UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Eliseu Gonçalves de Arruda

ESTÁGIO INTEGRADO NA IDEA! ELECTRONIC SYSTEMS

CAMPINA GRANDE

2019

LUCAS ELISEU GONÇALVES DE ARRUDA

ESTÁGIO INTEGRADO NA IDEA! ELECTRONIC SYSTEMS

Relatório de Estágio Integrado submetido à Universidade Federal de Campina Grande, como requisito necessário para obtenção do grau de Bacharel em Engenharia Elétrica

Campina Grande, novembro de 2019

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

LUCAS ELISEU GONÇALVES DE ARRUDA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Engenharia de Elétrica, sendo aprovada em sua forma final pelo orientador:

Prof. Dr. Marcos Ricardo Alcântara Morais
Universidade Federal de Campina Grande

Campina Grande, 18 de novembro de 2019

Agradecimentos

Inicialmente agradeço aos meus pais, que sempre me apoiaram em todas as minhas escolhas e me deram as melhores oportunidades de aprendizado possível. Vejam só estou me tornando um engenheiro eletricista e não um juiz!

Agradeço a Sara por estar sempre presente, mesmo em momentos de dificuldades, e me fazer levantar a cabeça quando eu não acreditava em mim. Obrigado, bem!

Em seguida, sou grato pela maravilhosa iniciativa dos professores Gutemberg e Marcos Morais pela criação do Laboratório de Excelência em Microeletrônica no Nordeste (XMEN). Esta experiência que eu tive a oportunidade de vivenciar além de desafiadora, foi inspiradora e enraizou laços de amizade com os seus membros que irei carregar para o resto da vida.

Por falar neles, muito obrigado pelos momentos de riso, raiva e gritaria que passamos juntos!

Finalmente, agradeço a Idea! por ter me oferecido todos os meios para realização deste trabalho.

"Uma design house é uma lanhouse que faz chips nas horas livres"
(Autor desconhecido)

Lista de ilustrações

Sumário

1 Introdução

Este relatório tem como objetivo descrever as atividades desenvolvidas durante o período de Estágio Integrado realizado na empresa Idea! Electronic Systems do dia 07 de janeiro de 2019 a 14 de novembro de 2019 totalizando uma carga horária de 1880 horas.

O estágio em questão foi realizado no setor de Microeletrônica da referida empresa, especificamente na área de verificação, na qual foram atribuídas ao estagiário as seguintes atividades:

- Estudo e familiarização com o fluxo de referência da Idea! para projeto lógico dos circuitos digitais e estratégias de verificação;
- Revisão dos principais artigos da área de sistemas ópticos com detecção coerente e algoritmos de processamento de sinais referentes aos blocos a serem verificados duranten o estágio;
- Modelagem em ambiente SystemVerilog para validação funcional e estrutural dos algoritmos de DSP e protocolos a serem atribuídos ao aluno;
- Estudo e teste dos modelos arquiteturais para validação funcional e estrutural;
- Verificação dos circuitos digitais seguindo a linguagem de descrição de *hardware* de referência usado na empresa (SystemVerilog) em conjunção com a metodologia de verificação UVM;
- Participação no desenvolvimento de um gerador de *testbenchs* UVM;

Essas atividades foram desenvolvidas sob supervisão dos coordenadores da área de Verificação e Protocolo Marcelo Guedes e Carlos Castro e do diretor de Microeletrônica Jacklyn Dias Reis, sob orientação do professor doutor Marcos Ricardo Alcântara Morais.

1.1 Idea! Electronic Systems

A Idea! Electronic Systems, localizada em Campinas - SP, é uma empresa de serviços de alta tecnologia focada na evolução do estado da arte das áreas de microeletrônica e fotônica, focando na fusão entre essas duas áreas (Idea! 2019).

A empresa vem alcançando as condições para atingir suas metas investindo em um time de engenheiros altamente capacitados (Figura 1), firmando parcerias com empresas de tecnologia de ponta (Figura 2) e investindo fortemente em infraestrutura, com salas

limpas para confecção de lasers, equipamentos, servidores e ferramentas de automação eletrônica (*electronic design automation* - EDA).

Figura 1 – Time Idea! Fonte: https://www.idea-ip.com/about-us/, acesso em nov. 2019

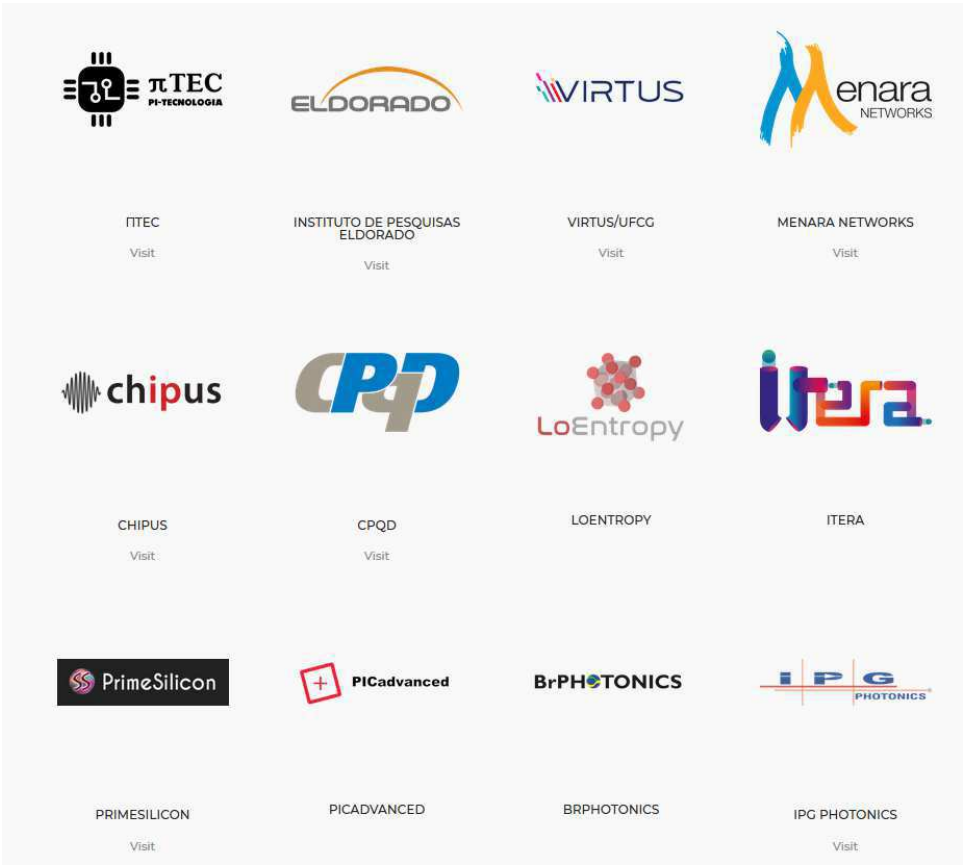

Figura 2 – Parceiros Idea! Fonte: https://www.idea-ip.com/partners/, acesso em nov. 2019

Seus projetos, focados na área de transmissão digital da informação, colaboram para suprir a crescente demanda no tráfego de dados digitais. Esses avanços estimulam o

que há de mais inovador no campo de processamento de sinais, sendo um desafio motivador para as equipes.

Neste contexto, a empresa desenvolve tecnologias voltadas para as comunicações ópticas, como processadores digitais de sinais (DSPs), lasers sintonizáveis e amplificadores ópticos.

2 Embasamento Teórico

2.1 Sistemas de Comunicação Óptica Coerente

Os sistemas de comunicação óptica são projetados baseados em dois tipos principais de detecção no receptor: a detecção direta e a detecção coerente. Na direta, apenas a intensidade do sinal é utilizada, de modo que fotodiodos realizam a conversão do domínio óptico para o elétrico.

Já na detecção coerente, a informação pode estar distribuída em mais características do sinal óptico, a saber a amplitude, a frequência, a fase e a polarização. Esse sinal é convertido para o domínio digital por meio de um conversor analógico-digital (*analogic to digital converter* - ADC), que é processado por um DSP para compensação dos efeitos de sua propagação pela fibra (Zhou e XIE 2016).

2.2 DSP Aplicado em Sistemas Ópticos Coerentes

Como mencionado anteriormente, o DSP apresenta a função de corrigir os diversos efeitos aplicados ao sinal ao longo do processo de transmissão, incluindo efeitos de propagação na fibra óptica e efeitos de amostragem do sinal.

É apresentado na Figura 3 um diagrama contendo as principais funções que o DSP realiza para restauração do sinal recebido.

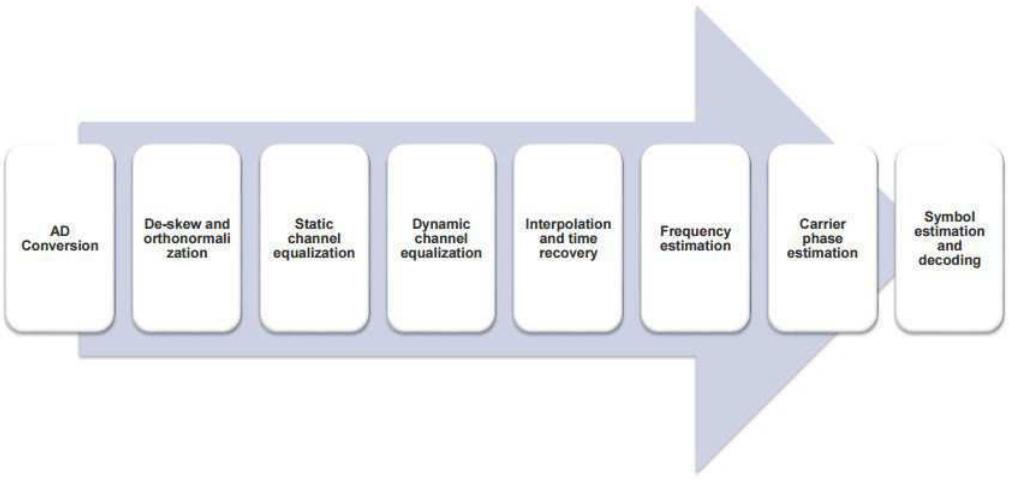

Figura 3 – Funções do DSP em sistemas ópticos coerentes. Fonte: Formiga 2017

Inicialmente o sinal é convertido para o domínio elétrico (não representado na Figura 3), sendo submetido a um ADC, que é responsável por amostrar e quantizá-lo para

a quantidade de bits estabelecida na arquitetura-alvo. Esse componente é responsável por inserir o efeito de inclinação (skew) no sinal, que é tratado pelo segundo bloco apresentado na Figura 3.

Um dos principais efeitos inseridos pela fibra é a dispersão cromática, que ocorre devido ao fato de que a velocidade de propagação da luz em um meio depende de sua frequência. Esse efeito é compensado por meio de um equalizador estático, que aplica atrasos nas diferentes frequências de modo a sincronizá-las. Esse componente não compensa completamente a dispersão cromática, deixando passar uma dispersão cromática residual, que é corrigida por um equalizador dinâmico também responsável por compensar a dispersão de modo de polarização (*polarization mode dispertion* - PMD). Estes dois equalizadores são representados como terceiro e quarto blocos da Figura 3.

Em seguida, há um módulo de correção de erro de *timing*, isto é, quando um símbolo amostrado não é recebido na quantidade de *clocks* esperada, que é seguido por um bloco para estimar o desvio de frequência entre o sinal do laser transmissor e o oscilador local para posterior correção e outro para estimação e correção de fase do sinal. O processo de recuperação do sinal é finalizada por um módulo de estimação e decodificação do símbolos em uma sequência de bits.

Esse processo é ainda ilustrado na Figura 4 acompanhado pela ilustração do sinal em cada etapa no caso de modulações QPSK e 16QAM.

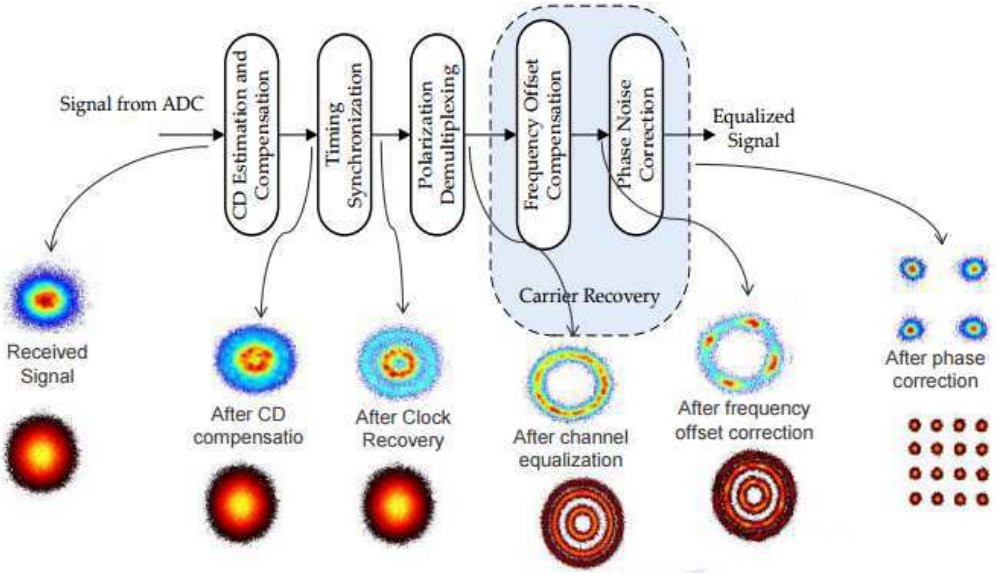

Figura 4 – Representação do processamento do sinal. Fonte: Formiga 2017

2.3 Conceitos Gerais

Para melhor contextualização sobre o trabalho da verificação na área da microeletrônica e para evidenciar os termos utilizados ao longo do texto, serão explicados conceitos

fundamentais do setor.

2.3.1 Linguagem de Descrição de Hardware (HDL)

As HDLs são linguagens de programação utilizadas principalmente para descrever o comportamento e a estrutura de circuitos eletrônicos, comumente circuitos digitais. Esta descrição pode apresentar dois níveis de abstração: o nível lógico e o nível de transferência de registros (*register transfer level* - RTL).

No nível lógico, o circuito é descrito detalhadamente levando em consideração cada porta lógica e é impraticável de ser realizado quando a complexidade dos componentes aumenta. Já em RTL, os dados são processados no caminho de dados (*datapath*), sendo submetidos a blocos que abstraem o detalhamento das portas lógicas como registradores, somadores e multiplexadores. Isso facilita o desenvolvimento de lógicas complexas constituídas de milhões de transistores e diminui consideravelmente a probabilidade de erros, pois não se faz necessário analisar todas as portas lógicas individualmente.

Essas linguagens também apresentam suporte para controle de simulações pois explicitamente inclui noção de tempo, isto é, tornam possível a criação de ambientes virtuais (*testbenchs*) nos quais estímulos podem ocorrer ao longo do tempo simulado.

No âmbito dos circuitos digitais, as HDLs mais utilizadas são o *systemverilog* e o VHDL.

2.3.2 Verificação Funcional

O processo de desenvolvimento de IPs digitais é composto por diversas etapas. Inicialmente, existe uma especificação de seu funcionamento, podendo conter restrições de consumo de energia e temperaturas de funcionamento. Em seguida, estas orientações podem ser implementadas em linguagens de alto nível como Python ou MATLAB, que apresentam precisão infinita e podem abstrair as restrições impostas na especificação.

A transição de modelo em alto nível para implementação em hardware implica a aplicação de diversos compromissos como a diminuição de bits de precisão, a inclusão de níveis de *pipeline* para atingir critérios de temporização e a simplificação de circuitos para redução do tamanho do circuito para atingir critérios de área por exemplo.

Nesse contexto, a verificação funcional pode ser definida como a demonstração de que a intenção de um *design* é mantida em sua implementação [Piziali 2004].

Ela é estruturada por um plano de verificação responsável por determinar três aspectos principais:

- **Análise de cobertura**: Determina a extensão dos testes e os casos especiais em

que o *design* precisa ser submetido para validação;

- **Geração de estímulos**: Define que tipos de estímulos precisam ser enviados ao dispositivo de modo a garantir condições de funcionamento e fechamento de cobertura;
- **Comparação dos resultados**: Estabelece os critérios para avaliação da funcionalidade do DUT.

Em termos práticos, a verificação se consolida por meio da criação de um ambiente de virtual denominado *testbench*, que apresenta sub-blocos responsáveis por enviar e receber estímulos do DUT e modelo e compará-los. Um exemplo de *testbench* é apresentado na Figura 5.

Um dos elementos principais no *testbench* é o *agent*, que pode conter um *sequencer*, um *driver* e *monitors* quando classificado como ativo (isto é, quando envia sinais para DUT/modelo). Quando dotado apenas de *monitors*, este elemento é classificado como passivo. No primeiro caso, sequências de estímulos abstraídos, denominados transações, são geradas pelo *testbench* e conduzidas ao *driver* por meio do *sequencer*. O *driver* é responsável por converter essas transações para sinais de interface do DUT/model, podendo implementar os mais diversos protocolos de comunicação.

Ainda no *agent*, o *monitor* é responsável por monitorar os sinais de interface do DUT/modelo e convertê-los para transações, que são enviadas para os *scoreboards*, que têm função de checar se DUT e modelo estão coerentes entre si.

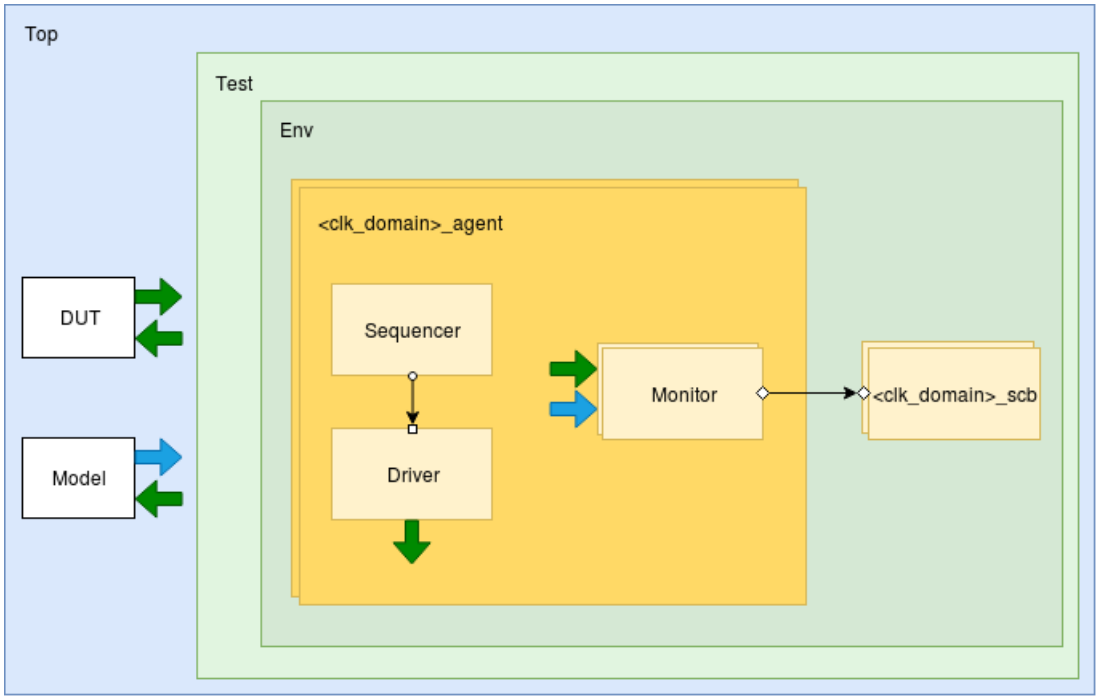

Figura 5 – Arquitetura do *testbench* deste trabalho

2.3.3 Transaction Level Modeling (TLM)

Um dos pontos para aumentar a produtividade em verificação é de trabalhar com níveis de abstração que façam sentido [Accellera 2015], isto é, não faz sentido ter que implementar todos os sinais de algum protocolo para realizar ações simples. Uma leitura em SPI, por exemplo, poderia ser representada apenas pelo dado, endereço e modo de operação (leitura).

A UVM dispõe de componentes capazes de expressar esse nível de abstração, que são as transações (*sequence_items*) e as sequências (*sequences*), que foram explicados anteriormente.

3 Atividades Desenvolvidas

3.1 Verificação de Blocos

3.1.1 Bloco de multiplicação

Um bloco cuja função era apenas multiplicar duas entradas foi verificado durante o período de estágio como forma de familiarização com o fluxo de verificação. O *testbench* foi gerado pelo aluno com base na Figura 5 vários detalhes de implementação que só são atentados na prática foram observados e contornados com êxito. Além disso, um modelo deste componente foi gerado e incorporado ao *testbench*.

3.1.2 Blocos de filtragem

Os primeiros blocos mais complexos a ser verificados envolveram filtros FIR. As aplicações desses blocos foram variadas, incluindo a realização de pré-enfase, recuperação de *timing* e aplicação de atrasos aos sinais de entrada.

A metodologia utilizada para a verificação destes blocos consistiu em garantir que os estímulos de entrada não causassem saturação na saída. Para isto, utilizou-se a métrica apresentada em 3.1, ou seja, a energia das entradas multiplicada pela energia dos coeficientes deve ser menor que a máxima energia da saída. Essa métrica limita os valores das entradas de modo a evitar a saturação da saída, evitando testes que resultem apenas em valores saturados.

$$E_{in} \cdot E_{coefs} < E_{out}^{max} \tag{3.1}$$

3.1.3 Bloco de cruzamento de clock

Um elemento chave em grandes projetos de microeletrônica digital são os blocos de cruzamento de *clock* (*clock domain crossing* - CDC), que são responsáveis por modificar a cadência de dados de um domínio de *clock* para outro. Inicialmente, estes blocos iriam ser verificados apenas em nível de topo, ou seja, nos ambientes de IPs que instanciassem o CDC. No entanto, constatou-se a necessidade de realizar a verificação individual desses blocos devido a recorrência de incongruências de sinais relacionados à troca de domínio de *clock*.

Essas inconsistências foram apenas observadas em *gate level*, isto é, após a tradução do RTL para portas lógicas mapeadas no nó tecnológico utilizado na empresa e, por isso, foi necessário realizar a síntese lógica dos CDCs para cada par de frequências utilizados no projeto. Isso deu origem a um *script* em Python responsável por modificar os parâmetros adequados.

Após conclusão das sínteses, foi possível reproduzir os erros observados em nível de topo, facilitando o processo de depuração de código para correção das inconsistências. Uma vez corrigidas isoladamente, a correção foi aplicada nos níveis de topo como forma de um *engineering change order* (ECO) em suas netlists.

3.2 Implementação de Gerador de Testbench

Em paralelo à verificação dos blocos, foi desenvolvida a atividade de implementar um gerador de *testbenchs* em UVM, sendo uma forma de aumentar a produtividade da equipe de verificação.

Os blocos a serem verificados são dispostos em planilhas nas quais cada sinal da interface está discriminado juntamente com as respectivas quantidades de bits. No caso de blocos que instanciam sub-blocos, as planilhas também indicam as conexões entre sinais da interface externa com os blocos internos, no entanto este tipo de circuito não é abordado neste trabalho.

A partir dessas planilhas, o *parser*, que é um objeto desenvolvido pela Idea!, disponibiliza para o gerador todos os sinais de interface também com suas quantidades de bits, e a partir desse ponto o gerador passa a criar hierarquicamente os objetos de sua estrutura. É apresentado nas Figuras 6 e 7 o diagrama de classes do gerador. Vale ressaltar que foram destacadas apenas as funções mais importantes por simplicidade de representação.

Esse diagrama é construído com inspiração na arquitetura de *testbench* implementada pelo gerador (Figura 5). Um objeto *tb* é criado recebendo como parâmetro um objeto do tipo *parser* que, como mencionado acima, contém todos os dados necessários para caracterizar os sinais do DUT. Este objeto é responsável por criar outros quatro: *o param_ obj*, que gera um arquivo contendo todas as informações relacionadas a quantidade de bits; o *pkg*, que cria um pacote de SystemVerilog incluindo todos os elementos UVM; o *dut*, que é um objeto que oferece funcionalidades auxiliares para o TbGen; o *interface*, que além de ter informações sobre os sinais, é responsável pela escrita das definições de *interface*, sendo gerado um objeto desse tipo para cada domínio de *clock*; e o *test*.

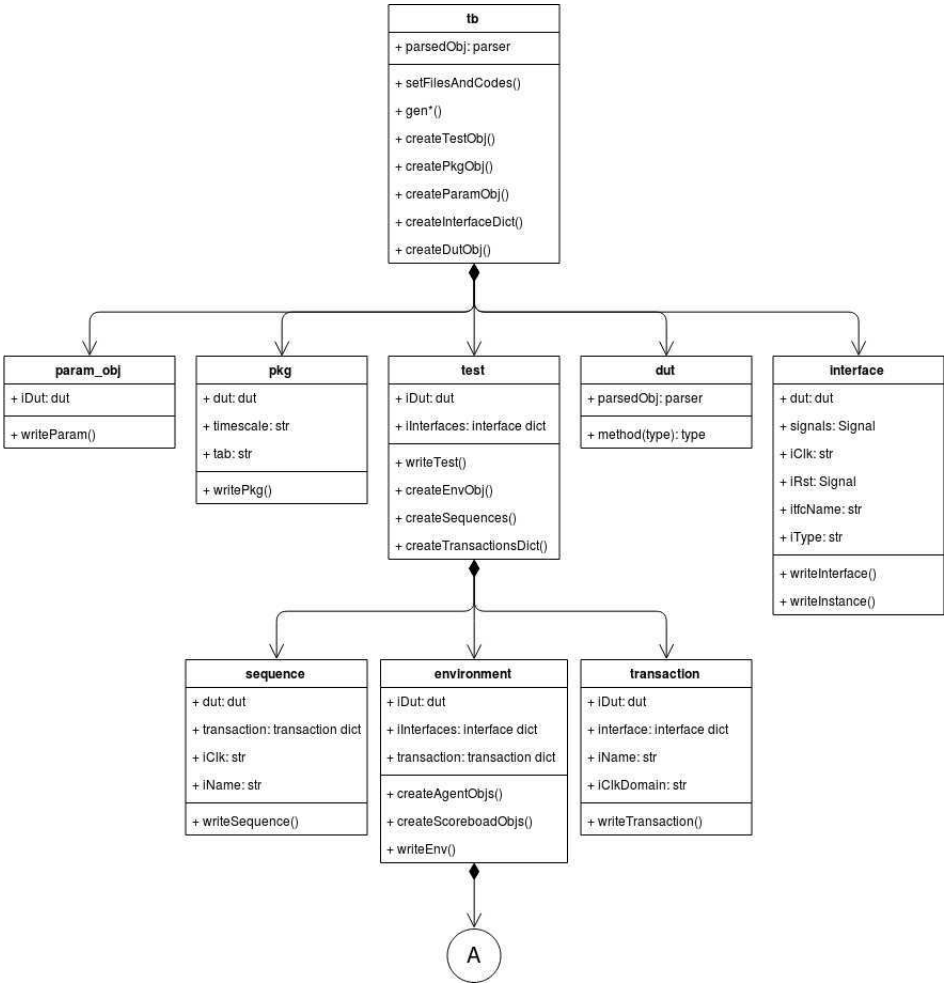

Figura 6 – Diagrama de classes do TBGen. Fonte: Autoria própria

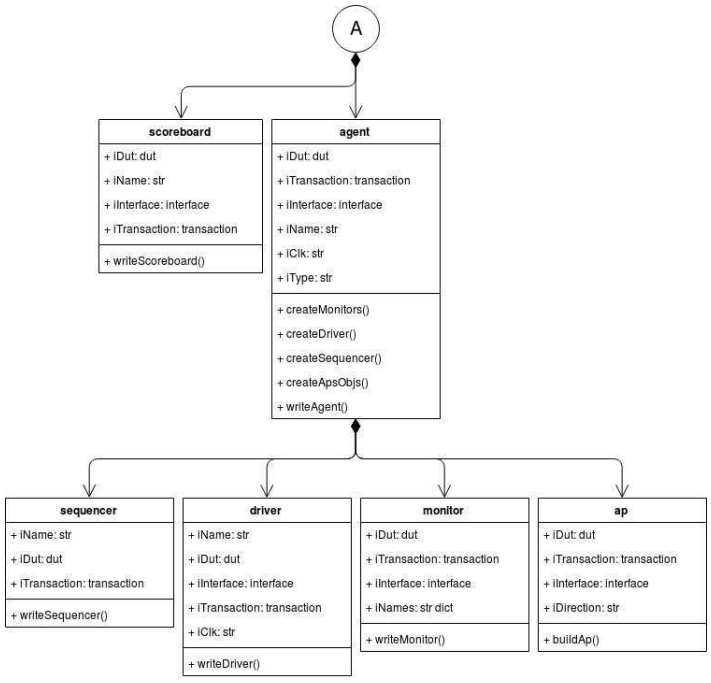

Figura 7 – Continuação do diagrama de classes do TBGen. Fonte: Autoria própria

Esse último objeto além de escrever o *test*, também agrupa objetos *interface* em um dicionário para criar três objetos: o *sequence*, responsável pela escrita da sequência (uma por domínio de *clock*); o *transaction*, que escreve o *sequence_ item* (também uma para cada domínio de *clock*); e o *environment*.

O *environment* além de escrever seu componente homônimo, cria um dicionário com as *transactions* e utiliza o de *interfaces* que foi passado pelo *test* para criar objetos do tipo *scoreboard* e *agent*, também um para cada domínio de *clock*.

Finalmente, cada *agent* é responsável pela criação de quatro tipos de objetos: *sequencer*, *driver*, *monitor* e *ap* (*analysis port*).

Cada objeto criado contém um método de escrita write* do seu componente homônimo (writeTest, writeInterface, etc). Esses métodos são mapeados hierarquicamente no *tb* pelas funções gen* (genTb, genTest, genInterface, etc). Dessa forma, o tb pode executar a função setFilesAndCodes(), que retorna um dicionário contendo o nome de cada componente como chave e o texto equivalente aos componentes como valor. Isso permite que um programa que instancie o *tb* possa escrever os arquivos correspondentes a cada elemento.

Os códigos de componentes em UVM apresentam muitas funções que se repetem com frequência, contendo apenas alguns campos onde alterações são necessárias. Por esse motivo, foram criados *templates* de cada componente contendo indicadores especiais nas regiões a serem modificados. Esses indicadores são etiquetas definidas da seguinte forma: {:*etiqueta*:}. Os *templates* são carregados antes das chamadas das funções write*, que utilizam o método de Python str.replace('{:*etiqueta*:}', variável) para sobrescrever localmente o *template* e gerar o código desejado.

Alguns elementos que apresentam códigos bastante mutáveis como as interfaces não utilizam *templates* para escrita, passando a ser escritos linha a linha nas funções write*.

4 Considerações Finais

As atividades programadas para o período de estágio foram executadas com êxito, oferecendo ao estagiário diversos conhecimentos na área de microeletrônica, processamento de sinais e verificação de IPs.

A empresa concedente teve um papel importante neste período, desde a determinação do plano de trabalho ao apoio técnico e estrutural, oferecendo um ambiente propício para o desenvolvimento de todos os envolvidos no projeto, promovendo constantes trocas de ideias e discussões profissionais.

Durante o estágio foi possível por em prática conhecimentos adquiridos nas disciplinas de graduação Técnicas de Programação, Circuitos Lógicos, Arquiteturas de Sistemas, Análise de Sinais, Processamento Digital de Sinais e Princípios de Comunicação e em atividades extracurriculares, particularmente os projetos desenvolvidos junto ao Laboratório de Excelência em Microeletrônica no Nordeste (XMEN). Mesmo com toda essa base, alguns conceitos utilizados durante o estágio não foram passados em nenhuma matéria, reiterando a importância do estágio para a formação de um engenheiro.

Referências

ACCELLERA, S. I. Universal verification methodology (uvm) user's guide. *Estados Unidos*, 2015. Citado na página 21.

FORMIGA, D. A. Estágio integrado na brphotonics. *Relatório de Estágio (Engenharia Elétrica), UFCG, Campina Grande Brasil*, 2017. Citado 2 vezes nas páginas 17 e 18.

IDEA! *Idea! Converging Photonics & Microelectronics*. 2019. Disponível em: <www.idea-ip.com>. Acesso em: 18 fev. 2019. Citado na página 13.

PIZIALI, A. Functional verification coverage measurement and analysis. *Kluwer Academic Publishers, Nova Iorque, Estados Unidos*, 2004. Citado na página 19.

ZHOU, X.; XIE, C. Enabling technologies for high spectral-efficiency coherent optical communication networks. *John Wiley and Sons*, 2016. Citado na página 17.