Saul Borges Nobre

Estágio Supervisionado na Idea!

Campina Grande, Brasil

25 de dezembro de 2018

Saul Borges Nobre

Estágio Supervisionado na Idea!

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título 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

Área de Concentração: Microeletrônica

Orientador: Gutemberg Gonçalves do SantosJunior, D.Sc.

Supervisor: Jacklyn Dias Reis, D.Sc.

Campina Grande, Brasil

25 de dezembro de 2018

Saul Borges Nobre

Estágio Supervisionado na Idea!

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Bacharel em Ciências no domínio da Engenharia Elétrica.

Aprovado em:

Gutemberg Gonçalves do SantosJunior, D.Sc.
Orientador

Marcos Ricardo Alcântara Morais, D.Sc.
Convidado

Campina Grande, Brasil
25 de dezembro de 2018

Dedico este trabalho aos meus amados pais, José Borges da Silva Filho e Maria Betânia Nobre Costa Borges, à minha família e a minha namorada Nathália Guerra de Sousa

Agradecimentos

Primeiramente, agradeço a Deus pelo dom da vida. Em seguida, aos meus pais, José Filho e Maria Betania, por todo o apoio e suporte que me deram nessa longa jornada ate aqui e por sempre acreditarem que eu seria capaz.

À minha irmã, Débora, que embora não ter convivido comigo nos últimos anos, agradeço por ter me feito, inconscientemente, evoluir tanto no meu lado profissional quanto no lado pessoal.

À minha namorada, Nathália, agradeço por sempre ter estado comigo nas horas que precisei, sempre ter me ajudado a levantar quando cai, sempre ter me incentivado a ser melhor e evoluir. Agradeço por ter sido minha companheira e confidente, e acima de tudo, ter sido paciente.

Agradeço aos Pseudosmitos, grupo de estudantes, que sempre estiveram presentes ao longo dessa caminha, em varias noites de estudos e momentos de descontração. Agradeço , também, por todas as risadas que foram dadas, por todo apoio e incentivo que nos demos ao longo dessa jornada.

Agradeço aos grupos que participei ao longo da graduação, PET, RAMO Estudantil IEEE, X-MEN Lab, que me trouxeram engradecimento profissional e pessoal.

Aos meus professores, por todos os ensinamentos e oportunidades que me foram dadas, em especial ao professor Gutemberg, que me orientou nessa reta final e sempre acreditou no meu potencial.

"First, have a definite, clear practical ideal; a goal, an objective. Second, have the necessary means to achieve your ends; wisdom, money, materials, and methods. Third, adjust all your means to that end"

Aristotle

Resumo

Neste relatório, descrevem-se as atividades que foram realizadas durante o período de estágio supervisionado na área de Microeletrônica da Idea!. O foco deste trabalho foi na verificação de blocos desenvolvidos pela empresa e na implementação de metodologias para um aumento na eficiência da verificação implantada na empresa. Para que isto fosse possível, foi necessário o entendimento do funcionamento de um Processador de Sinais Digitais (DSP) como um todo e um estudo mais profundo quanto à funcionalidade dos blocos verificados.

Palavras-chaves: Idea!; Verificação; DSP; Microeletrônica; Sistemas Ópticos Coerentes.

Abstract

In this report, were discribed the activites performed during the internship at Idea!' Microelectronics team. The main goal of this work was the verification of block that are been developing by Idea! and to increase the eficiency of the verfifcation team. To achieve that main goal, was necessary to understand the overall operation of a DSP used on an application of processing the signal from optical system and specialize on the knowledge of the funcitonality of the block that was verified.

Key-words: Idea!; DSP; Verification; Microelectronics Coherent Optical Systems.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Este relatório tem como objetivo descrever as atividades desenvolvidas pelo estudante Saul Borges Nobre durante o período de estágio integrado realizado na Idea! do dia 26/07/2018 ao dia 17/12/2018.

O estágio em questão foi realizado no setor de Microeletrônica da Idea! e ao estagiário foram atribuídas as seguintes atividades:

- Compreender o funcionamento dos principais blocos que compõem os modelos de Processadores de Sinais Digitais (DSP – *Digital Signal Processors*) de um transceptor coerente para comunicação em alta velocidade;
- Compreender o processo de verificação de um IP utilizado na empresa;
- Realizar a implementação de um *wrapper* de comunicação entre *SystemVerilog* e Python, para que fosse utilizado pelo time de verificação;
- Ajudar na confecção de um gerador de ambiente de verificação que atendesse às necessidades do time de verificação;
- Verificar blocos dos projetos em que a empresa está trabalhando.

Além desses pontos apresentados, o aluno pôde prestar suporte aos demais membros da equipe de verificação sobre o uso dos *wrappers* que estavam sendo implantados e de funções e utilidades para o ambiente que eram requisitada pelos verificadores. Tudo isso foi realizado com a supervisão do líder do time de verificação Roberto Borgognoni e do gerente de Microeletrônica, Jacklyn Dias Reis, sob orientação de Gutemberg Gonçalves do Santos Júnior.

1.1 Idea! Electronic Systems

Idea! Electronic Systems é uma empresa de alta tecnologia com foco em aplicações de fotônica e microeletrônica em seus estados da arte para sistemas de comunicação óptica de alta velocidade (100G, 400G, 1T). Com foco no tráfego da informação digital, a empresa colabora direta ou indiretamente com o atendimento à uma demanda crescente do consumo de informação digital. Com isso, estimulando o desenvolvimento de novas tecnologias e novas aplicações no setor como, por exemplo, *Internet of Things (IoT)*, aplicações em *Data Sciense*, *Data Centers* etc.

Figura 1 – Logo da Idea!

A empresa possui uma rede de colaboradores que abrangem quase 100 pessoas, desde dos setores administrativos até os setores técnicos. Estes colaboradores estão espalhados entre as instituições associadas, que abrangem tanto empresas como universidades, a Figura 2 tem alguns exemplos destas instituições. Estas parcerias possuem funcionalidades diversas, como treinamento de pessoas, compartilhamento de quadro técnico e instalações, salas limpas e laboratórios de teste, mostrando que a empresa não busca só ter o conhecimento técnico, visando também a formação de novas pessoas especializadas. Dentre as tecnologias desenvolvidas na empresa, destacam-se: *Application Specific Integrated Circuits* (ASICs), *Photonic Integrated Circuits* (PICs), laser sintonizáveis e amplificadores ópticos.

Figura 2 – Parceiros da Idea!

A importância relativa da tecnologia de fibra óptica é evidenciada pela sua velocidade e eficácia na modelagem da infraestrutura de comunicações em tão curto período, como vem ocorrendo nos últimos anos (ZADEH, 2004). O mercado de comunicações ópticas está se expandindo, conforme evidenciado pelo crescimento nas receitas e margens dos fornecedores (SOCIETY, 2018). Como pode ser observado na Figura 3, a indústria de lasers vem crescendo constantemente há mais de cinco décadas, com valores de vendas anuais de aproximadamente 10 bilhões de dólares atualmente. Além disso, as empresas estão investindo para aumentar a capacidade de produção e as perspectivas de crescimento são boas (SOCIETY, 2018).

Com o gráfico mostrado na Figura 3, pode-se notar que a Idea! está inserida em um mercado promissor e que deve ter um aumento linear durante as próximas décadas.

Figura 3 – Crescimento do mercado de laser nas últimas décadas

Assim, contribuindo com desenvolvimento de um mercado em expansão e que tem como fim produtos de tecnologia de ponta.

2 Fundamentação Teórica

2.1 Sistema de Comunicação Óptica Coerente

Quando se transmite dados em um sistema de comunicação óptica, tem-se dois métodos de se realizar a detecção no receptor: a direta ou a coerente. Na direta, utiliza-se de fotodiodos para converter o sinal do domínio óptico para o elétrico, e as informações do sinal são recuperadas a partir da amplitude do sinal óptico recebido.

Em contrapartida, na detecção coerente, a informação é extraída a partir de suas características: amplitude, frequência, fase e polarização. Para recuperar tais informações do sinal é utilizado um conversor analógico-digital (ADC - *Analog to Digital Converter*) que irá amostrar o sinal, para que seja processado por um DSP (SEIMETZ, 2009). Este irá compensar digitalmente os efeitos da propagação do sinal na fibra, que acabam degradando o sinal (ZHOU; XIE, 2016).

2.2 DSP Aplicado em Sistemas Ópticos Coerentes

O principal objetivo da utilização de um DSP em sistemas ópticos coerentes é a compensação de efeitos adversos advindos da fibra óptica e do transceptor. Uma topologia que pode ser utilizada de DSP pode ser encontrado na figura 4.

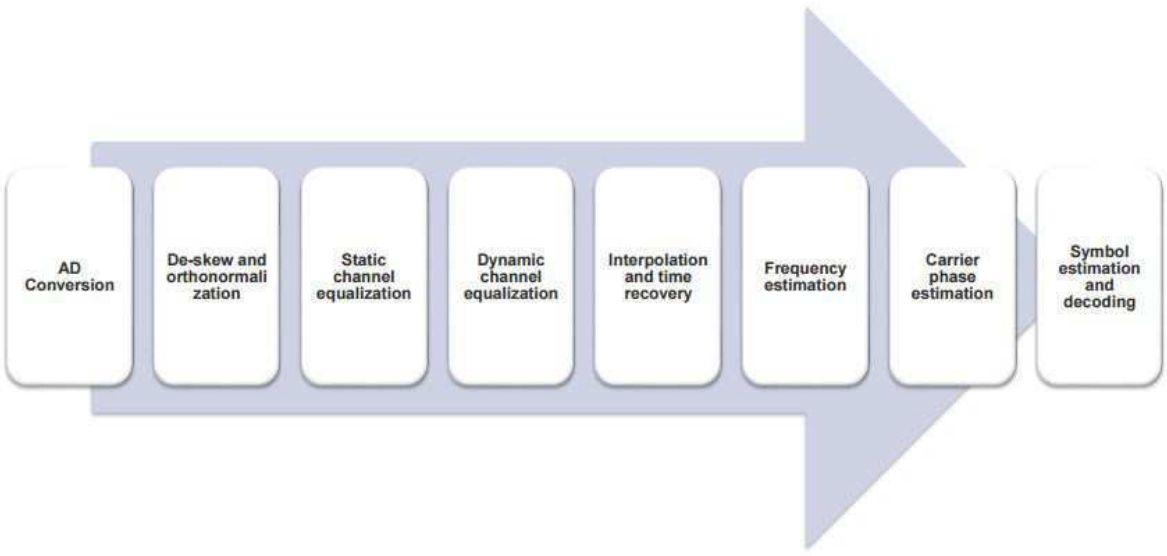

Figura 4 – Esquema arquitetural do receptor de um DSP voltado para sistemas ópticos coerentes

O fluxograma apresentado na Figura 4 é formado por blocos com funções específi-

cas no DSP, os quais blocos, em microeletrônica, são chamados de propriedades intelectuais (IP - *Intelectual Properties*), que no nível de simulação podem ser implementadas em linguagens de alto nível, como Python ou MATLAB e, a nível de hardware em linguagem de descrição, no caso da Idea!, *SystemVerilog*.

Analisando o lado do receptor, o primeiro bloco é um conversor de domínio analógico para digital, o qual é responsável por amostrar e quantizar o sinal elétrico recebido. Este primeiro componente, em seu funcionamento, acaba inseririndo o efeito *skew*, que trata-se de um atraso temporal entre a as duas polarizações do sinal.

Assim, para correção desse efeito, é inserido o segundo bloco da Figura 4, que irá compensar o atraso temporal inserido pelo bloco anterior, e ainda corrigir uma possível alteração na ortogonalidade das polarizações.

Um dos principais efeitos introduzidos pela fibra é a dispersão cromática. Para corrigir o referido efeito é inserido um equalizador estático que tem seus coeficientes calculados partindo de informações da transmissão, como comprimento de onda da luz e a taxa de amostragem do sinal. Porém, este equalizador não elimina completamente a dispersão cromática, tendo, assim, a necessidade de utilizar um equalizador dinâmico como terceiro bloco. Este último bloco, além de corrigir a dispersão cromática residual, também é utilizado para corrigir efeitos inseridos pela fibra em que as polarizações começam a se misturar, sendo esse efeito denominado de dispersão de modo de polarização (PMD - *Polarization Mode Dispersion*) (KIKUCHI, 2016).

O próximo bloco trata sobre o erro de relógio gerado pelo atraso do dado na fibra em relação ao período de amostragem, o que eventualmente ocasiona uma diferença na quantidade de ciclos (variações do relógio local), que deve ser esperado para que a amostra esteja correta, ocasionando assim uma variação na quantidade de amostras obtidas por unidade de tempo. Este bloco é responsável por estimar e corrigir o erro referente à esse efeito anteriormente citado.

O sinal, ao se propagar na fibra, também sofre desvio de fase e frequência, e, para que estes efeitos sejam corrigidos, é inserido no DSP os próximo dois blocos da Figura 4. O primeiro é responsável por estimar o desvio de frequência entre o sinal recebido e o oscilador local, enquanto o segundo tem como foco a estimação e correção do desvio de fase do sinal. Por fim, o último bloco desta arquitetura é responsável por converter os símbolos da modulação em uma sequência de bits.

Como exemplos dos produtos de cada bloco tem-se a Figura 5, para um processo de recepção de um sinal modulado em QPSK na parte superior e 16QAM na parte inferior.

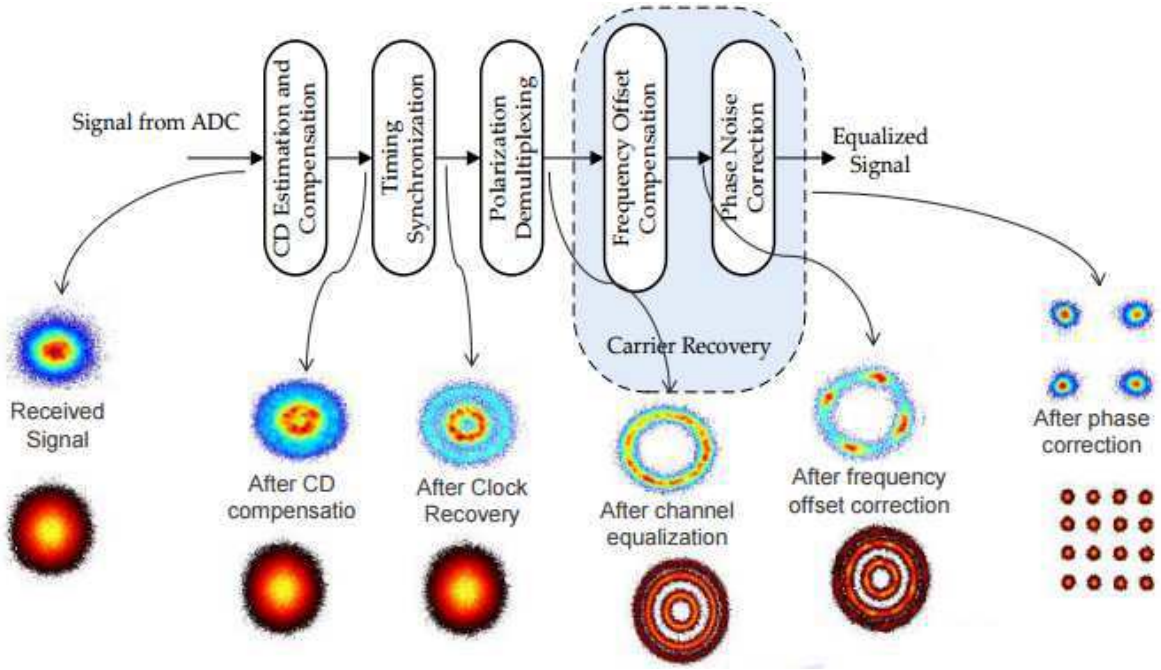

Figura 5 – Representação do processamento do sinal no DSP-RX

2.3 Verificação de um IP

O time de verificação está presente em quase todos os processos de um fluxo de projeto de microeletrônica. Desde da implementação do IP em linguagem de hardware até a síntese lógica, os verificadores devem realizar suas tarefas sobre o produto que foi gerado em cada etapa.

A verificação de um IP envolve o uso de técnicas e topologia em seus ambientes, um exemplo de topologia pode ser visto na Figura 6. As técnicas abrangem as metodologia utilizadas, como UVM (*Universal Verification Metodoly*), e se a verificação implementada será formal ou funcional. Na verificação formal, definem-se axiomas dentro do contexto utilizado, sendo muito utilizada para a verificação de protocolos de comunicação, na qual têm-se sequências que devem ser obedecidas obrigatoriamente pelos sinais.

Na técnica funcional, procura-se verificar a funcionalidade em si do IP, visando, assim, a implementação de teste que exercitem ao máximo os blocos, de forma que se cubra o máximo de cenários possíveis para a utilização do IP.

Todavia, mesmo que a verificação funcional seja planejada e pensada para cobrir as funcionalidades do bloco e possíveis fontes de erros em situações de contorno, algumas peculiaridades dos sinais não podem ser cobertos devido à limitação da ferramenta, como por exemplo, a oscilação da corrente dos sinais que estão ligados aos pinos externos, fazendo com que a velocidade em que o sinal se propaga para dentro ou para fora do IP varie.

No tocante ao ambiente em que é realizado a verificação, pode adotar inúmeras

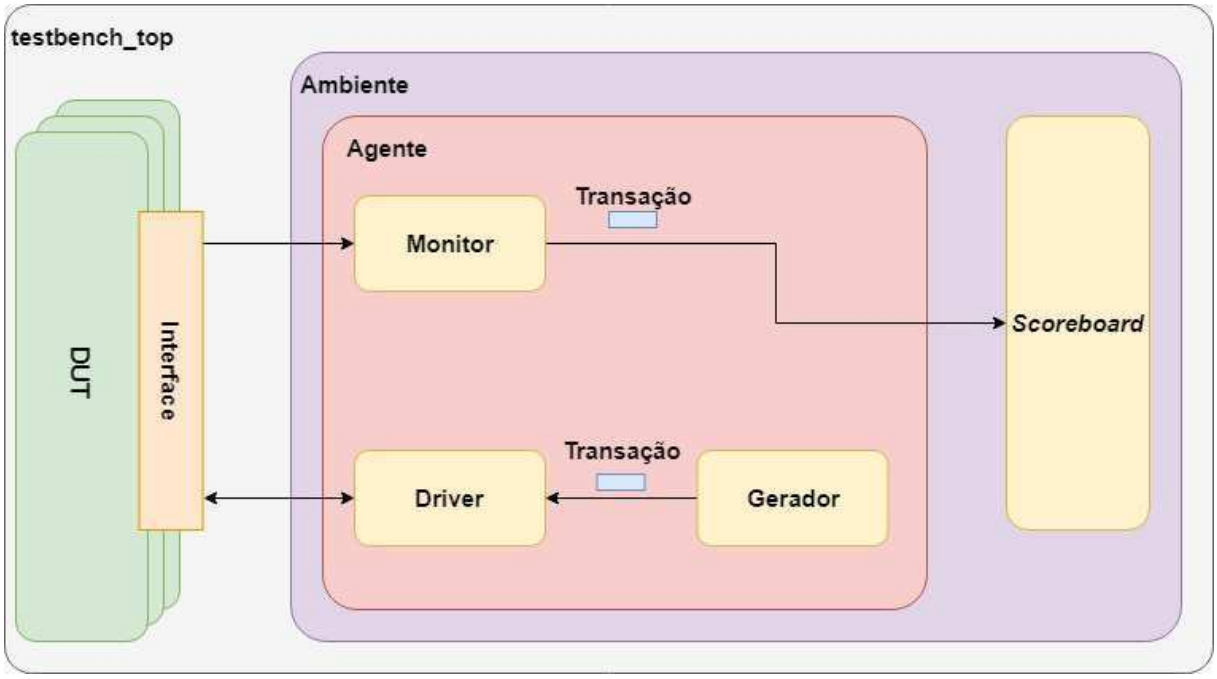

Figura 6 – Exemplo de topologia de um ambiente de verificação

topologias, mas todas seguem a mesma ideia de manter sempre o DUT (*Device Under Test*) sobre constante monitoramento. Para isto, a topologia necessita de um driver que irá injetar os estímulos no DUT, de um monitor, responsável pelo monitoramento dos sinais de saída, de um *checker*, que realiza a comparação com modelo que está sendo seguido, de um *scoreboard*, no qual são contabilizados os erros encontrados pelo *checker*, de um gerador, que gera o estímulos para o driver e de uma interface, que serve como conexão entre o DUT e o ambiente de verificação. Alguns desse componentes podem ser agrupados em um único componente, não deixando de existir, mas passando sua funcionalidade para dentro de outro bloco.

3 Atividades Desenvolvidas

Apresenta-se, na Figura 7, o fluxo decorrente de um projeto de microeletrônica.

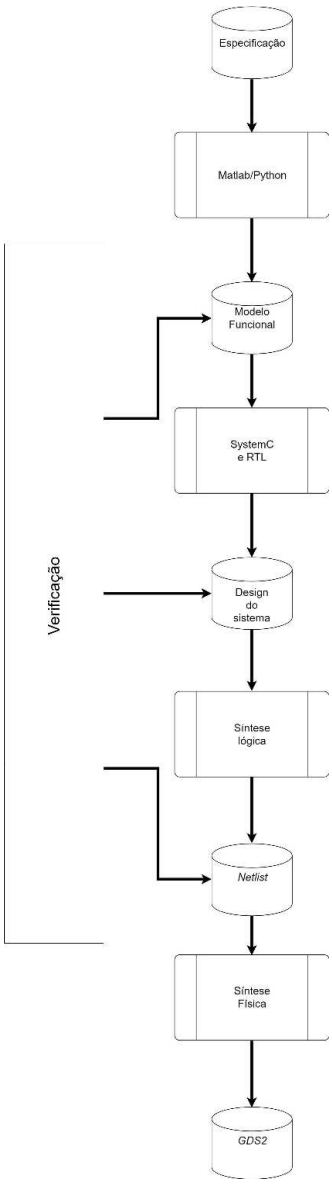

Figura 7 – Fluxo do projeto de microeletrônica

As especificações são advindas do cliente, com isto é possível desenvolver um modelo em linguagem de alto nível, possivelmente Matlab ou Python. Após esta implementação, segue-se com o desenvolvimento de um modelo em SystemC, no qual o modelo consegue ter visão de latência entre os blocos e dentro dele, podendo ainda utilizar de aritmética de ponto fixo. Seguindo o fluxo, a próxima etapa é a descrição do hardware através de uma linguagem, como SystemVerilog, fazendo assim o que se entende por RTL (*Register Transfer Level*). RTL e modelo em SystemC são construídos de maneira a serem

idênticos, bit a bit. Com isso, tem-se a primeira etapa de verificação, no qual é comparado RTL versus SystemC, para verificar a igualdade entre eles para todos os casos em que o sistema deverá atuar; também compara-se RTL versus modelo em alto nível, no caso chama-se de *golden model*, comparação feita com objetivo de averiguar a semelhança entre eles, de modo a assegurar a implementação do RTL de acordo com o modelo em alto nível.

Nesta última etapa de verificação é utilizado um grau de semelhança, ou seja, é aceita uma tolerância na diferença entre o *golden model* e RTL, visto que a implementação em alto nível não possui noção de latência e aritmética de ponto fixo. Assim, com o RTL validado e verificado, gera-se uma lista com todas as conexões realizadas a nível de circuito digital, recebendo o nome de *netlist*. Após a validação da igualdade pelo *design*, é feita uma verificação a nível de GTL (*Gate Transfer Level*) visando garantir que durante a geração da *netlist* não ocorreu nenhum erro que prejudicou o funcionamento. Com a validação e verificação da *netlist* inicia-se o trabalho da última etapa no processo de desenvolvimento, o *back-end*. Este irá projetar o *layout* dos blocos em um circuito integrado. Após o término desta última etapa, obtém-se a base de dados gráficos (GDS - *Graphic Database System*, que será utilizada na fabricação do *chip*.

O aluno participou ativamente do desenvolvimento de um gerador para que os ambientes de verificação fossem gerados, de maneira automatizada partindo de uma planilha já utilizadas para outros fins. Como parte da geração foi solicitado ao estudante também o desenvolvimento de *wrappers* que possibilitassem a verificação do RTL com o *goldem model* ciclo a ciclo. Para que isso fosse possível, era necessário fazer com que os dados advindos de um modelo implementado em uma linguagem em alto nível, no caso Python, conseguisse se comunicar a cada ciclo de funcionamento com o *testbench*, para assim, ser possível a comparação ciclo a ciclo. Então, para isso foi atribuído ao estudante a tarefa de implementar *wrappers* que conseguissem comunicar de *SystemVerilog* para C, e deste para Python, fazendo também o sentido inverso. Nos últimos dois terços do estágio foi atribuído ao estagiário a verificação de três blocos, um com objetivo de codificar dados de entrada para código Gray, um para decodificar o código Gray, e outro bloco responsável pela recuperação de tempo do sinal no DSP (*Digital Signal Processors)*), assim como o auxílio sempre quando existia a necessidade de se gerar novos *wrappers*.

3.1 Implementação dos *wrappers* e gerador de *testbench*

3.1.1 Implementação dos *wrappers*

A empresa possuía uma metodologia de verificação que se baseava em utilizar um modelo em MATLAB para gerar um pacote de dados tanto de entrada quanto de referência de saída do bloco. Ambos pacotes eram utilizados na verificação do bloco, um como estímulo de entrada, e outro para comparação com os sinais de saída dos blocos. Um

dos grandes problemas com essa metodologia é a incapacidade do verificador de conseguir comparar o modelo de referência e o DUT que está sendo verificado ciclo a ciclo de dados. Um outro problema encontrado no referido modelo é o valor das licenças de MATLAB, impossibilitando que haja licenças suficientes para todos os verificadores e modeladores, e consequentemente de se utilizar de artifícios que possibilitem a comunicação entre o MATLAB e o *testbench*.

Assim, decidiu-se fazer uma versão de modelos em Python, onde não teria problemas com licenças, visto que a linguagem é gratuita e sua documentação pode ser facilmente encontrada na internet. Dessa forma, para possibilitar o uso dessas novas versões, fez-se necessário encontrar meios que possibilitassem a comunicação com o *SystemVerilog*.

Como primeira tarefa atribuída ao estudante, foi solicitado o desenvolvimento de um *wrapper* para os modelos em Python. Para que isto fosse possível foi utilizado uma API de Python para C e a DPI-C de *SystemVerilog*, que possibilita a comunicação da linguagem de descrição de hardware com C.

Primeiramente, um arquivo em C foi implementado visando a comunicação entre a linguagem C e Python. Para isso, utilizou-se uma biblioteca de C que contém funções que possibilitam a criação de objetos Python, assim como a declaração de variáveis com os tipos de dados existentes em Python e funções que possibilitam o uso de arquivos escritos em Python.

No desenvolvimento de referido arquivo, o aluno procurou pelas referências disponibilizadas pelos próprios desenvolvedores da linguagem, as quais pode ser encontrada com facilidade no Google. Após uma extensa consulta nos documentos disponibilizados, foi possível a implementação de funções capazes de inicializar um objeto de uma classe que foi implementada em Python e chamar seu construtor, e funções que chamam métodos da classe que foi inicializada. Nessas últimas é possível também a passagem de argumentos para função assim como a leitura do seus retornos, alcançando, assim, o objetivo principal de realizar a comunicação entre o Python e a linguagem C.

A próxima etapa foi a implementação da comunicação das funções escritas em C com a linguagem *SystemVerilog*. Para isso, foi utilizada uma interface de programação direta *DPI - Direct Programming Interface* - já existente na própria linguagem que possibilita o uso de chamada de funções definidas em outras linguagens dentro de um código escrito em *SystemVerilog*.

A DPI é uma interface que possibilita o uso de linguagens ditas como estrangeiras que podem ser C, C++, SystemC, entre outras. As DPI consistem em duas camadas: a camada de *SystemVerilog* e a camada da linguagem "estrangeira". Ambas camadas são isoladas umas das outras. Para o lado do *SystemVerilog* é irrelevante e transparente qual linguagem está sendo utilizada como "estrangeira". Nem ao compilador de *SystemVerilog*

nem ao compilador da outra linguagem é requerido analisar o código fonte da linguagem estrangeira. Diferentes linguagens de programação podem ser usadas e suportadas com a mesma camada de *SystemVerilog*.

O requerimento metodológico é que a interface deve permitir que sistemas heterogêneos sejam construídos (um *design* ou um *testbench*) nas quais alguns componentes podem ser escritos em uma linguagem (ou mais linguagens) diferentes de *SystemVerilog*, que é chamada de "língua estrangeira". Por outro lado, há a necessidade prática para, de uma maneira fácil e eficiente, conectar um código existente, normalmente escrito em C ou C++, sem o conhecimento ou sobrecarga da linguagem de interface de programação (PLI - *Programming Language Interface*) ou interface procedural de Verilog (VPI - *Verilog Procedural Interface*). A DPI segue o princípio de "caixa preta": a especificação e a implementação dos componentes são separadas, e a implementação real é transparente para o sistema. A separação entre código em *SystemVerilog* e a linguagem estrangeira é baseada no uso de funções como a unidade padrão de encapsulamento de *SystemVerilog*. Em geral, qualquer função pode ser tratada como uma caixa preta e implementada em *SystemVerilog* ou em uma linguagem estrangeira de maneira transparente, sem alterar suas chamadas.

Assim, o estudante implementou um código em *SystemVerilog* que importa as funções em C que, por sua vez, possuíam funções que invocam métodos em Python, realizando assim, a comunicação entre o *SystemVerilog* e Python com sucesso.

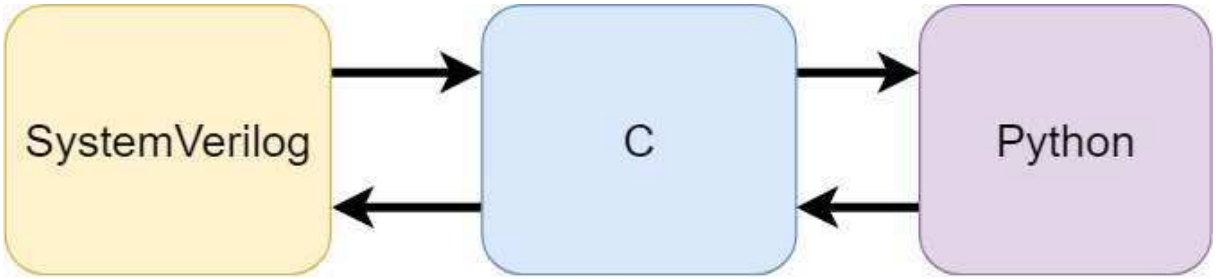

Figura 8 – Fluxo do transição de dados

Seguindo as mudanças do modelo de referência para Python, também houve uma mudança de como os estímulos de entradas são gerados. A empresa desenvolveu um gerador de dados de entrada para os blocos que possui a capacidade de por todos os efeitos inseridos pela fibra nos dados transmitidos que os blocos devem corrigir, ou funcionar com eles. Inicialmente este gerador foi implementado em MATLAB, e depois traduzido para Python. Com isto, teve-se que ser implementado *wrappers* para que fosse possível a utilização deste pela equipe de verificação. Assim, também foi atribuído ao estudante a tarefa de implementar os referidos *wrappers* para a geração dos estímulos ciclo a ciclo, sendo possível obter um ganho na velocidade de simulação e maior economia no espaço utilizado durante a simulação.

3.1.2 Implementação de gerador de *testbench*

Como atividade subsequente foi requisitada a geração dos *wrappers*, para cada bloco que estava sendo desenvolvido, de maneira automática. Com isso, o aluno começou a implementar em Java um gerador que fosse capaz de, a partir da leitura de um planilha gerar *wrapper*. Inicialmente, a necessidade foi o desenvolvimento dos *wrappers* para SystemC, visto que na empresa também são desenvolvidos DUTs em SystemC sendo assim necessário a verificação deste. Para isso, é necessário sua comunicação com o *testbench* que encontra-se em *SysmteVerilog*. Assim, foi implementado em um gerador a capacidade de gerar os arquivos quer servem para a tradução dos diferentes tipos utilizados no *testbench* para o SystemC.

Após essa implementação, foi adicionado ao gerador a parte de comunicação com o Python, constituindo assim um conjunto de 4 arquivos que transacionam dados entre o *SystemVerilog* e o Python.

Esta tarefa se estendeu ao longo de todo o estágio, pois o gerador foi ganhando mais funcionalidades e personalizações que davam suporte aos interesses da empresa. Como resultado dessa atividade, os verificadores conseguiram integrar e simular os modelos em Python e, a cada nova versão de interface dos blocos, era possível gerar novos *wrappers* sem muita demora, tornando o fluxo mais eficiente no quesito tempo.

3.2 Implementação de função para cálculo de correlação

Ao se desenvolver um DSP é inerente o uso de algoritmos que possuam convergência, ou seja, que vão demorar algumas interações para que alcancem os valores corretos de resultado. Assim, o que ocorre quando se compara um modelo de referência em alto nível com algo em implementado em baixo nível, como *SystemVerilog*, é que eles divergem durante o período de convergência, dificultando o trabalho dos verificadores, visto que neste intervalo não se pode comparar o modelo com DUT ciclo a ciclo.

Esta fato ocorre devido a como os cálculos são feitos no modelo e no DUT. Para o modelo, todas as contas possuem pontos flutuantes e resoluções altas, fazendo com que seus cálculos sejam mais precisos e com menos erros por arredondamento. Em contraponto, o DUT, por representar um circuito digital, possui noção de bit, e, com isso, opera com aritmética de ponto fixo e resolução finita, ocasionando erros por arredondamento. Estas diferenças são as principais causas pela divergência que pode existir entre modelo e DUT, quando ambos foram implementados com a mesma arquitetura.

Visando implementar uma solução para o referido problema encontrado e determinar quando e onde dever ser comparado o modelo com o DUT, foi sugerida a utilização de uma função que calculasse a correlação entre os dois vetores de saída e, com o resultado

desse cálculo, descobrir em que ponto se deu a maior correlação entre os vetores sabendo, assim, de onde deve começar a comparação entre os vetores.

Este artifício já era utilizado antes do início do modelo em Python, porém, era gravado um pacote de dados em um arquivo que seria utilizado, após a simulação, para o calculo da correlação. Utilizando-se do fato de que toda a verificação passou a ser feita ciclo a ciclo durante a própria simulação, foi solicitado ao aluno a implementação de uma classe em Python que utiliza-se de classes já existentes em bibliotecas da linguagem que fossem capaz de calcular a correlação entre dois vetores, mas que o fizessem durante a simulação, utilizando as amostras que fossem sendo produzidas pelo modelo e pelo DUT.

A função foi, então, implementada, de maneira que a calcular a correção com as amostras que já foram produzidas durante a simulação, ou com uma quantidade fixa de amostras, e retornar o valor da maior correlação encontrada e em que posição desse vetor a maior correlação ocorreu. Com essas informações é possível saber quando o DUT e o modelo começam a convergir e com isso começar a comparação entre ambos, alcançando, assim, o objetivo da tarefa.

3.3 Verificação de blocos

Ao estudante também foi atribuído a verificação de alguns blocos presentes nos atuais projetos da empresa. Para que fosse possível a verificação, o aluno implementou o ambiente de verificação, juntamente com as peculiaridades que foram solicitadas.

3.3.1 Blocos de codificação e decodificação

Um dos blocos verificados pelo estudante realiza uma codificação para código Gray. O código Gray, também chamado de código binário refletido, foi desenvolvido por Frank Gray nos *Bell Labs* em 1947. A motivação para o desenvolvimento desta codificação foi tentar diminuir a quantidade de bits incorretos entre uma transação e outra. Isto ocorre com o sistema de código binário tradicional, no qual é possível notar a seguir a contagem em base binária: 000, 001, 010, 011, 100, 101, 110, 111. Ao analisar como os valores variam de representação para representação, tem-se variação de mais de um número na transição de valores, como é caso do 1 para 2, na qual ocorre a mudança de dois números: 001 -> 010, ou entre 3 e 4: 011 -> 100. Porém, em dispositivos reais, como os *switches*, é que, por eles não serem ideais, no momento de sua transação de estados pode ocorrer de terem valores intermediários que possam assemelhar-se a outras posições fazendo com que o observador não possa afirmar em qual posição estão os *switches*, como, por exemplo, observa-se o valor 001 na transição entre 3 e 4, porém este valor não é o correto e sim trata-se de um valor de transição da posição dos *switches*.

Para solucionar o problema da transação, Gray propôs um código no qual só haveria variação de um único número de uma posição para outra, obtendo assim, a Tabela 1. Ao analisar a tabela é possível ver que os *switches*, por exemplo, só terão que mudar um único elemento para ter a transação de um valor para o outro, fazendo com que valores de transação não sejam mais relevantes.

Número Decimal	**Código Binario**	**Código Gray**
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Tabela 1 – Tabela do código Gray

Este código é utilizado hoje para minimizar erros em conversão analógica-digital, correção de erro em comunicação digital e comunicação entre domínios de tempo diferentes, estando presente, assim, em sistemas de comunicação óptica coerente.

Dessa forma, o bloco atribuído ao estudante possuía a função de mapear uma entrada de bits para uma configuração em código Gray, com zero de tolerância entre o que era retornado pelo DUT e pelo modelo de referência. Com essa premissa, foi feita a verificação do bloco que, com exceção de pequenos problemas com alguns sinais de controle, não apresentou nenhum erro. Também foi levantado valores de cobertura para o bloco, tanto funcional quanto do bloco, o qual seguindo as métricas da empresa, conseguiu atingir o nível desejado.

Também foi dado como tarefa paralela a verificação de um decodificador de código Gray que possuía a função de de-mapear o código para os símbolos originais. Assim como o codificador, foi implementado um ambiente de verificação com as características necessárias, adquirindo os valores de cobertura funcional e do bloco. O bloco não apresentou nenhum problema e seus valores de cobertura atingiram as métricas desejadas pela empresa.

3.3.2 Bloco de interpolação

Ao final do estágio foi atribuído a verificação de mais um bloco ao estudante. Este, responsável pela parte de recuperação de tempo, realiza uma interpolação para que a recuperação seja possível.

A arquitetura de uma recuperação de relógio pode ser vista na Figura 9. É importante salientar que a menor diferença entre o transmissor e receptor ocasionará uma perda de dados em algum momento. Quando se transmite dados em fibra óptica, o sinal transmitido sofre uma degradação em sua fase e alteração na sua frequência. Assim, para corrigir estes efeitos inseridos no sinal por causa do meio de transmissão, é posto um bloco que tem como funcionalidade a recuperação de relógio.

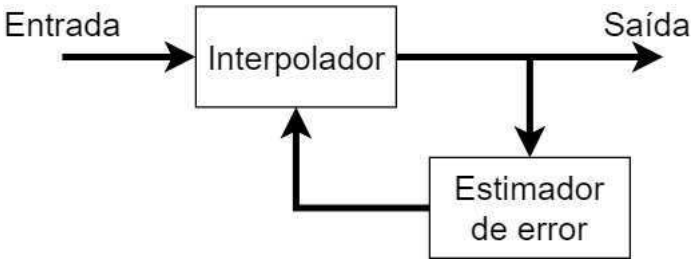

Figura 9 – Arquitetura digital do conjunto para correção de erro de relógio. Fonte: (CáRDENAS, 2010)

Para que isso seja possível, o bloco de interpolação trabalha junto com outro, o qual tem como função a estimação do erro de relógio que se encontra o sinal. Comumente chamado de TED (*Timing Error Detector*), ele se assemelha a operação de um detector de fase em analogia ao PLL (*Phase Look Lock*), o qual calcula a informação de erro baseado na diferença de fase. Existem vários algoritmos que implementam a estimação de erro de relógio e que dependem do fator de super-amostra do formato de modulação. Um desses algoritmos se utiliza de técnicas de controle para estimar o erro partindo de amostra das entradas, que, no caso, serão as saídas do bloco de interpolação. (CáRDENAS, 2010)

A tarefa de um interpolador aqui é de computar a amostra ótima partindo de um conjunto de amostras recebidas. Idealmente ele se comportaria como um filtro FIR (*Finite Impulse Response*) com número de coeficientes infinitos em que seus valores dependeriam do atraso calculado pelo TED. Porém, para uma implementação prática, o interpolador pode ser aproximado para um filtro de ordem finita com a equação 3.1. (CáRDENAS, 2010)

$$H(e^{j\omega T_s}, \mu) = \sum_{n=-I_1}^{I_2} h_n(\mu) e^{-j\omega T_s n} \tag{3.1}$$

Onde, I_1 e I_2 definem o menor e maior amostra de um impulso discreto perto do ponto central. A saído do filtro é dada por combinação linear de I_1+I_2+1 amostras do sinal envolta do ponto m_k. o que leva a formulação da equação fundamental de interpolação 3.2.

$$y(m_k T_s + \mu T_s) = \sum_{n=-I_1}^{I_2} x[(m_k - n)T_s] h_n(\mu) \tag{3.2}$$

Como mencionado anteriormente, os coeficientes dos filtros são dependentes do valor de atraso calculado pelo TED, sendo assim, não fixos e variantes no tempo fazendo com que os valores de atraso tenham que ser limitados.

O controle do bloco de interpolação calcula o ponto base e o atraso baseado no erro de tempo estimado. Para cada amostra o valor de m_k e do atraso tem que ser computado para que seja possível obter a amostra interpolada y. Assim, se o relógio está mais rápido que a taxa de amostragem recebida, em algum ponto se obterá uma amostra extra advinda do conversor analógico-digital. Neste caso, o bloco do interpolador deve retirar uma amostra ou não receber uma amostra, que seria referente à "extra". Por outro lado, se o relógio estiver mais devagar que a taxa de amostragem recebida, em algum momento uma amostra será perdida, fazendo com que o interpolador necessite de uma amostra a mais.

Para a verificação deste bloco, além do ambiente de verificação foi necessário a implementação do modelo de referência em Python. Assim foi requerido do estudante um conhecimento mais completo do que o bloco deveria realizar com o sinal de entrada. Ao contrário dos outros dois blocos verificado pelo estagiário, este, por se tratar de algoritmos, não possuía uma tolerância zero no valor da diferença entre o modelo e DUT, gerando com isso um histograma dos erros para cada cenários de operação do bloco. Estes histogramas servem de objetos para a fabricação dos relatórios produzidos pelo supervisor.

4 Considerações Finais

As atividades que foram designadas para o período de estágio foram realizadas com êxito e, durante esse tempo, foi possível adquirir bastante conhecimento na área de processamento de sinais e Microeletrônica, mais voltado para a verificação de IP.

O ambiente da Idea! foi muito propício para a realização das atividades, pois todos estão focados em elevar o nível da empresa e, assim, atingir o sucesso ao mesmo tempo em que o bem-estar do colaborador é colocado em primeiro plano, o que contribui bastante para que os colaboradores fiquem motivados. Ainda, foi possível estabelecer várias conexões profissionais, o que contribuiu para adquirir mais experiência em determinados assuntos.

Durante o período de estágio, foram utilizados diversos conceitos das disciplinas do curso de Engenharia Elétrica da Universidade Federal de Campina Grande, porém muitos outros tiveram que ser formados com o a experiência do estágio. Após apresentar este ponto, é importante ressaltar a necessidade de um programa de estágio que forneça aos alunos da instituição a possibilidade de adquirir os referidos conhecimentos, os quais não são obtidos apenas com o comparecimentos às disciplinas teóricas e laboratórias ministradas no curso de graduação.

Visando a necessidade dessa experiência para o engrandecimento curricular e formação do aluno como um engenheiro, é necessário que o curso procure tomar conhecimento do que a indústria de ponta necessita, ou seja, procurar formar um engenheiro capaz de trabalhar com a tecnologia que está sendo usada no momento e com o conhecimento com o qual as empresas possam se beneficiar. O papel da universidade como instituição de ensino e formação de profissionais é garantir que os alunos terão acesso a essa experiência, visto que ela engradece a formação, de maneira suave, que possa evitar grandes dificuldades eventualmente encontradas pelos alunos para conseguir e assumir uma vaga em um estágio. Assim, o dever da universidade perante o aluno é auxilia-lo (ajuda-lo) para conseguir a ser inserido em um programa de estágio das empresas ofertantes.

Dessa forma, considero que toda a experiência de estágio foi muito proveitosa e produtiva, possibilitando adquirir conhecimentos não só profissionalizantes, mas também humanos, porém essenciais em qualquer ambiente empresarial ou estudantil.

Referências

CáRDENAS, G. A. D. *All Digital Timing Recovery and FPGA*. 2010. Citado 2 vezes nas páginas 8 e 15.

KIKUCHI, K. *Fundamentals of Coherent Optical Fiber Communications. Journal of Lightwave Technology*, v. 34, p. 157–179, 2016. Citado na página 5.

SEIMETZ, M. *High-order modulation for optical fiber transmission*. [S.l.]: Springer, 2009. v. 143. Citado na página 4.

SOCIETY, T. O. *Lighting the path to a brighter future*. 2018. Citado na página 2.

ZADEH, R. *Evolution of innovation : fiber optics and the communications industry Download*. Massachusetts Institute of Technology, 2004. Citado na página 2.

ZHOU, X.; XIE, C. *Enabling Technologies for High Spectral-efficiency Coherent Optical Communication Networks*. John Wiley and Sons, 2016. Citado na página 4.