez que as macro-atividades do desenvolvimento foram criadas, as *sprints* iniciaram.

Os eventos de *sprints* tinham duração de uma semana e tinha seu início na terça-feira e fim na segunda-feira. No primeiro dia de cada evento os *developers* organizavam as atividades que iam realizar de acordo com o fluxo de desenvolvimento e tempo hábil para realização da atividade dentro da *sprint*. Na Figura 9 é mostrado como era gerenciado

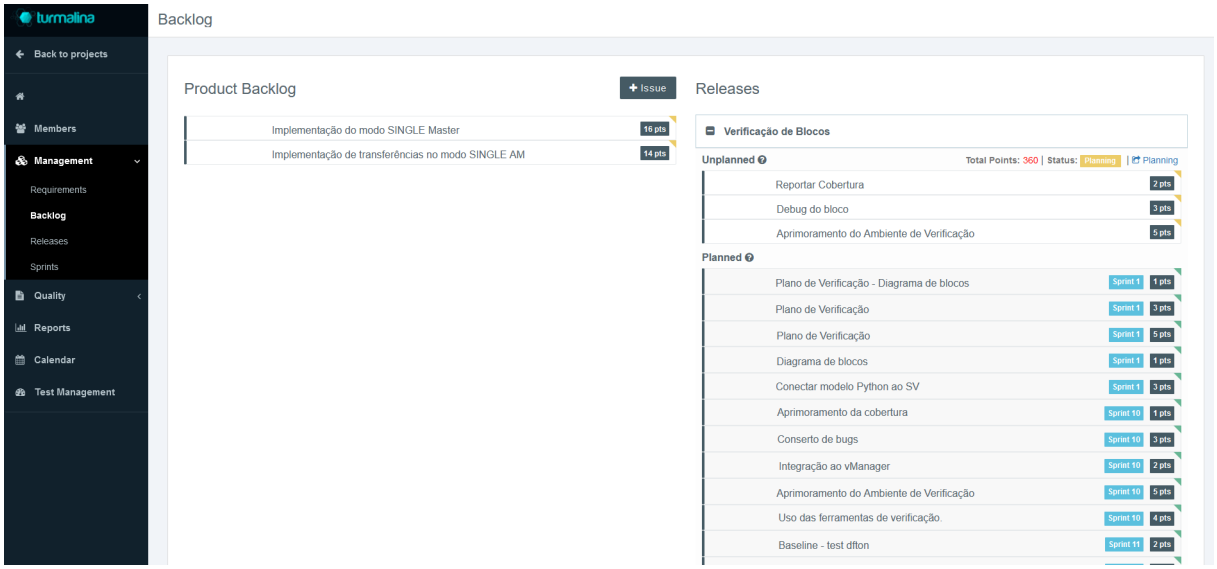

Figura 8 – Aba de *backlog* da plataforma Turmalina

Fonte – Própria

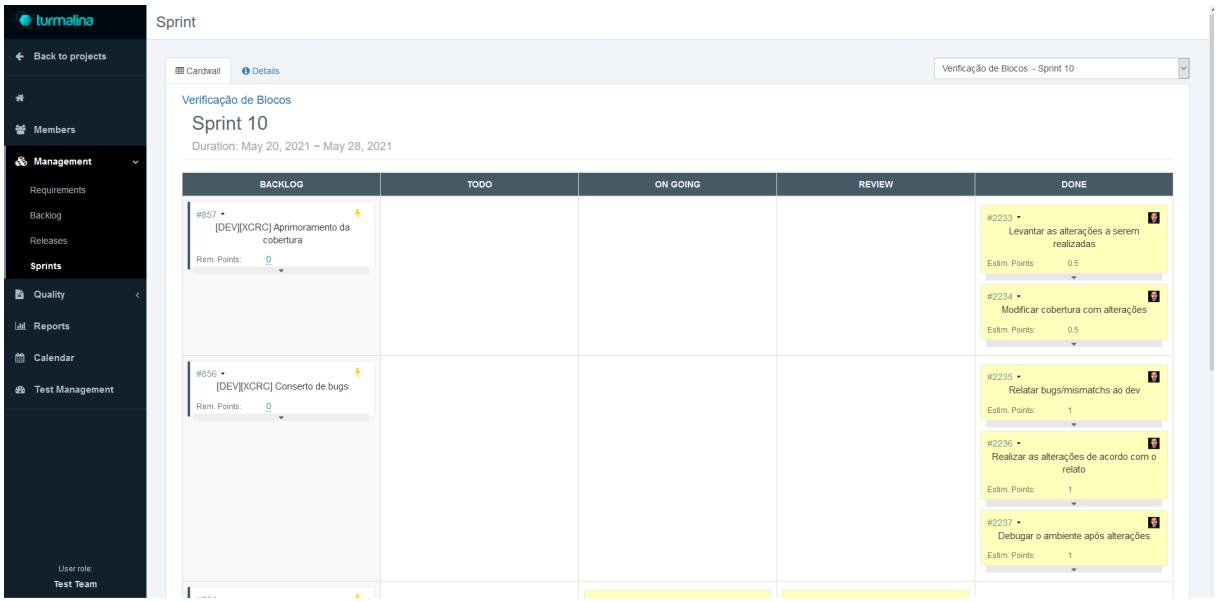

Figura 9 – Aba de *sprint* da plataforma Turmalina

Fonte – Própria

o desenvolvimento de atividades do *product backlog* durante uma *sprint* na plataforma Turmalina.

As atividades que seriam realizadas durante aquele evento tem seus status inicializados com *TO DO* que significa ser as atividades a serem feitas, dado o início da atividade o seu status é mudado para *ON GOING* e permanece assim até ser finalizada. Ao fim da realização da atividade ela é posta como em *REVIEW* para alguma eventual análise e correção, constatado então que a atividade foi de fato finalizada é colocada como *DONE*.

Para acompanhamento e auxílio durante as *sprints* eram realizadas reuniões diárias

que tinham duração de aproximadamente 15 minutos. O gerenciamento das reuniões também eram realizados por meio da plataforma Turmalina, onde era posto: os reportes de problemas encontrados durante a execução de alguma atividade e assuntos discutidos durante as mesmas.

5 CONSIDERAÇÕES FINAIS

Neste relatório foi possível explanar as atividades realizadas pelo estagiário, Vinicius Bonfim Araujo, durante todo o período de estágio, transparecendo todas as dificuldades que ocasionalmente ocorreram e a aptidão mostrado pelo mesmo para solucioná-las. O auxílio disposto por toda a equipe que forma o Laboratório de Excelência em Microeletrônica do Nordeste foi de fundamental importância para que a experiência presenciada pelo estagiário fosse enriquecedora e construtiva.

Conhecimentos obtidos durante as disciplinas de Introdução a Programação e Técnicas de Programação foram essenciais para o desenvolvimento dos modelos durante a verificação, bem como, as disciplinas de Circuitos Lógicos, Arquitetura de Sistemas Digitais e Arquiteturas Avançadas para Computação para a compreensão da metodologia UVM e a produção dos *testbenches* utilizados para a verificação funcional.

A realização do estágio no XMEN foi de grande privilégio ao discente, pois tornou possível o seu contato com ferramentas de verificação funcional de difícil acesso e a possibilidade de trabalhar com profissionais que possuem bastante experiência no ramo da eletrônica digital no país.

Por fim, o fato do laboratório usufruir de uma metodologia que permite um ótimo gerenciamento de desenvolvimento, foi essencial para que o estágio fosse concluído com a entrega de um bloco verificado e por fazer com que a parte remota do estágio não tivesse prejuízos durante sua duração.

REFERÊNCIAS

BERGERON, J. *Writing testbenches: functional verification of HDL models*. [S.l.]: Springer Science + Business Media, LLC, 2003. 15

BORRELLI, C. Ieee 802.3 cyclic redundancy check. 2001. Disponível em: <https://www-inst.cs.berkeley.edu/~cs150/fa04/Lecture/xapp209.pdf>. 13

CAMPOS, N. A basic tutorial of uvm. 2016. Disponível em: <https://sistenix.com/basic_uvm.html>. 17, 19

CARVALHO, B.; HENRIQUE, C.; MELLO, C. Scrum agile product development method - literature review, analysis and classification. *Product: Management & Development*, v. 9, p. 39–49, 01 2011. 19

IEEE Standard Test Access Port and Boundary Scan Architecture. *IEEE Std 1149.1-2001*, p. 1–212, 2001. 14, 15

SCHWABER, K. Scrum development process. In: SUTHERLAND, J. et al. (Ed.). *Business Object Design and Implementation*. London: Springer London, 1997. p. 117–134. ISBN 9781447109471. 20

SILVA, K. R. G. da. *Uma Metodologia de Verificação Funcional para Circuitos Digitais*. Tese (Doutorado) — Universidade Federal de Campina Grande, 2007. 16

SUTHERLAND, J.; SCHWABER, K. The scrum guide. *Retrieved 08-07-2021 from Scrumguides.org*, 2020. 19

TANENBAUM, A. *Redes de computadores*. Elsevier, 2003. ISBN 9788535211856. Disponível em: <https://books.google.com.br/books?id=0tjB8FbV590C>. 14