UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE SEMICONDUTORES, INSTRUMENTOS E FOTÔNICA

DISSERTAÇÃO DE MESTRADO

Simulação de Sistemas de Comunicação Óptica Baseada em Simulação a Eventos Discretos

Nelson Kiyoshi Sasaki
Orientador: Prof. Dr. Edson Moschim

Campinas, SP – Brasil
Julho - 2007

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE SEMICONDUTORES, INSTRUMENTOS E FOTÔNICA

DISSERTAÇÃO DE MESTRADO

Simulação de Sistemas de Comunicação Óptica Baseada em Simulação a Eventos Discretos

Dissertação apresentada à Faculdade de Engenharia Elétrica
e de Computação da Universidade Estadual
de Campinas, como parte dos requisitos exigidos
para a obtenção do título de Mestre em Engenharia Elétrica.

Nelson Kiyoshi Sasaki
Orientador: Prof. Dr. Edson Moschim

Campinas, SP – Brasil
Julho - 2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Sasaki, Nelson Kiyoshi
Sa78s Simulação de sistemas de comunicação óptica baseada em simulação a eventos discretos / Nelson Kiyoshi Sasaki. -- Campinas, SP: [s.n.], 2007.

Orientador: Edson Moschim
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Simulação. 2. Comunicações óticas. I. Moschim, Edson. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Optical communication systems simulation based on discrete-events simulation.
Palavras-chave em Inglês: Simulation, Optical communications
Área de concentração: Eletrônica, Microeletrônica e Optoeletrônica.
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Sérgio Barcelos, Luiz Henrique Bonani do Nascimento, Furio Damiani e Yuzo Iano.
Data da defesa: 18/07/2007
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Nelson Kiyoshi Sasaki

Data da Defesa: 18 de julho de 2007

Título da Tese: "Simulação de Sistemas de Comunicação Óptica Baseada em Simulação a Eventos Discretos"

Prof. Dr. Edson Moschim (Presidente): _____

Dr. Sérgio Barcelos: _____

Prof. Dr. Luiz Henrique Bonani do Nascimento: _____

Prof. Dr. Furio Damiani: _____

Prof. Dr. Yuzo Iano: _____

Resumo

O objetivo desse trabalho é estudar a aplicação da técnica de simulação a eventos discretos na simulação de Sistemas de Comunicação Óptica, considerando a camada física. É mostrada uma forma de se modelar esse tipo de sistema para que se possa aplicar a simulação a eventos discretos. É demonstrado que a orientação a processos é mais adequada para esse caso. A arquitetura do software que implementa o simulador é descrita, assim como os resultados obtidos com as simulações. Uma análise qualitativa é feita sobre os resultados, chegando-se à conclusão que o simulador, que emprega o método apresentado, executa simulações confiáveis.

Abstract

This work aims to study the discrete event simulation applied to Optical Communication Systems, considering the physical layer. It is shown a way to model this kind of systems so that it can be simulated with discrete event simulation technique. It is shown that process orientation is suitable for this case. The architecture of the software, which implements the simulator, is presented, as well as the simulation results. A qualitative analysis is made on the results, ant it is concluded that the simulator, which implements the presented simulation method, runs trustable simulations.

Aos meus pais, Kazuaki e Nair.

Agradecimentos

Agradeço a Deus por me proporcionar os meios para a realização desse trabalho.

Gostaria de agradecer ao meu orientador, Professor Edson Moschim, pela orientação recebida durante esse trabalho e também pela amizade.

Agradeço à minha família pelo constante e incondicional apoio que sempre recebi.

Também gostaria de agradecer a Willian Muramoto, pela ajuda no desenvolvimento de modelos para as simulações.

Finalmente, agradeço aos meus amigos, que acreditaram no meu esforço e me incentivaram na minha caminhada.

Índice

Lista de Figuras

Lista de Tabelas

Glossário

API – Application Programming Interface

ASE – Amplified Spontaneous Emission

BER – Bit Error Ratio

CATV – Common Antenna Television (Cable TV)

ISDN – Integrated Services Digital Network

DFB – Distributed Feedback lasers

DLL – Dynamic Link Library

FFT – Fast Fourier Transform

LAN – Local Area Network

LED – Light-Emitting Diode

OPLL – Optical Phase-Lock Loop

PMD – Polarization Mode Dispersion

SOA – Semiconductor Optical Amplifier

UML – Unified Modeling Language

1 Introdução

1.1 Motivação

A simulação computacional é um instrumento muito útil na modelagem e estudo de vários tipos de sistemas, desde sistemas naturais até sistemas econômicos e sociais. A simulação faz com que sistemas reais possam ser estudados sem que realmente haja um, permitindo que mudanças em vários aspectos do sistema possam ser experimentadas sem correr o risco de sofrer conseqüências indesejadas. Essa liberdade proporcionada pela simulação permite antever resultados, melhorar o desempenho do sistema real, evitar acidentes, reduzir custos de projetos, estudar sistemas, entre outros.

O mesmo acontece em comunicações ópticas. Vários simuladores têm sido desenvolvidos para simular sistemas de comunicação óptica em todos os seus aspectos, desde o comportamento físico da luz, até protocolos e pacotes em redes ópticas. Isso reflete não somente a variedade de aspectos possíveis de serem simulados, mas também o número de técnicas de simulação que foram desenvolvidas até hoje.

A variedade de aspectos possíveis e de técnicas de simulação torna ainda mais importante e interessante o estudo para determinar qual é a melhor combinação entre elas. Para se ter bons simuladores e boas simulações, é necessário haver um bom casamento entre essas duas partes, ou seja, é necessário conhecer bem tanto o sistema a ser simulado, como as técnicas disponíveis para realizar a simulação. Obviamente, para esta análise, vários elementos devem ser levados em consideração, tal como o ambiente de simulação, recursos disponíveis, e outros.

Essa é uma motivação desse trabalho: propor a utilização de uma técnica de simulação para simular sistemas de comunicação óptica para um dado aspecto. O aspecto desse trabalho é a camada física de sistemas de comunicação óptica, e a técnica de simulação, como diz o título do trabalho, é a simulação a eventos discretos.

Outras motivações vêm de trabalhos anteriores [1], [2], que desenvolveram um simulador de enlaces ópticos, o PCSinfo. Apesar de ser um simulador eficiente, apresenta limitações. Uma delas é a impossibilidade de simular sistemas cuja topologia apresenta

realimentação, já que a técnica de simulação utilizada é adequada apenas para sistemas com topologias do tipo ponto-a-ponto ou ponto-multiponto.

Se fosse possível simular topologias com realimentações, um conjunto muito importante de sistemas ópticos poderia ser simulado: o conjunto dos sistemas de recepção coerente. Os receptores coerentes são capazes de selecionar canais sem a utilização de filtragem óptica. E como a seletividade é muito alta, torna-se viável a implementação de sistemas que utilizam a multiplexação por divisão em comprimento de ondas (WDM) com alta densidade de canais [3].

Outra limitação do PCSinfo está relacionada à complexidade do código fonte do software. O projeto de software não foi feito visando a modularização, e por isso a manutenção e expansão do software são muito custosas. Além disso, a inserção de novos modelos de componentes no simulador requer conhecimento do código fonte de grande parte do software e também a recompilação de todo o código fonte.

1.2 Objetivos

O objetivo primário desse trabalho é o desenvolvimento de um simulador de sistemas de comunicação óptica, tais como: sistemas de transmissão e recepção ópticos, enlaces ópticos, sistemas de recepção coerente e sistemas de amplificação de sinal. No entanto, o foco deve estar na camada física dos sistemas e na observação do comportamento da luz.

Como a aplicação da simulação a eventos discretos para a simulação de sistemas de comunicação óptica não é trivial, a demonstração de como isso pode ser feito é outro objetivo desse trabalho.

Também é objetivo desse trabalho a eliminação das limitações observadas no PCSinfo. Assim, deve-se desenvolver um simulador que seja capaz de simular topologias com realimentação, que seja de fácil manutenção e expansão e que a inserção de novos modelos de componentes seja uma tarefa fácil.

Sumarizando, os objetivos desse trabalho são:

- Desenvolver um simulador de sistemas de comunicação óptica na camada física;
- Mostrar a utilização de eventos discretos na simulação de sistemas de comunicação óptica na camada física;
- Eliminar as limitações dos trabalhos anteriores (PCSinfo);
    - Simular topologias com realimentações;
    - Tornar fácil a inclusão de modelos de componentes;
    - Desenvolver o software de simulação de forma a torná-lo de fácil expansão e de fácil manutenção.

O simulador deve ser um software executável em ambiente PC, obedecendo as boas práticas de engenharia de software. Também deve ser de fácil utilização, sendo conveniente o uso de recursos gráficos.

1.3 Estrutura da dissertação

Nesse primeiro capítulo é dada uma introdução ao trabalho, expondo a motivação os objetivos e a estrutura dessa dissertação.

No capítulo "Simulação a Eventos Discretos" é explicado o funcionamento da técnica de simulação a eventos discretos. São introduzidos conceitos e também mostrados alguns exemplos de simulação.

O capítulo "Outras técnicas de simulação" aborda alguns outros métodos de simulação que foram considerados na fase inicial desse trabalho. Uma comparação entre os métodos é feita, a fim de se justificar a escolha de eventos discretos para o esse trabalho.

O capítulo "Simulação de sistemas de comunicação óptica" aborda o escopo do trabalho e a maneira encontrada para simular sistemas de comunicação óptica usando eventos discretos. São descritas as adaptações feitas e as considerações necessárias.

O capítulo "Implementação" expõe o software do simulador, descrevendo os aspectos computacionais envolvidos na sua implementação.

No capítulo “Resultados” são mostrados os resultados obtidos com o simulador implementado. Análises são feitas para comprovar que a aplicação do método de simulação foi bem sucedida.

Finalmente, no capítulo “Conclusões”, é feita a conclusão do trabalho.

2 Simulação a Eventos Discretos

2.1 Introdução

Simulação a eventos discretos é um método muito estudado e utilizado em simulações de sistemas em geral. Existem vários simuladores comerciais para simulação em diversas áreas, dentre eles [4]: AutoMod, SLX, Extent, SIMAN (Arena) e ProModel. Sua fundamentação teórica [5] é baseada no formalismo de linguagens[1] e autômatos de estados[2]. Nesse contexto, a simulação a eventos discretos é simplesmente a execução de um programa de computador que implementa um autômato de estados temporizado.

Mas, para facilitar o entendimento no contexto desse trabalho, a simulação a eventos discretos será abordada sob um ponto de vista diferente, mais voltado para implementação computacional e sem adotar a modelagem de autômatos de estados e linguagens.

Mas mesmo simplificando a abordagem, é necessário ter-se alguns conceitos e algumas definições relacionados à modelagem de sistemas. Como será visto neste capítulo, com algumas definições básicas de simulação a eventos discretos, já é possível simular sistemas simples. Mas para poder chegar à simulação de sistemas de comunicação óptica, é necessário um pouco mais de sofisticação.

2.2 Definições e conceitos

Uma das formas mais simples de modelar um sistema é a partir de suas entradas e saídas, como mostra a Figura 2.1:

---

[1] Linguagem é uma maneira formal de descrever o comportamento de um sistema a eventos discretos. Nesse formalismo, usam-se analogias entre elementos de eventos discretos e uma linguagem propriamente dita. Por essa analogia, um conjunto de eventos *E* corresponde a um alfabeto, e uma seqüência de eventos de *E* corresponde a uma palavra.

[2] Autômatos são dispositivos que geram ou reconhecem uma linguagem *L* através de regras bem definidas. Gerar uma linguagem significa produzir palavras a partir dessas regras e de um alfabeto *E*; reconhecer uma linguagem significa identificar qual linguagem se encaixa a essas regras e ao alfabeto *E*.

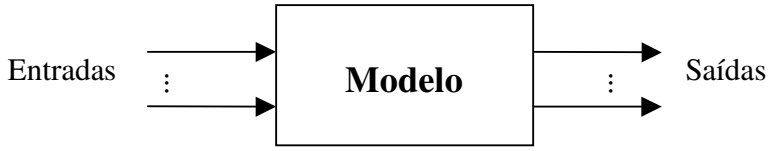

Figura 2.1 – Modelo simplificado de um sistema

A partir dos dados de entrada e de uma relação, o modelo produz os dados de saída. Mas muitas vezes o modelo também necessita de informação a respeito de seu estado corrente para produzir a saída.

O estado do sistema pode ter natureza contínua ou discreta. É de natureza contínua se o estado pode assumir qualquer valor dentro de um intervalo de valores, como por exemplo, a temperatura do sistema. O estado é de natureza discreta se apenas valores discretos são possíveis, como por exemplo, o estado de um interruptor (aberto ou fechado).

Na simulação a eventos discretos, a passagem do tempo é percebida de forma discreta. Portanto, o sistema modelado pode apenas mudar de estado em instantes discretos. A ação ou acontecimento que provoca essa mudança de estado em um determinado instante de tempo é um evento. Por exemplo, se uma máquina está no estado “em funcionamento”, e acontece uma “falha”, a máquina passa para o estado “parado”. Portanto, “falha” é um evento. Um evento pode, ou não, conter informação da entrada do modelo, e também pode provocar a transição de estados para o próprio estado atual.

Cada estado possui um conjunto de eventos aceitáveis, que são eventos possíveis de serem processados e causar a transição para um outro estado. Os eventos não aceitáveis são aqueles que não fazem sentido ocorrer em um determinado estado. Por exemplo, se o sistema modelado é um computador, e seu estado atual é “desligado”, então o evento “falha de leitura no HD” não é aceitável, enquanto que o evento “ligar” é aceitável.

Cada evento deve estar associado a um instante de tempo, que é o instante de tempo que o evento “ocorre” e provoca a transição de estado. Os eventos devem ocorrer obedecendo a uma ordem cronológica, ou seja, os eventos de menor tempo associado primeiro. Se houver dois ou mais eventos com o mesmo tempo associado, deve-se definir um critério de prioridades para decidir qual evento deve ocorrer primeiro.

Todo o sistema deve perceber a mesma passagem de tempo. Na simulação o tempo é uma variável global chamada de tempo de simulação, e deve ser sempre igual ao tempo associado ao evento que está ocorrendo no momento, ou o último evento que ocorreu.

O intervalo de tempo entre cada evento não é necessariamente igual. A simulação deve ser orientada a eventos e não a tempo. Assim, pode-se dizer que os eventos são assíncronos, ou seja, são independentes de um eventual relógio que governa a simulação. Um evento deve simplesmente "anunciar" que está ocorrendo, e todo o sistema deve reagir a essa ocorrência.

A simulação consiste em processar os eventos em ordem cronológica com seus tempos associados, e os resultados são obtidos pela observação da evolução temporal do estado ou das saídas do modelo. Para facilitar a organização, os eventos são colocados em uma lista, ordenados de forma crescente em relação aos tempos associados. O primeiro evento da lista é o de menor tempo associado e será o próximo evento a ocorrer. Assim que o primeiro evento da lista for processado, esse deve ser retirado, dando lugar ao próximo evento.

No início da simulação, a lista deve ser preenchida com os eventos e seus respectivos tempos associados. Os eventos e seus tempos associados podem ser pré-determinados ou gerados através de um gerador de números aleatórios. Este é freqüentemente usado quando se trata de uma simulação com observações estocásticas.

O processamento completo de um evento é uma iteração. Com o que foi descrito até este momento, uma iteração pode ser resumida através dos seguintes passos:

1. Verificar se o primeiro evento da lista é aceitável para o estado atual. Se for aceitável prosseguir com o passo 2. Caso contrário, retirá-lo da lista e repetir o passo 1.
2. Avançar o tempo global de simulação para o tempo associado ao primeiro evento da lista.
3. Atualizar o estado e as saídas do sistema de acordo com o primeiro evento da lista.
4. Retirar o primeiro evento da lista.

As iterações são repetidas até que se atinja uma condição de término de simulação. É muito importante estabelecer um critério bem definido de término, para que a simulação não prossiga indefinidamente, ou pare antes que o desejado. As condições de término mais usuais são:

- limite de tempo de simulação alcançado (o próximo evento da lista tem um tempo associado maior que o limite estabelecido);
- lista de eventos vazia;
- número máximo de iterações alcançado;
- erro no processamento do modelo.

2.3 Exemplos de simulação a Eventos Discretos

A seguir, são dados dois exemplos de simulação a eventos discretos. O primeiro é um exemplo simples para entendimento do método. Já o segundo tem a intenção de preparar o leitor para a simulação de sistemas de comunicação óptica.

2.3.1 Simulação de um sistema de fila simples

Considera-se um sistema simples de fila com um servidor. Nesse sistema, pacotes chegam a fim de serem processados pelo servidor, e assim que são processados, eles deixam o sistema. O servidor processa um pacote por vez e o tempo de processamento não é nulo. Portanto, se o servidor estiver ocupado com um pacote enquanto novos pacotes chegam ao sistema, os mesmos devem aguardar na fila em ordem de chegada até que o servidor esteja disponível.

Na fila há apenas lugar para três pacotes. Assim se a fila estiver cheia e outro pacote chegar, esse deve ser descartado.

Para facilitar a análise, considera-se que o pacote que está sendo processado pelo servidor é o primeiro da fila, ou seja, não há pacotes “dentro” do servidor.

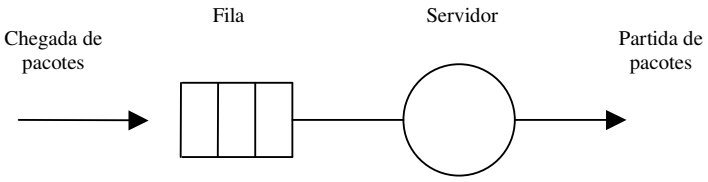

Figura 2.2 – Sistema de fila simples

O número de pacotes na fila é definido como o estado do sistema, e é armazenado na variável S. O estado inicial considerado é nulo, ou seja, inicialmente a fila está vazia (S = 0).

São considerados dois eventos: chegada de novo pacote, e término de processamento do pacote atualmente sendo processado. Esses dois eventos são representados por E1 e E2, respectivamente. Deve estar claro que a ocorrência de E1 ou E2 provoca uma alteração no valor de S, exceto no caso de ocorrer o evento E1 no estado S = 3 (descarte de pacote).

É importante notar que se não há pacotes na fila, não pode haver processamento. Portanto, o evento E2 não é aceitável no estado S = 0.

No início da simulação, um gerador de números aleatórios gera uma seqüência de tempos associados a eventos do tipo E1 e outra para o tipo E2. Considerando a junção dos dois tipos de eventos, é dada a seguinte lista única de eventos:

Tabela 2.1 – Lista de eventos

Evento	Tempo associado (s)
E1	1
E1	3
E2	4
E1	6
E1	7
E1	9
E2	11
E2	15
E1	16
E1	17
E1	20

São desejados dois resultados: o número de pacotes descartados, devido à fila estar cheia, e o tempo médio de espera na fila. No primeiro caso, basta que, no decorrer da simulação, um contador seja incrementado toda vez que um pacote é descartado. Para o segundo resultado, é preciso armazenar a soma dos tempos de espera de todos os pacotes, para que, no final, se possa calcular a média. Esse tempo de espera deve contabilizar o tempo desde a entrada de cada pacote na fila até sua partida.

O critério de término da simulação será o tempo limite de 20 s, ou seja, a simulação deve terminar quando o tempo global de simulação atingir 20 s.

A tabela a seguir mostra a evolução das iterações dessa simulação:

Tabela 2.2 – Evolução das iterações

Tempo (s)	1	3	4	6	7	9	11	15	16	17	20
Evento	E1	E1	E2	E1	E1	E1	E2	E2	E1	E1	E1
S	1	2	1	2	3	3	2	1	2	3	3
Pacotes descartados	0	0	0	0	0	1	1	1	1	1	2
Tempo total de espera	0	0	3	3	3	3	11	20	20	20	20

Na tabela acima, pode-se observar a evolução temporal das variáveis envolvidas. No tempo inicial (0s), o número de pacotes na fila é 0 (S = 0). Quando tempo = 1s, o primeiro pacote chega ao sistema, ou seja, o evento E1 ocorre. Isso faz com que o estado seja alterado, S = 1.

Em tempo = 3s, outro pacote chega (E1), e, portanto, o estado é incrementado (S = 2).

Já em tempo = 4s, o primeiro pacote termina de ser processado (E2), e por isso, o estado deve ser atualizado (S=1). Como em tempo = 4s o primeiro pacote foi totalmente processado, é também contabilizado o tempo total de espera desse pacote: sua chegada foi em tempo = 1s, e sua saída foi em tempo = 4s; portanto o tempo de espera foi de 3s.

Esse processamento continua até se atingir tempo = 20s. Nota-se que no final da simulação, chegou-se a um total de 2 pacotes descartados. O tempo total de espera na fila foi

de 20s, com 3 pacotes sendo completamente processados. Assim, o tempo médio de espera na fila é de 20/3 = 6,666 s.

2.3.2 Simulação de um laser semicondutor DFB

Nesse exemplo, não será feita uma simulação passo a passo como no exemplo anterior, mas será analisada uma forma simples de simular um componente óptico: um laser DFB. A intenção é ressaltar os conceitos de simulação a eventos discretos mostrados até agora, deixando de lado detalhes de modelagem. A simulação de um laser DFB é retomada de forma mais completa nos próximos capítulos, onde haverá elementos mais sofisticados para realizar a simulação.

Considera-se o modelo de laser semicondutor DFB. Seu comportamento é modelado pelas equações de taxa que determinam a variação temporal da densidade de fótons, da densidade de portadores e da fase. As equações de taxa são três equações diferenciais acopladas (equações 5.1, 5.2 e 5.2) e podem ser resolvidas numericamente através do método clássico de Runge-Kutta de quarta ordem.

Supõe-se que a corrente de entrada do laser possa assumir apenas dois valores: 60mA e 40mA. Todos os outros parâmetros, tais como volume da cavidade, fator de confinamento e tempo de vida do elétron, são fixos. Portanto, a simulação deverá determinar a densidade de fótons, a densidade de portadores e a fase, a partir da variação da corrente de entrada do laser.

Apenas o nível da corrente de entrada é considerado como estado, que, portanto, pode assumir apenas dois valores: I = 60mA e I = 40mA. Conseqüentemente, são considerados dois eventos, E1 e E2, que significam mudança no nível da corrente de entrada para 60mA e para 40mA, respectivamente. Os valores de densidade de fótons e de portadores da última iteração também são armazenados, mas não farão parte do estado.

As iterações de Runge-Kutta são executadas por um laço de execução (*looping*) independente de eventos. Cada iteração utiliza-se do valor atual da corrente de entrada. Assim, quando houver uma alteração no nível de corrente, as iterações numéricas devem refletir essa variação na densidade de fótons, de portadores e fase.

Com esses elementos, pode-se simular uma seqüência de pulsos ideais de corrente, com um nível baixo de 40mA e um nível alto de 60mA, como mostra a figura a seguir:

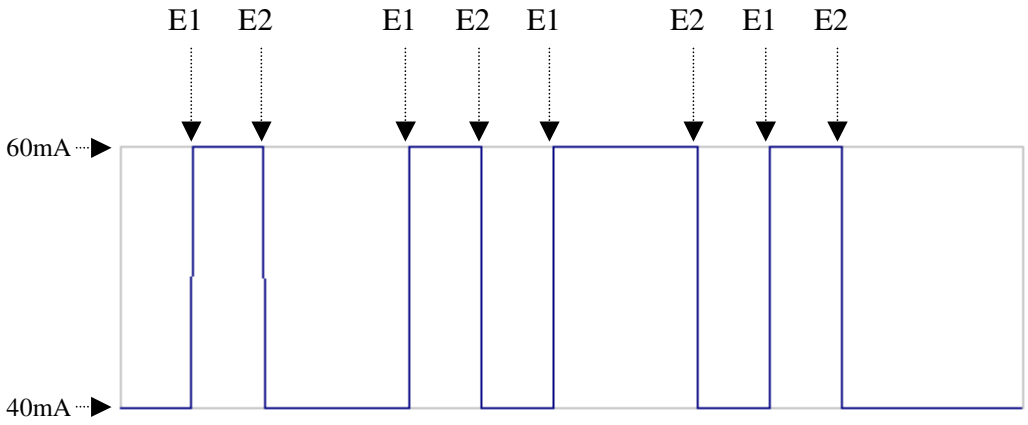

Figura 2.3 – Seqüência de pulsos de corrente (transição de estados do laser)

A figura Figura 2.3 mostra a transição de estados, ou seja, a corrente de entrada, para a seqüência de eventos E1,E2...,E1,E2. Apesar dos eventos estarem uniformemente intercalados, os intervalos de tempo entre eventos não são iguais.

Supondo que no intervalo entre dois eventos sucessivos haja tempo suficiente para executar várias iterações numéricas para resolver as equações de taxa, o resultado qualitativo para a densidade de fótons é mostrado na figuraFigura 2.4.

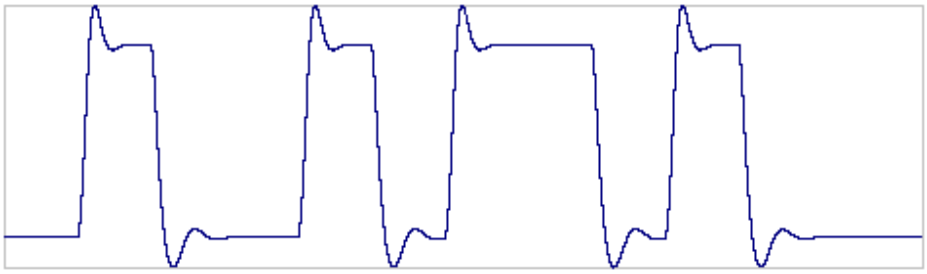

Figura 2.4 – Densidade de fótons

Nota-se que a densidade de fótons acompanha a variação da corrente de entrada, e apresenta as oscilações de relaxação, resultante da solução numérica das equações de taxa.

2.4 Conclusão

A simulação a eventos discretos está intimamente relacionada com a discretização do tempo. Eventos ocorrem em instantes de tempo bem definidos, alterando o estado do sistema e produzindo dados de saída do modelo. O intervalo de tempo entre as ocorrências de eventos não é necessariamente sempre o mesmo, já que a simulação é orientada a eventos.

Com o simples processamento dos eventos em ordem cronológica, como foi visto neste capítulo, já é possível simular comportamentos simples de sistemas simples, tal como um sistema de fila simples. Mas para simular sistemas de comunicação óptica, são ainda necessários mais alguns elementos, conceitos e metodologia, o que serão visto nos próximos capítulos.

3 Outras técnicas de simulação

3.1 Introdução

Antes de se optar pela simulação a eventos discretos, outros métodos de simulação também foram analisados. O objetivo dessa análise é comparar os métodos, e achar qual é a melhor opção para a simulação de sistemas de comunicação óptica no escopo deste trabalho (camada física).

Assim como eventos discretos, os métodos apresentados a seguir não são apenas métodos de simulação. São paradigmas mais gerais, usados também em modelagem computacional, e sistemas de controle e pesquisa operacional. Mas para esse trabalho, esses paradigmas são vistos do ponto de vista de simulação.

3.2 *Synchronous Data Flow* (SDF)

Data flow é um paradigma que usa grafos orientados para descrever um processo [6]. Em um grafo, cada nó representa uma unidade de processamento, e cada arco, que liga um nó a outro, representa um caminho por onde os dados percorrem. A Figura 3.1 mostra um exemplo de grafo de *data flow*.

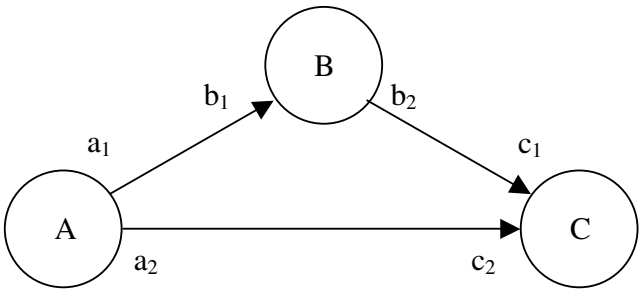

Figura 3.1 – Grafo de *data flow*

Quando um nó é disparado (executa um processamento), uma certa quantidade de dados é consumida dos arcos que entram no nó, e outra quantidade de dados é produzida e enviada a outros nós através dos arcos que saem do nó. Uma entidade ou unidade de dado que é produzida ou consumida pelos nós, e trafega pelos arcos, é chamada de *token*.

Na Figura 3.1, os pares letra-número associados às extremidades dos arcos, representam o número de *tokens* consumidos ou produzidos quando o nó em questão é disparado. Por exempo, no disparo do nó B, b1 *tokens* são consumidos e b2 *tokens* são produzidos.

Um nó pode ser disparado tão logo haja dados (*tokens*) em suas entradas. Um nó que não tenha arcos de entrada pode ser disparado a qualquer momento. Pelo fato da condição de disparo ser a simples presença de dados, diz-se que *data flow* é um paradigma orientado a dados.

Synchronous Data Flow, SDF, é um caso especial de *data flow*, onde o número de *tokens* consumidos e produzidos por cada nó permanece constante durante todo o processo. Portanto, a Figura 3.1 também pode ser um SDF, desde que a_1, a_2, b_1, b_2, c_1 e c_2 permaneçam constantes.

Por causa dessa constância, as taxas de consumo e produção de *tokens* de cada nó podem e devem ser determinadas a priori, antes da execução do processo. A programação (*scheduling*) do fluxo de dados e disparos dos nós é estática, e os dados não podem exercer controle sobre seu fluxo, ou seja, o conteúdo dos *tokens* não altera a quantidade de *tokens* consumidos e produzidos pelo nó.

As taxas de consumo e produção dos *tokens* são calculadas a priori através das equações de balanço. No exemplo da Figura 1, as equações de balanço são:

$$\begin{aligned} n_A\, a_1 &= n_B\, b_1 \\ n_B\, b_2 &= n_C\, c_1 \\ n_A\, a_2 &= n_C\, c_2 \end{aligned} \qquad (3.1)$$

Em (3.1), n_A, n_B e n_C são, respectivamente, os números de vezes que o os nós A, B e C são disparados.

Se, por exemplo,

$$\begin{aligned} a_1 = a_2 = b_1 = b_2 = 1 \qquad (3.2) \\ c_1 = c_2 = 2 \end{aligned}$$

Então, usando (3.1),

$$n_A = n_B$$
$$n_B = 2n_C \qquad (3.3)$$
$$n_A = 2n_C$$

Uma solução não-nula seria:

$$n_A = n_B = 2 \qquad (3.4)$$
$$n_C = 1$$

Assim, cada vez que o nó C é disparado, os nós A e B devem ser disparados duas vezes.

Existem casos em que as equações de balanço não podem ser satisfeitas. Por exemplo, na Figura 3.1, se

$$a_1 = a_2 = c_1 = c_2 = b_1 = 1 \qquad (3.5)$$
$$b_2 = 5$$

Não há solução para n_A, n_B e n_C. Nesse caso, em uma iteração, se os nós A e C forem disparados 5 vezes para cada disparo de B, então serão acumulados 4 *tokens* no arco que liga o nó A e B. Se o processo é contínuo, com infinitas iterações, esse arco necessitará um buffer de tamanho infinito para armazenar os *tokens*.

Portanto, para que um sistema seja computacionalmente modelável no SDF, é necessário que haja consistência entre as taxas de consumo e produção dos *tokens* e a topologia. Essa é a principal limitação do SDF.

Porém, apesar de ter essa limitação, o SDF é de implementação fácil e robusta. O fato da programação de fluxo ser determinada estaticamente *a priori* faz com que não haja *overhead* em tempo de execução, o que garante um bom desempenho do processo.

3.3 *Distributed Discrete Event* (DDE)

A simulação usando a técnica *Distributed Discrete Event* (Eventos Discretos Distribuídos), DDE, consiste em aplicar a simulação a eventos discretos em um ambiente multi-processado, normalmente um conjunto de computadores, ou um computador com vários processadores.

DDE utiliza a abordagem de orientação a processos de eventos discretos, que será detalhada no capítulo 4. Nessa abordagem, modela-se o sistema em vários processos, onde os processos trocam mensagens entre si.

DDE explora o processamento paralelo de processos, e, se bem projetado, pode diminuir drasticamente o tempo de simulação em comparação com a simulação a eventos discretos de processamento seqüencial (capítulo 2). Isso é especialmente interessante em simulações que gastam muito tempo de processamento (horas e até dias). Mas, com o processamento paralelo, surge um problema: a sincronização.

Na técnica de simulação DE seqüencial, o evento processado é sempre o que tem o menor tempo de ocorrência. Se caso um outro evento que tenha um tempo de ocorrência maior for processado antes, e esses dois eventos acessam a mesma variável de estado, ocorre uma situação de erro em que o futuro afeta o passado. Esse erro é chamado de erro de causalidade.

O tratamento de erros de causalidade é o maior problema a ser resolvido em DDE. Para ilustrar melhor, supõe-se que existam dois eventos, E_1 e E_2, com tempo de ocorrência T_1 e T_2 respectivamente, e que $T_1 < T_2$. Se os dois eventos acessam a mesma variável de estado (ou outro recurso comum aos dois), então E_1 deve ser processado antes que E_2, de forma seqüencial.

É necessário garantir que é seguro processar o evento, dados os outros eventos. Determinar qual processamento deve ocorrer antes de outro não é uma tarefa fácil, pois essa decisão também depende do conteúdo dos dados. Se for adotada a política de sempre processar o evento cujo tempo de ocorrência é o menor de todos, garante-se que não ocorrerão erros de causalidade, ou seja, é sempre seguro processar o evento com o menor tempo de ocorrência. Mas fazendo isso, volta-se ao caso da simulação seqüencial, onde um processo é processado após o outro, e não se extrai nenhum ganho do processamento

paralelo. Assim, ao mesmo tempo em que é preciso garantir a coerência de causalidade, é preciso garantir um mínimo de processamento paralelo para que a simulação DDE valha a pena.

Existem duas categorias de mecanismos que tentam resolver esse problema: os conservadores e os otimistas [7]. A diretiva do mecanismo conservador é evitar a todo custo a situação de erro de causalidade. Para isso, é necessário determinar quando os eventos que podem afetar o evento em questão já foram processados, ou seja, quando é seguro processar um evento. Processos que não contenham eventos seguros de serem processados devem ser bloqueados, até que se tenha uma condição de segurança. Devido a esses bloqueios e a possíveis disposições dos processos, podem ocorrer problemas de *deadlock*. *Deadlock* é o bloqueio de todo o sistema devido à espera de um processo bloqueado por eventos seguros de outros processos que também estão bloqueados. Durante o processo (simulação) deve haver um constante monitoramento dessas condições, para que *deadlocks* sejam detectados e tratados.

Já no mecanismo otimista, a idéia não é evitar erros de causalidade, mas corrigi-los quando ocorrerem. Assim que um erro de causalidade é detectado, procedimentos são executados para fazer o estado do sistema voltar até a condição anterior ao erro, e então processar novamente os eventos da maneira correta. O mecanismo otimista explora melhor a vantagem de processamento paralelo do DDE.

3.4 *Process Network* (PN)

Process Network (PN) é um modelo computacional onde processos comunicam entre si através de canais unidirecionais, formando uma rede de processos. Cada processo produz elementos de dado, *tokens*, e os envia pelo canal unidirecional que contém uma fila de *tokens*. Esta fila é do tipo FIFO (*First In First Out*), e os *tokens* permanecem na fila até que o processo de destino os consuma.

Um processo continuamente consome *tokens* do canal de entrada. Mas se caso o canal estiver vazio, o processo deve ficar bloqueado. Um único canal vazio deve ser suficiente para o bloqueio; mesmo se outros canais de entrada não estiverem vazios, o processo deve permanecer bloqueado. Assim, um processo pode estar em apenas um de dois

estados: processando *token*, ou bloqueado. Isso faz com que o paradigma PN seja determinístico, ou seja, a seqüência de *tokens* em um canal é completamente determinada pela modelagem do sistema.

O paradigma *data flow*, descrito no item 3.1, é na verdade um caso especial de *process network* [8]. Em vez de ter regras que determinam as taxas de consumo e produção de *tokens*, presente no *data flow*, o *process network* usa a semântica de bloqueio. A Figura 3.1 pode representar também um sistema PN, onde cada nó representa um processo.

Da mesma forma que em DDE, *deadlocks* também podem ocorrer. Esses podem ser detectados monitorando o estado dos processos; quando todos os processos estiverem bloqueados, então aconteceu um *deadlock*.

PN é muito usado para a modelagem de sistemas concorrentes, como por exemplo, sistemas embarcados.

3.5 Comparações

A técnica escolhida para este trabalho foi eventos discretos. Essa escolha se deve a várias vantagens observadas dessa técnica quando comparadas com as outras e levando-se em consideração o escopo de simulação de sistemas de enlace óptico.

Primeiramente, DDE e PN são, por natureza, técnicas usadas em ambientes multi-processados. Considerando que um dos objetivos desse trabalho é implementar um simulador no ambiente PC, com um único processador, essas técnicas podem ser implementadas através de *threads* [3]. Mas, nesse caso, a vantagem da rapidez do processamento paralelo se perde. Além disso, por envolver processamento paralelo, essas técnicas são de implementação complexa, especialmente DDE, por causa do tratamento dos erros de causalidade. Assim, não faz sentido pagar um preço caro pela implementação complexa dessas técnicas se não for obtido nenhum ganho.

---

[3] *Thread* é uma linha de execução de instruções, e é controla pelo sistema operacional. Duas *threads* são processadas independentemente uma da outra, implementando um processamento paralelo (ou aparentemente paralelo).

Como foi abordado anteriormente, DDE tem implementação complexa por causa do sincronismo. No método PN, nada foi dito a respeito de tempo. Na verdade, é necessário introduzir mecanismos de sincronização para introduzir a noção de tempo. A implementação desses mecanismos também não é simples.

No SDF também não há a noção de tempo, e mecanismos temporais devem ser implementados para esse fim. Apesar de ser muito usado em modelagem de processamento paralelo, o SDF pode também ser implementado de forma seqüencial.

Trabalhos anteriores [1], [2], implementaram um simulador de enlaces ópticos (PCSinfo) usando um caso especial de SDF. A simulação é desenvolvida no domínio da freqüência, e não do tempo. Assim, não é necessária a noção do tempo. Como o processamento é no domínio da freqüência, um componente (processo) deve ser simulado completamente para todo o intervalo (temporal) de simulação. Feito isso, o próximo componente é simulado, e assim sucessivamente, até o último componente. Nesse caso especial de SDF, o número de vezes que cada processo é disparado, R, é igual a 1.

Essa implementação é muito robusta, e funciona muito bem para enlaces ópticos, onde a topologia é do tipo ponto-a-ponto. Mas topologias com realimentações não são possíveis de serem simuladas nessa implementação, por causa do erro de causalidade.

Já com eventos discretos, é possível implementar a simulação de sistemas de comunicações ópticas com ou sem realimentações, com noção de tempo, seqüencialmente, sem a complexidade do processamento paralelo e de forma eficiente.

Mas para isso, são necessários alguns conceitos adicionais em relação ao que foi descrito no Capítulo 2. Esses conceitos serão abordados no próximo capítulo, já integrando com a simulação de sistemas de comunicação óptica.

3.6 Conclusão

Cada um dos métodos apresentados neste capítulo apresentam vantagens e desvantagens em relação a simulação a eventos discretos. A escolha por simulação a eventos discretos se deu pela comparação entre esses métodos, levando-se em consideração o tipo de sistemas a serem simulados, e os recursos disponíveis para realizar a simulação (ambiente

PC monoprocessado). Considerando esses pontos, a simulação a eventos discretos apresenta várias vantagens e poucas desvantagens, e se mostrou o mais adequado.

Além disso, simulação a eventos discretos é uma técnica que já foi muito bem estudada e fundamentada. Sua implementação em um programa de computador também não é custosa.

4 Simulação de sistemas de comunicação óptica

4.1 Introdução

A simulação a eventos discretos apresentada no capítulo 2 não é muito adequada quando a modelagem do sistema envolve vários processos interdependentes. Essa é a modelagem comumente empregada em sistemas de comunicação óptica, onde componentes são processos interligados que trocam informações entre si.

Neste capítulo será apresentada a aplicação da técnica de simulação a eventos discretos na simulação de sistemas de comunicação óptica para o escopo desse trabalho. Primeiramente será explicitado o cenário da simulação, onde será apresentada a modelagem adotada para sistemas de comunicação óptica. Serão explicados os problemas que surgem ao se tentar aplicar diretamente a técnica abordada no capítulo 2. Depois serão expostas progressivamente as adaptações que contornam esses problemas e possibilitam a simulação.

4.2 Sistemas de comunicação óptica

Sistema de comunicação óptica é um sistema de comunicação que usa a luz para transmitir informação [9]. Normalmente, as informações transmitidas são dados digitais (bits). Esses dados, que inicialmente estão no domínio elétrico, são convertidos para o domínio óptico, transmitidos, e então convertidos para o domínio elétrico novamente.

As três principais partes de um sistema de comunicação óptica são: canal óptico, transmissor óptico e receptor óptico. Na maior parte dos casos, o canal óptico usado é a fibra óptica. Existem, no entanto, sistemas que usam o espaço livre como canal, tais como *Free Space Optics* (FSO) e IrDA (Comunicação por ondas infra-vermelhas). Como esses são casos especiais, doravante a fibra óptica será considerada como canal óptico.

O transmissor óptico tem o papel de converter um sinal elétrico em óptico, e injetá-lo no canal óptico. Para isso, é usada uma fonte de luz, que normalmente é uma fonte de luz semicondutora, tal como o diodo emissor de luz (LED) e o laser semicondutor. A conversão do sinal elétrico em sinal óptico se dá através de modulação da luz, que pode ser direta ou

externa. Na modulação direta, a própria corrente de polarização do LED ou laser é modulada, resultando na modulação da intensidade da luz. Já na modulação externa, um dispositivo externo à fonte de luz é usado para modular a luz.

O receptor óptico tem como objetivo converter o sinal de luz proveniente do canal óptico em sinal elétrico e recuperar a informação original. O principal componente do receptor óptico é o fotodetector, que normalmente é um semicondutor: o fotodiodo.

Sob o ponto de vista arquitetural, sistemas de comunicação óptica podem ser de três tipos: enlace óptico, distribuição e redes.

Enlace óptico é uma arquitetura cuja topologia é do tipo ponto-a-ponto. É usado para transmitir dados a taxas elevadas (na ordem de vários Gbit/s) de um lugar a outro, e seu comprimento pode variar de alguns metros (*short haul*) até milhares de kilômetros (*long haul*).

No enlace óptico, especialmente nos de longa distância, o sinal de luz é degenerado à medida que percorre a fibra óptica. Isso é devido à atenuação da intensidade da luz, dispersão cromática, dispersão do modo de polarização, e outros fatores de degradação do sinal presentes na fibra óptica. Para contornar esses problemas são usados repetidores opto-eletrônicos no enlace para regenerar o sinal. Um repetidor nada mais é do que um par receptor-transmissor que converte o sinal óptico em elétrico, o regenera, e o converte novamente em sinal óptico.

Com o advento de amplificadores ópticos, eliminou-se a necessidade de transformar o sinal óptico para elétrico a fim de amplificar o sinal. Isso possibilita um custo menor para enlaces ópticos. Porém, o amplificador óptico injeta ruído no sinal óptico, e também o efeito acumulado de vários amplificadores ópticos em cascata aumenta o impacto da dispersão da fibra, não-linearidades e de outras degradações. Assim, para enlaces de longa distância, ainda são necessários alguns repetidores opto-eletrônicos para quebrar esse efeito em cascata dos amplificadores ópticos.

A Figura 4.1 ilustra uma configuração de um enlace óptico típico.

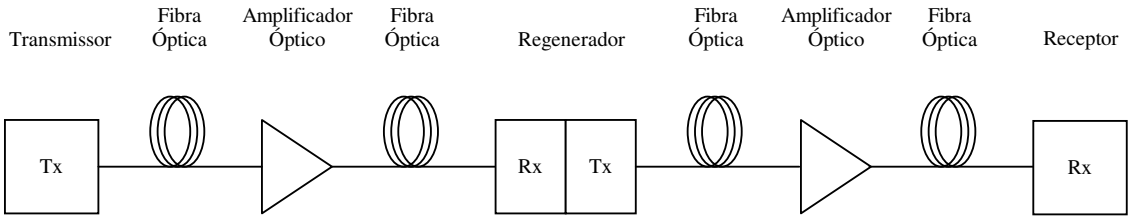

Figura 4.1 – Enlace óptico típico

Sistemas de comunicação óptica de distribuição apresentam uma topologia do tipo ponto-multiponto. São usados em distribuição de dados, por exemplo, CATV e ISDN. Já os sistemas de comunicação do tipo redes são usados em LANs.

Para simplificar e facilitar o entendimento, o enlace óptico será o tipo de sistema de comunicação óptica considerado desse ponto em diante. Quando não for o caso, será explicitamente indicado.

4.3 Cenário da simulação

Foi definido como objetivo desse trabalho a simulação de sistemas de comunicação óptica em sua camada física. Mas o objetivo de uma simulação dessa natureza é observar o comportamento do sinal óptico, envolvendo sua geração, recepção e eventuais tratamentos que possa receber. Sinais elétricos que fazem parte do sistema simulado também devem ser observados.

O comportamento do sinal óptico é influenciado por vários fenômenos ópticos que ocorrem devido à natureza do meio óptico e ao tratamento que a luz recebe. São exemplos desses fenômenos: relaxação do laser semicondutor, atenuação da intensidade da luz na fibra, dispersão cromática da fibra, inserção de ruído ASE pelos amplificadores ópticos e dispersão do modo de polarização (PMD). Em um eventual projeto de sistema de comunicação óptica, se esses fenômenos não forem bem estudados e controlados, a qualidade do sinal digital será comprometida, resultando em altas taxas de erros de bits (BER), redução na taxa máxima de bits transmitidos, necessidade de aumento da potência

do transmissor óptico, e outras conseqüências indesejadas. Assim, esses fenômenos ópticos devem fazer parte do escopo de simulação.

Muitas vezes, a simulação envolve a passagem de dados digitais (bits) pelos diversos componentes do sistema. Todavia, é necessário que haja uma observação contínua com o tempo para perceber os fenômenos descritos anteriormente. Por exemplo, dado um laser modulado diretamente, um fenômeno de interesse a ser observado é a oscilação de relaxação, provocada pela mudança abrupta da intensidade de luz quando há uma transição de um bit "0" para um bit "1" e vice-e-versa.

Nota-se que a natureza do escopo de simulação é contínua. Para reforçar ainda mais essa característica, as modelagens dos componentes ópticos do sistema são geralmente feitas através de equações diferenciais. De fato, se o sistema de comunicação óptica na camada física fosse classificado nos critérios de [5], esse não seria um sistema a eventos discretos, mas um sistema dinâmico, invariante no tempo e contínuo! Nesse contexto, fracassaria uma tentativa de enquadrar diretamente o sistema de comunicação óptica no domínio de eventos discretos, pois um é de natureza contínua e outro é discreto. Mas apesar desse não ser um sistema a eventos discretos, não significa que o método de simulação a eventos discretos não pode simulá-lo. Há uma diferença entre simulação de sistemas a eventos discretos e simulação a eventos discretos de sistemas. A simulação a eventos discretos de um sistema de natureza contínua é o cenário da simulação deste trabalho.

Para que essa simulação seja possível, são necessárias algumas adaptações em relação à simulação a eventos discretos abordada no capítulo 2. Essas adaptações são descritas a seguir.

4.4 Modelagem de um sistema de comunicação óptica

Para que um sistema real possa ser simulado, esse deve sofrer abstrações a fim de se obter um modelo possível de ser simulado computacionalmente. Mais ainda, as abstrações podem ser feitas de tal forma que se tenha um modelo que se encaixe em uma determinada técnica de simulação.

Neste trabalho, as abstrações são feitas sempre visando a obtenção de um modelo de sistema de comunicação óptica que possa ser simulado usando eventos discretos. Essas abstrações são descritas a seguir.

4.4.1 Mensagens

Nos sistemas de comunicação óptica, a interconexão dos componentes é para permitir que sinais elétricos ou ópticos fluam no sistema, sendo alterados a cada componente por onde passam. Esses sinais são modulados, transportando a informação que normalmente são dados digitais, como explicado em 4.1.

Para facilitar a análise, considera-se um único sinal que sai de um componente e entra em outro. Entre a porta de saída do componente de origem e a porta de entrada do componente de destino, não deve existir nenhum meio de propagação do sinal, de tal forma que a transferência de sinal é feita instantaneamente. Se existir um meio de propagação, esse também deve ser considerado um componente, como é o caso da fibra óptica.

Esse sinal deve ser abstraído a fim de se ter um modelo possível de ser implementado computacionalmente. Em princípio, para poder observar os fenômenos de interesse descritos anteriormente, que são de natureza contínua, esse sinal também deveria ser modelado na forma contínua no tempo. Isso seria muito inconveniente para a simulação a eventos discretos.

No entanto, baseando-se no teorema da amostragem de Nyquist, que diz que se um sinal contínuo for amostrado de tal forma que a freqüência de amostragem seja maior que duas vezes o maior componente de freqüência do sinal contínuo, então não há perda de informação no sinal amostrado. Portanto, para a abstração da simulação, pode-se considerar um sinal discreto no tempo obtido pela amostragem do sinal contínuo. A freqüência de amostragem deve ser maior que duas vezes o maior componente de freqüência do fenômeno contínuo que se deseja observar, a fim de que o mesmo seja percebido.

É importante notar que, apesar de ser discreto no o tempo, o sinal não é quantizado, ou seja, as amostras podem assumir qualquer valor dentro de uma faixa contínua de valores. A amostra do sinal é na verdade um conjunto de informações a respeito do sinal no momento da amostragem. As informações são relativas à luz, se o sinal for óptico; ou

relativas às grandezas elétricas, se o sinal for elétrico. No caso de sinal óptico, as informações a respeito da luz são: amplitude, fase, comprimento de onda e polarização. No caso de sinal elétrico, as informações são: tensão e corrente.

Uma amostra do sinal deve sempre estar relacionada com o instante de tempo da amostragem. A associação entre os dados de amostragem do sinal e o tempo de amostragem forma uma estrutura que é chamada de mensagem. Portanto, o sinal que vai de um componente a outro pode ser modelado por uma seqüência de mensagens.

4.4.2 Processos

Neste trabalho, o termo "processo" se refere a uma entidade computacional que executa um conjunto de instruções para gerar dados de saída a partir de dados de entrada. A execução dessas instruções é chamada de processamento. Um processo pode ter variáveis internas que representam seu estado, e que também são usadas para gerar os dados de saída.

Em geral, os componentes de um sistema de comunicações ópticas transformam os sinais ópticos ou elétricos de suas entradas em outros sinais ópticos ou elétricos em suas saídas. Assim, se os sinais de entrada e de saída do componente forem modelados como fluxos de mensagens, um componente pode ser visto como um processo que gera mensagens em suas portas de saída a partir de mensagens que entram nas suas portas de entrada.

Quando uma mensagem chega à porta de entrada, o processo deve aplicar as informações dessa mensagem ao modelo matemático do componente e, juntamente com o seu estado armazenado em variáveis, gerar uma ou mais mensagens em suas portas de saída. Outros processos, cujas portas de entrada estão conectadas às portas de saída do processo atual, receberão as mensagens geradas, e executarão o mesmo procedimento.

Dessa forma, um sistema de comunicação óptica passa a ser um conjunto de processos conectados de tal forma que exista um mapeamento direto entre componentes e processos. As conexões entre processos devem formar a mesma topologia das conexões físicas reais dos componentes.

A Figura 4.2 ilustra o que foi explicado até agora:

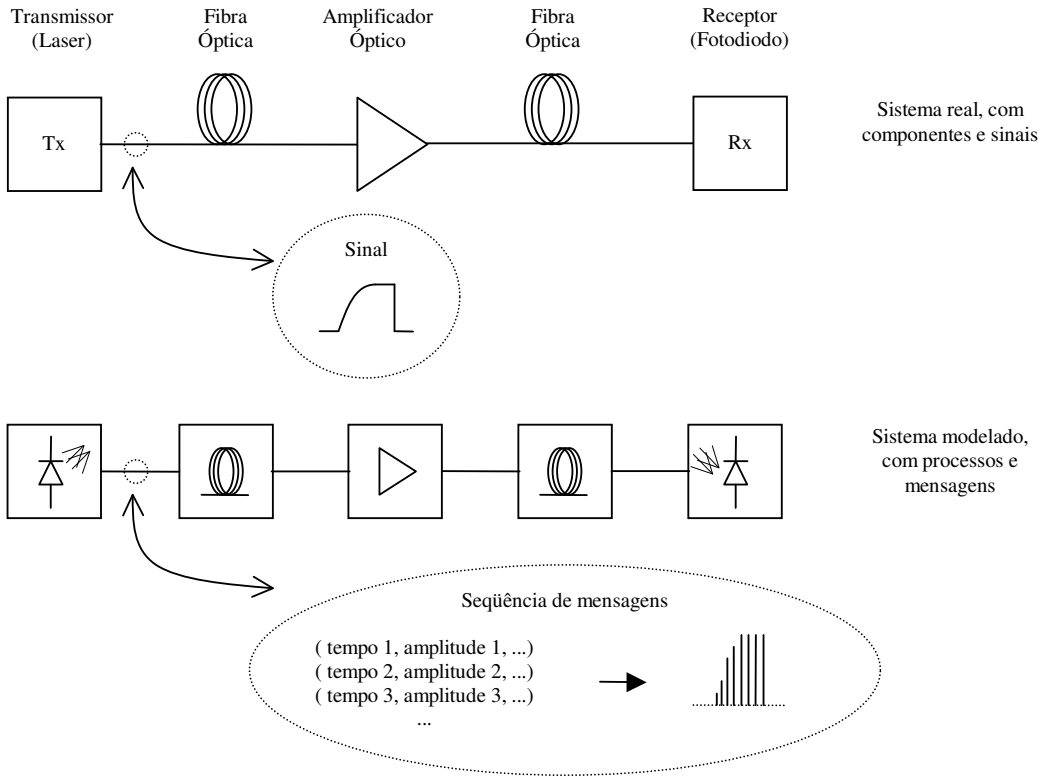

Figura 4.2 - Sistemas real e modelado

Visando a simplificação, na Figura 4.2 não são explicitados os processos referentes aos componentes elétricos conectados ao laser e nem ao fotodiodo.

Uma grande vantagem computacional em se adotar um processo para cada componente, é que se torna mais fácil e direto implementar o modelo matemático do componente. Isso porque cada modelo matemático fica isolado de outro no processamento. A interação entre os modelos é feita apenas com os resultados dos processamentos individuais, ou seja, as mensagens.

Por exemplo, em um laser semicondutor, a modelagem matemática é feita pelas equações de taxa. Quando uma mensagem chega em sua porta de entrada com informações a respeito do sinal elétrico (corrente), essas são aplicadas nas equações de taxa, produzindo informações a respeito do sinal óptico que estarão na mensagem de saída.

Esse tipo de abordagem, onde um processo representa um componente, é chamado de orientação a processos, e será complementada adiante.

4.4.3 Atrasos

Em sistemas de comunicação óptica, a luz que se propaga pelos componentes sofre atrasos devido ao tempo de propagação em cada componente. Todos os componentes introduzem atrasos, mas isso é mais evidente em componentes que têm um comprimento óptico grande, o que significa ter um longo percurso de propagação da luz, ou um alto índice de refração do meio, ou ainda, a combinação dos dois. Por exemplo, quando a luz entra por uma das pontas de uma fibra óptica de 10 km de comprimento, a luz não sai instantaneamente na outra ponta, mas depois de alguns micro-segundos. Já para um simples acoplador, o atraso é praticamente imperceptível, pois o comprimento do caminho óptico é muito pequeno.

Analogamente, quando um processo gera as mensagens de saída, essas devem ter os tempos associados maior que o tempo associado à mensagem entrante. O incremento de tempo deve ser igual ao atraso provocado pelo componente.

Em topologias de enlace óptico, onde os processos são dispostos um após o outro, uma mensagem gerada na extremidade de transmissão desencadeia uma seqüência de geração de mensagens, de tal forma que a última mensagem gerada deve ter um tempo associado igual ao tempo associado à mensagem inicial mais a soma de todos os atrasos dos processos.

Será visto adiante que atrasos são crucialmente importantes em topologias com realimentação.

4.5 Adaptações da simulação a Eventos Discretos

A simulação a eventos discretos abordada no capítulo 2 foi focada em simulações de sistemas com um único processo. A simulação de um único componente, por exemplo, se encaixaria bem nesse contexto.

Se essa abordagem fosse usada para simular todo um sistema de comunicação óptica, o sistema deveria ser visto como sendo um único processo contendo todos os componentes. Nesse caso, o estado seria um conjunto de várias variáveis referentes aos “sub-estados” de

cada componente, sendo que um estado pode diferir de outro apenas pela diferença de uma única variável de estado. Além disso, a função de transição de estados deveria ser uma complexa composição de todas as funções de transição de estados de cada componente. Se um evento referente a um único componente ocorre, alterando uma única variável de estado, então o sistema como um todo transitaria para um novo estado. Quanto mais componentes tiver o sistema, maior será o número de variáveis de estado e mais complexa será a função de transição de estados. Essa abordagem pode se tornar inconvenientemente complexa quando houver muitos componentes.

A orientação a processos, introduzida em 4.3.2, é muito mais interessante para esse caso. A seguir são apresentas as adaptações desenvolvidas para que, juntamente com as abstrações promovidas por mensagens e processos, se possa implementar a simulação a eventos discretos com orientação a processos.

4.5.1 Mensagens e eventos

No método de simulação a eventos discretos apresentada no capítulo 2, a base da simulação era a lista de eventos que continha os eventos que iriam ocorrer, ordenados de acordo com o instante de tempo de ocorrência.

Mas ao usar o conceito de mensagens e processos para modelar o sistema, o número de eventos necessários para o sistema se reduz a um. Esse evento é a “chegada de mensagem”. Processar a mensagem entrante, gerar mensagens de saída, e atualizar o estado são as únicas tarefas do processo, e essas podem ser executadas de uma única vez, no momento da chegada da mensagem.

Por isso, apesar de ser uma simulação a eventos discretos, é mais conveniente tratar mensagens ao invés de eventos. Conseqüentemente, em vez de um processo ter uma lista de eventos a serem executados, é mais conveniente usar uma fila de mensagens. Nesta fila estariam as mensagens que chegariam às portas de entrada do processo e que foram geradas por outros processos, cujas portas de saída estão conectadas às portas de entrada do processo em questão. As mensagens devem estar ordenadas cronologicamente, ou seja, em ordem crescente em relação aos seus respectivos tempos associados. Assim, a próxima mensagem a ser processada pelo processo sempre é a primeira da fila.

4.5.2 Fila de mensagens global

Na orientação a processos, seria natural atribuir uma fila de mensagens para cada processo, como foi explicado anteriormente. Mas fazendo isso, surge o problema de sincronismo de mensagens, típico de topologias onde existe o acoplamento de sinais.

Supõe-se que um processo recebe uma mensagem M_1 com um tempo associado T_1. M_1 é então processado e mensagens de saída são geradas, com tempos associados acrescidos do atraso do componente. Se uma outra mensagem, M_2, chegar ao processo, com um tempo associado menor que o anterior, $T_2 < T_1$, então há uma inconsistência temporal. A posterior chegada de M_2 é incorreta, pois pode ter deixado de alterar o estado do processo no momento do processamento de M_1, fazendo com que as mensagens geradas a partir de M_1 ficassem inconsistentes. Por isso, um processo não pode processar uma mensagem sem ter certeza de que nenhuma outra mensagem, com tempo associado menor, irá chegar.

Isso mostra que é necessário que as mensagens de todo o sistema sejam sincronizadas, de tal forma que uma mensagem não chegue em um processo antes que outra mensagem com um tempo associado menor. O processamento sincronizado das mensagens é possível substituindo todas as filas de mensagens de cada processo por uma única fila de mensagens global.

Essa fila de mensagens é uma fila que contém as mensagens de todos os processos ordenadas cronologicamente. A próxima mensagem a ser processada, dentre todas do sistema, deve ser a primeira mensagem dessa fila. Dessa forma, garante-se que nenhuma mensagem, com um tempo associado maior, será processada antes que outra, com tempo associado menor.

Essa restrição no processamento das mensagens faz surgir a concepção de um tempo global. Todo o sistema percebe a mesma passagem de tempo simulado, que avança de acordo com o tempo associado da primeira mensagem da fila. Assim que uma mensagem é processada, ela deve ser retirada da fila, dando lugar à próxima mensagem. O tempo global passa a ser, então, igual ao tempo associado a essa nova primeira mensagem da fila. Por convenção, o tempo global é chamado de tempo de simulação.

O processamento da primeira mensagem da fila gera novas mensagens, cujos tempos associados são iguais ao tempo de simulação atual mais o tempo de atraso do componente. Essas novas mensagens devem ser inseridas na fila antes do processamento da próxima mensagem, e obedecendo a ordem crescente dos tempos associados das mensagens na fila.

Com o uso da fila de mensagens global, se torna necessária a informação de qual processo e qual porta de entrada a que uma mensagem se destina. Essa informação pode ser incorporada na própria mensagem. Assim a mensagem passa a ter as seguintes informações: dados da amostragem do sinal, tempo associado, processo de destino, e porta de entrada do processo de destino.

4.5.3 Realimentação

Em topologias do tipo ponto-a-ponto, como em enlaces ópticos, os processos são ordenados em cascata, um após o outro. Devido a essa característica seqüencial, uma forma de simular um enlace óptico, sem necessariamente usar eventos discretos, é processar completamente o primeiro componente, para todo o intervalo de simulação. Depois, com os dados de saída do primeiro componente, processar completamente o segundo, e assim por diante até o último componente. Essa forma é simples, e é aplicável também para topologias de distribuição.

Porém, essa forma de simulação não pode ser aplicada em topologias como o da Figura 4.3, onde ocorre uma realimentação.

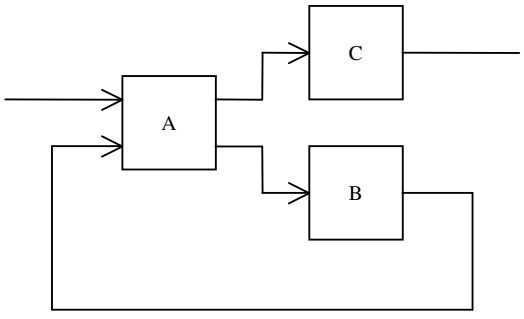

Figura 4.3 - Topologia com realimentação

Na Figura 4.3, o processo A não pode ser completamente simulado para todo o intervalo de tempo da simulação, pois há uma dependência dos resultados da simulação do processo B, que por sua vez, depende do processamento de A. Se caso o processo A for simulado sem os dados do processo B, os resultados serão inconsistentes.

Mas processando uma mensagem de cada vez, esse problema pode ser contornado. Quando A gera uma mensagem para B, esse gera uma nova mensagem que entra em A novamente. Se a mensagem de A para B tem um tempo associado T, então a mensagem de B para A terá um tempo associado de $T + t_B$, onde t_B é o atraso provocado por B. A mensagem de B para A é necessária para a geração da próxima mensagem de A para B (ou de A para C) em um tempo posterior.

É importante notar que se os atrasos inseridos por A e B forem nulos, o tempo de simulação nunca passará adiante, pois as mensagens trocadas por A e B sempre serão os de menor tempo associado, e sempre serão os primeiros da fila de mensagem. Os processamentos de A e B continuarão indefinidamente, gerando continuamente mensagens para o mesmo instante de tempo. Essa condição de *looping* infinito é chamada de *Deadlock*.

4.6 Dinâmica de simulação

O elemento central da dinâmica de simulação é a fila de mensagens global, por onde todos os processos recebem e enviam mensagens de e para outros processos. A execução da simulação consiste em fazer com que cada processo processe as mensagens atribuídas que estão na fila de mensagens, sendo que o processamento de uma mensagem gera outras, que devem ser inseridas na fila de mensagens. Uma vez que a mensagem é processada, essa deve ser retirada da fila e descartada.

Nota-se que há um "ciclo de vida" das mensagens, já que elas são geradas, produzem outras, e depois são eliminadas. ("nascem", "reproduzem", e "morrem"). Da biologia para a computação, cada ciclo se traduz em uma iteração, o que faz com que a execução da simulação consista em executar iterações repetidamente.

Uma iteração, por sua vez, é um conjunto de passos que devem ser executados, que são:

1. Atualizar o tempo global de simulação para o valor do tempo associado da primeira mensagem da fila de eventos.
2. Enviar a primeira mensagem da fila de eventos para o processo e porta de entrada correspondente.
3. Processar a mensagem.
4. Inserir as novas mensagens geradas na fila, obedecendo a ordem cronológica.
5. Retirar a primeira mensagem da fila.

Como exemplo, considera-se o mesmo sistema da Figura 4.2. Supõe-se que existam duas mensagens na fila de mensagens, msg1 e msg2, conforme ilustrado na

Figura 4.4. Apesar de fazerem parte da estrutura da mensagem, os tempos associados são mostrados separadamente, a fim de explicitar a ordenação cronológica da fila de mensagem. A mensagem msg1 tem um tempo associado t1, e é destinada ao amplificador óptico em sua única porta de entrada. Já a mensagem msg2 tem um tempo associado t2, e é destinada ao laser semicondutor. Apesar de não estar explícito na figura, considera-se que o laser possui uma porta de entrada por onde entra a mensagem msg2. Na verdade, msg2 é um tipo de mensagem especial, que será explicado adiante. É considerado também que $t1 < t2$. Assim, msg1 deve ser o primeiro da fila de mensagens. Na

Figura 4.4, a primeira mensagem da fila de mensagens é a que está na posição inferior.

Simulando a partir desse cenário, o tempo de simulação deve ser atualizado com o tempo associado da primeira mensagem da fila, ou seja, $T = t1$ (Passo 1). Feito isso, msg1 deve ser enviado para o seu processo de destino, o amplificador óptico, para que seja processado (Passo 2).

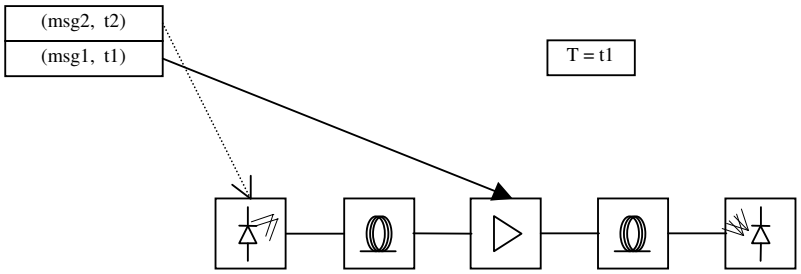

Figura 4.4 – msg1 é enviada ao amplificador óptico

O resultado do processamento de msg1 pelo amplificador óptico (Passo 3) é a mensagem msg3, destinada ao processo da segunda fibra óptica, como ilustrado na Figura 4.5. Supondo que t3 < t2, msg3 deve ser inserida na fila de mensagens em uma posição anterior a msg2 (Passo 4). Ao mesmo tempo, msg1, que já foi processada, deve ser retirada da fila (Passo 5). Nesse ponto, a iteração de processamento da mensagem msg1 é finalizada.

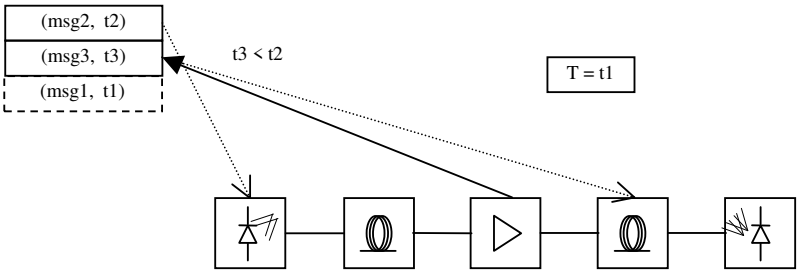

Figura 4.5 – msg3 é inserida na fila de mensagens

A Figura 4.6 e a Figura 4.7 mostram a iteração para msg3.

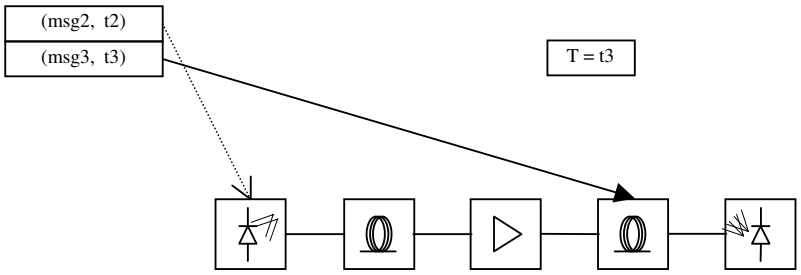

Figura 4.6 – msg3 é enviada à segunda fibra óptica

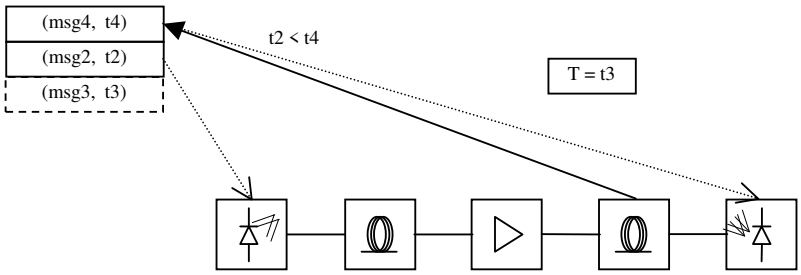

Figura 4.7 – msg4 é inserida na fila de mensagens

Até esse ponto, sempre foi dito que uma mensagem gera outras quando é processada. Mas, em geral, a cadeia de geração de mensagens têm um fim em um processo, como é o caso do fotodetector da Figura 4.4. Por isso, se novas mensagens não forem repostas no sistema, a fila de mensagens ficará vazia depois de algumas iterações. Em outras palavras, o que foi considerado até agora é a propagação do sinal pelos componentes do sistema, não a sua geração.

Para implementar a geração do sinal, é necessário que haja processos que gerem mensagens independentemente de outros processos. Esses processos são os processos correspondentes a componentes ativos do sistema de comunicação óptica, tal como o laser semicondutor.

Para se adequar ao mecanismo de mensagens, o processo de componente ativo deve enviar uma mensagem cujo destinatário é ele mesmo, mas em um tempo futuro. Quando essa mensagem chegar, o processo deve gerar uma mensagem na sua porta de saída para o próximo componente e, ao mesmo tempo, deve gerar uma nova mensagem para si próprio, engatilhando a próxima geração de mensagens. Com esse auto-envio de mensagens, garante-se uma fonte de mensagens inesgotável.

Um outro problema parecido com esse é que inicialmente a fila de mensagens está vazia. Assim, não é possível nem começar a simulação, pois com nenhuma mensagem para gerar outras, nada acontece. Com a fila de mensagens vazia, não pode haver simulação.

Para resolver esse problema, o simulador, no início da simulação, deve executar uma rotina de partida dos processos de componentes ativos, de tal forma que uma primeira mensagem auto-enviada seja inserida na fila de mensagens. Assim, quando a simulação

propriamente dita começar, a fila de mensagens terá mensagens suficientes para desencadear o ciclo de geração de mensagens e a propagação do sinal pelo sistema.

Portanto, uma simulação não acontece se não houver processos de componentes ativos. Isso é totalmente coerente com um sistema real, pois esse não funciona se não houver uma fonte de luz (componente ativo) para gerar o sinal.

Depois de iniciada a simulação, as iterações são repetidas até que se encontre uma condição de término de simulação. Se a simulação ocorrer bem, a condição de término é algo pré-estabelecido, como o tempo limite de simulação atingido ou número máximo de iterações atingido. Mas, se por algum motivo anormal, a fila de mensagens esvaziar, a simulação também deve ser encerrada.

Durante a simulação, e durante o processamento de uma mensagem, o processo deve não apenas gerar outras mensagens e atualizar as variáveis de estado, mas também produzir dados que serão os resultados da simulação. Esses dados podem ser simplesmente algum valor da amostragem, que é diretamente o valor presente na mensagem, ou algum valor relacionado ao modelo matemático do componente.

Por exemplo, em um amplificador óptico, é interessante observar o valor da intensidade da luz que entra (que é diretamente o valor da amostra na mensagem de entrada), e o valor da intensidade da luz amplificada que sai (que é diretamente o valor da amostra na mensagem de saída). Já em um laser semicondutor, um dos resultados a serem observados é o número de portadores, resultante de uma das equações de taxa do modelo.

4.7 Modelagem dos componentes

Cada processo aplica as informações presentes na mensagem de entrada ao modelo matemático do componente, gerando as mensagens de saída e também os resultados da simulação referentes a esse componente. O modelo matemático do componente pode ser desde uma simples repetição de dados, até complexas equações diferenciais.

Por causa do processamento por mensagens, e porque as mensagens carregam informação a respeito de uma amostra do sinal no tempo, os componentes devem ser modelados considerando o domínio do tempo, e não o domínio da freqüência. Todo o

processamento que necessite operar no domínio da freqüência deve ser feito no final da simulação, quando todo o tempo de simulação já passou.

As equações diferenciais dos modelos dos componentes podem ser resolvidas por métodos numéricos, onde o passo das iterações deve ser o intervalo entre mensagens. Os valores envolvidos na resolução numérica das equações diferenciais fazem parte das variáveis de estado, pois esses devem ser persistidos de um processamento de mensagem a outro.

4.8 Conclusão

Nesse trabalho, a modelagem de um sistema de comunicação óptica é feita considerando-se cada componente do sistema como um processo. Usa-se a discretização e amostragem do sinal óptico e elétrico, e o encapsulamento dos eventos e outras informações em mensagens. Os processos produzem mensagens e as trocam entre si, de forma ordenada e sincronizada, o que é garantido pela fila única e global de mensagens.

Como as mensagens da fila estão ordenadas cronologicamente, a execução da simulação consiste em processar sempre a primeira mensagem da fila, e inserir na fila as novas mensagens produzidas por esse processamento. A inserção de mensagens deve respeitar a ordem cronológica.

Com isso, é possível simular topologias com realimentação, sem que ocorram erros de causalidade. Porém, em topologias que apresentam realimentação, é importante que haja atrasos entre a mensagem que entra no processo e as mensagens produzidas pelo processo, para que não ocorra *deadlock*.

5 Implementação

5.1 introdução

A adaptação do método de simulação a eventos discretos, orientado a processos, à simulação de sistemas de comunicação óptica, desenvolvida no capítulo 4, foi implementada em um software. Esse software é o simulador de sistemas de comunicação óptica LightSim.

A forma encontrada para tornar fácil a inserção de modelos no simulador, e ao mesmo tempo deixar o software flexível para manutenção e futuras expansões, foi a modularização. A estrutura modular e a arquitetura do software serão abordados nesse capítulo.

Assim, nas próximas sessões, a abordagem é mais focada no lado computacional e de engenharia de software. A implementação do LightSim é apresentada a seguir, e os resultados são apresentados no capítulo 6.

5.2 Requisitos do simulador

Trabalhos anteriores [1], [2], implementaram um software de simulação de enlaces ópticos, o PCSimfo. Esse software utiliza a técnica de simulação SDF, com taxa de repetição dos processos igual a 1 ($n_x = 1$). Além disso, a simulação se dá no domínio da freqüência. As limitações do PCSinfo foram apontadas no capítulo 1, mas são repetidas a seguir:

- A simulação de sistemas cuja topologia apresenta realimentação não é possível. Devido à técnica de simulação utilizada, o PCSimfo simula apenas enlaces ópticos do tipo ponto-a-ponto ou ponto-multiponto.
- A inclusão de um modelo de componente ao simulador é uma tarefa trabalhosa, e exige conhecimento detalhado da implementação do código fonte, além da recompilação de todo o software.
- A manutenção e expansão do software são tarefas difíceis.

Na fase inicial do projeto LightSim, foram elaborados os requisitos do simulador, visando a simulação de sistemas de comunicação óptica em geral e a eliminação das limitações acima. São eles:

1. Simulação de sistemas de comunicação óptica do tipo ponto-a-ponto, ponto-multiponto e redes, com ou sem realimentação.
2. Fácil inclusão de modelos por parte do usuário final do simulador.
3. Fácil manutenção e expansão do software.
4. Interface gráfica de fácil entendimento e amigável ao usuário.

O primeiro requisito é o mais complexo, e é satisfeito ao se usar a técnica de eventos discretos com as adaptações descritas no Capítulo 4.

Já os outros requisitos são dependentes da arquitetura do software, e são abordados a seguir.

5.3 Arquitetura do software

5.3.1 Visão geral

Para que um software seja de fácil manutenção e expansão (além de fácil inclusão de modelos, como será visto adiante), é imprescindível que o mesmo seja projetado visando um alto grau de modularidade.

O software do simulador foi implementado usando a linguagem C++, com orientação a objetos. A orientação a objetos faz com que o software seja modular ao nível de classes e objetos, satisfazendo em parte o requisito 3.

Mas essa modularidade da orientação a objetos ainda é muito granular. Um agrupamento de classes foi elaborado a fim de se ter modularidade a um nível maior. Assim, três grandes módulos de software foram criados: interface gráfica, *kernel* e modelos, como mostra o diagrama de pacotes (UML) da Figura 5.1.

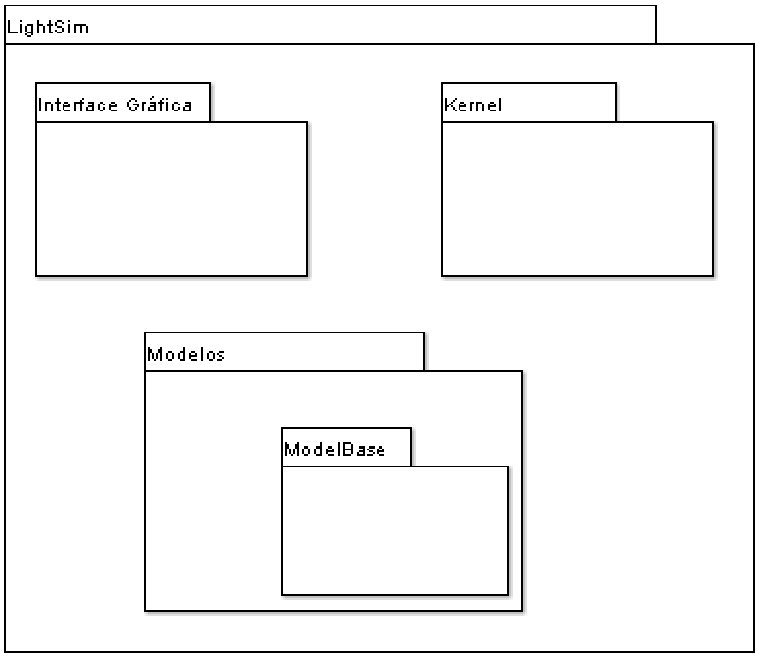

Figura 5.1 – Módulos do LightSim (em forma de pacotes)

Esses módulos são explicados a seguir.

5.3.2 Interface gráfica

A interface gráfica é a interface entre o usuário e o simulador, e é onde ocorre toda interação entre essas duas partes. Portanto, é muito importante que a interface gráfica seja a mais intuitiva possível, e que apresente de forma fácil todas as opções de simulação, os controles e comandos, e também os resultados da simulação.

Os recursos gráficos de um software são dependentes do sistema operacional sobre o qual o software é executado. Essa característica é inevitavelmente presente no módulo de interface gráfica. Pensando nisso, os módulos do LightSim foram projetados de tal maneira a confinar toda a dependência ao sistema operacional no módulo de interface gráfica, seja pela utilização de recursos gráficos ou não-gráficos. Assim, os outros dois módulos podem ser reutilizados na implementação do simulador em outros sistemas operacionais, bastando para isso implementar uma nova interface gráfica.

Além disso, no LightSim, a interface gráfica é o programa executável principal (EXE), e os outros módulos são compilados em bibliotecas dinâmicas (DLLs). Assim, garante-se uma modularidade ainda maior, já que, se for mantida a interface entre os módulos (API), um módulo pode ser alterado ou substituído sem a necessidade de alterar ou recompilar os outros.

O sistema operacional para o qual a interface gráfica do LightSim foi desenvolvida é o Microsoft Windows. Há uma intensa utilização de janelas, botões, menus, recursos do mouse, e outras facilidades disponíveis, como mostra a Figura 5.2.

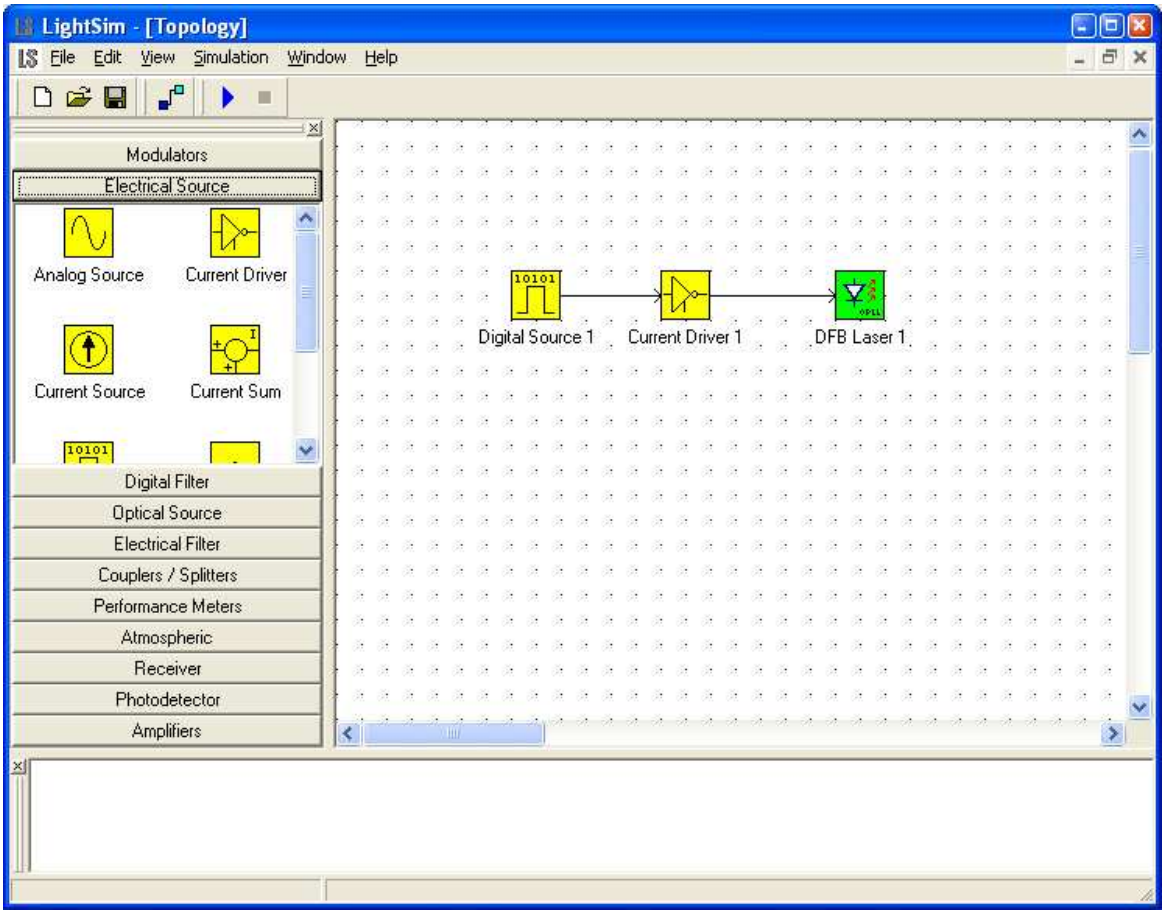

Figura 5.2 – Interface gráfica do LightSim (tela principal)

Na tela principal, o usuário pode criar a topologia desejada arrastando com o mouse os componentes disponíveis na janela de componentes (barra da esquerda) para a janela de

topologia (central), e conectando os componentes entre si através da ferramenta de conexão. Ao pressionar duas vezes com o botão esquerdo no mouse em cima de um componente, uma janela com todos os parâmetros do componente é aberta, e o usuário pode alterar os valores. A Figura 5.3 mostra a janela de parâmetros de uma fonte digital:

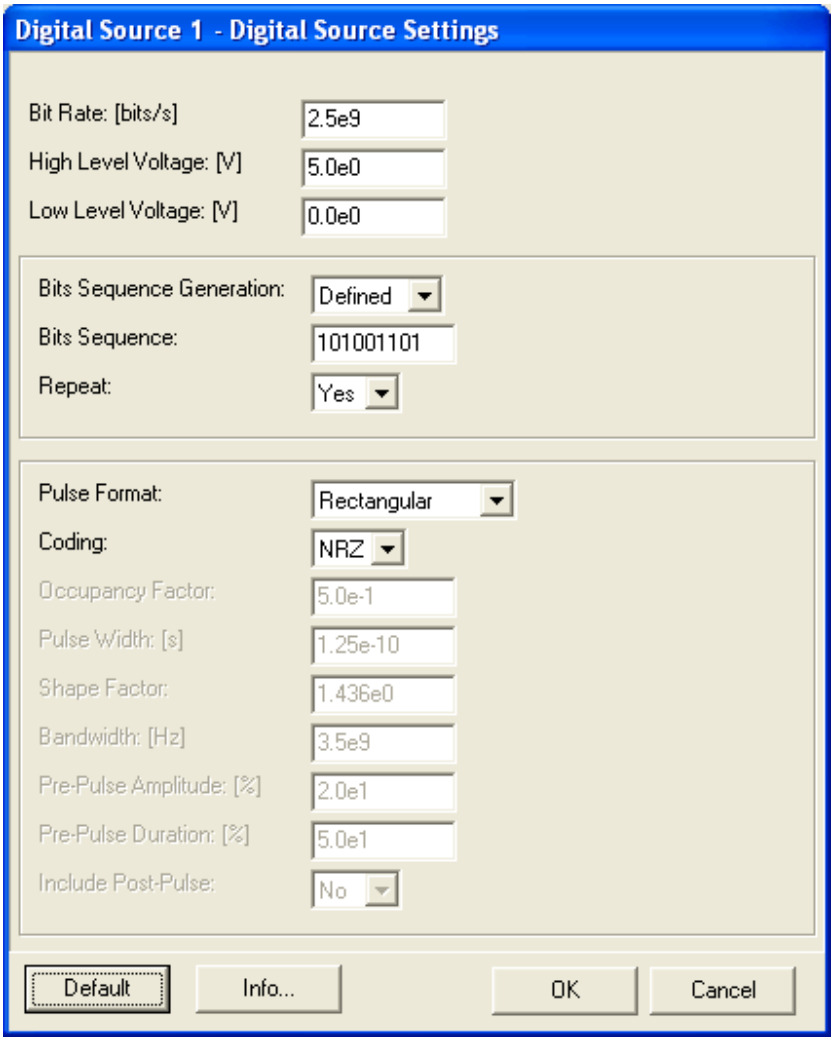

Figura 5.3 – janela de parâmetros de uma fonte digital

Uma vez que todos os componentes estão postos na janela de topologia, conectados de acordo com a topologia desejada, e com os parâmetros de componente desejados, basta pressionar o botão de início de simulação, que está na barra de ferramentas, para iniciar o processo de simulação.

Ainda antes da simulação propriamente dita, é requisitado ao usuário definir o tempo limite de simulação. O usuário deve ter noção do tempo de simulação através, por exemplo, da taxa de bits do gerador de bits.

Feito isso, a simulação é iniciada. Durante a simulação, mensagens de aviso, informação ou erro (ou ainda de *debug*) são mostradas na janela de *log* (inferior). A mensagem de término de simulação também é mostrada nessa janela.

Terminada a simulação, os resultados podem ser acessados pressionando o botão direito do mouse sobre o componente desejado. Um menu *popup* é aberto, contendo todos os resultados para aquele componente. Ao selecionar um dos resultados do menu, uma janela contendo o gráfico correspondente é aberta. Nesta janela, algumas ferramentas de visualização são disponíveis para facilitar a análise. A Figura 5.4 mostra uma janela de visualização de gráfico:

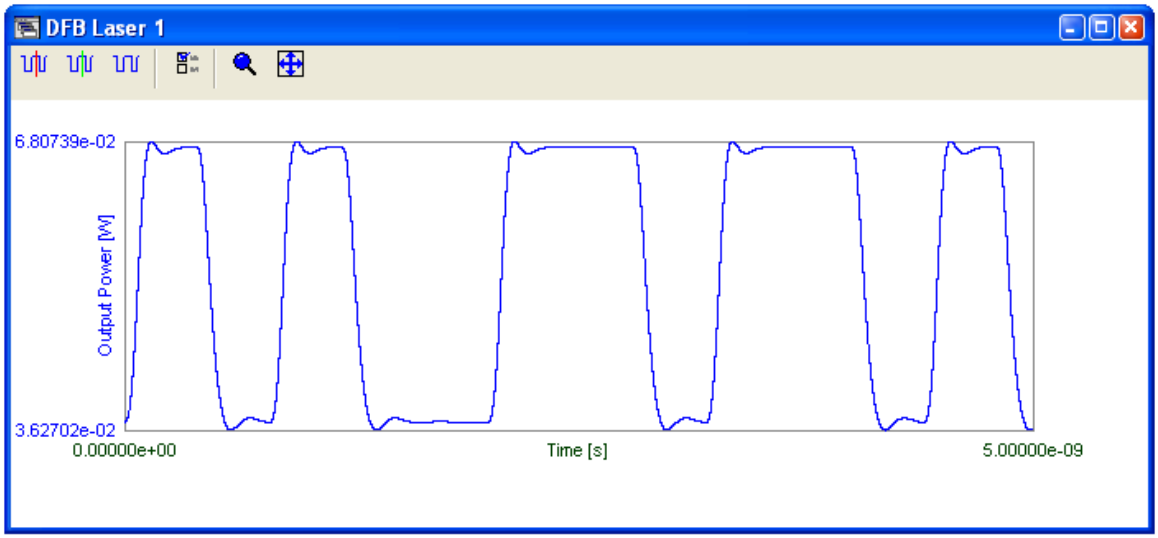

Figura 5.4 – Janela de visualização do gráfico da potência óptica de saída de um laser DFB

Nota-se que na interface gráfica existem vários elementos gráficos, mas todos agrupáveis pela funcionalidade. Esse agrupamento se dá pelas janelas: janela principal (*frame* principal), janela de topologia, janela de *log*, janela de componentes, janela de parâmetros, janela de gráfico, janela de opções de simulação (tempo limite de simulação) e

janela de ajuda (*about window*). Do ponto de vista de implementação, cada janela pode ser traduzida em uma classe.

A Figura 5.5 mostra o diagrama de classes do módulo de interface gráfica.

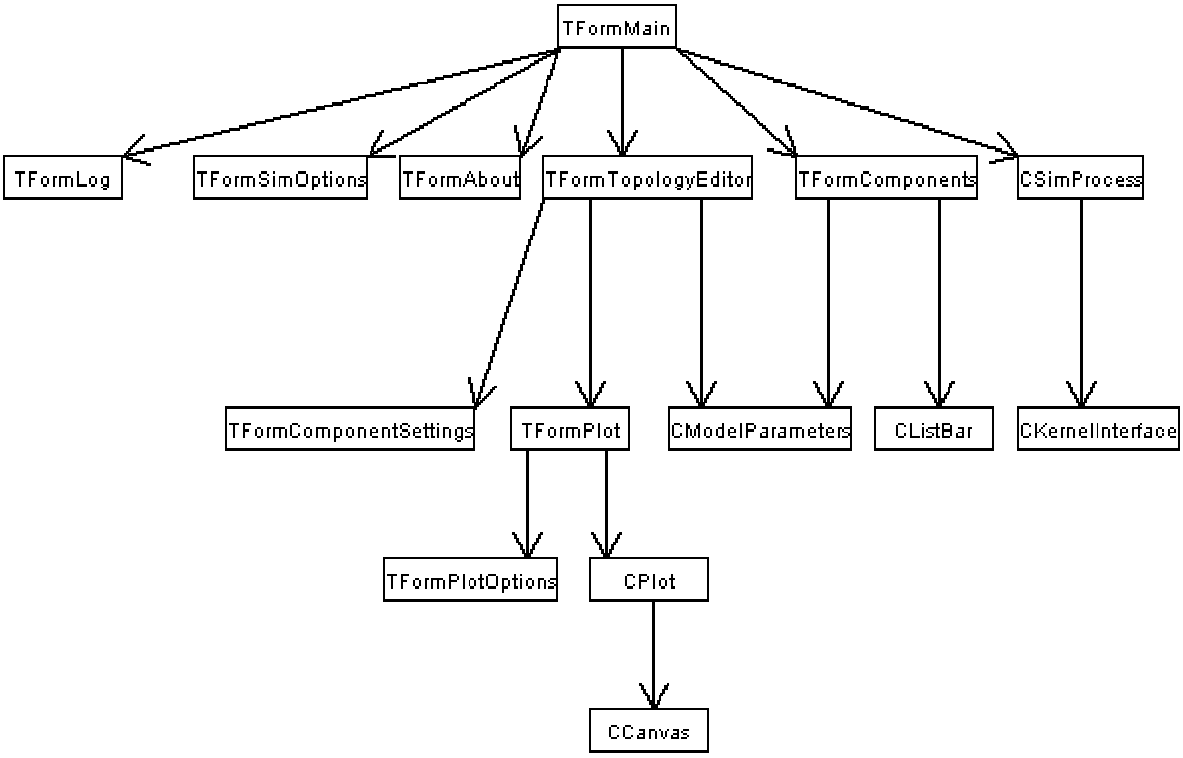

Figura 5.5 - Diagrama de classes (UML) simplificado do módulo de interface gráfica

As classes cujo prefixo “TForm” aparece no nome são classes que implementam janelas e suas funcionalidades. As classes “CPlot” e “CCanvas” implementam o desenho dos gráficos nas janelas de gráfico. Já a classe “CListBar” implementa a lista de componentes na janela de componentes.

Todas as classes da interface gráfica implementam recursos gráficos, exceto três: “CModelParamenters”, “CSimProcess” e “CKernelInterface”. A primeira é um encapsulamento dos parâmetros dos modelos, que são de diversos tipos: números, opções, seqüência de caracteres, etc. A classe “CModelParameters” faz com que as outras classes possam tratar esses parâmetros de forma genérica.

A classe “CSimProcess” é encarregada de criar uma *Thread* especialmente para o processo de simulação, separada do programa principal. Assim, o processo de simulação não bloqueia a interface gráfica, permitindo que o usuário interaja com o simulador mesmo durante a simulação. Por exemplo, se durante a simulação o usuário quer abortar, basta pressionar o botão de parada de simulação, e a simulação é abortada. Caso não fosse implementada uma *Thread* separada, a ação de pressionar o botão (qualquer botão) seria possível apenas após o término da simulação. Esse é um exemplo de recurso não-gráfico dependente do sistema operacional, confinado no módulo de interface gráfica.

Já a classe CKernelInterface é encarregada de ser a interface entre a interface gráfica e o *kernel*. Nessa classe são implementadas as diversas APIs comuns também ao *kernel*, tornando possível a comunicação entre as duas partes de forma organizada e eficiente.

5.3.3 *Kernel*

O *kernel* é o núcleo do simulador, e é onde se encontra o *engine* (“motor”) de simulação a eventos discretos.

Nesse módulo, não são usadas bibliotecas ou classes específicas de plataforma ou sistema operacional. Assim, a menos de pequenos detalhes de compilação, o mesmo código pode ser usado para compilar programas executáveis em diferentes sistemas operacionais.

Além disso, a forma de entrada de dados no *kernel* é feita através do arquivo de simulação, que é um arquivo textual, com formatação definida, onde se descreve a topologia do sistema e os parâmetros dos componentes. Assim, o *kernel* pode ser compilado como um programa separadamente, e, desde que se forneça o arquivo de simulação, interfaces mais simples, como linha de comando, também podem executá-lo.

O número de classes no *kernel* é reduzido, como mostra a Figura 5.6.

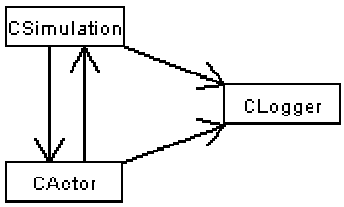

Figura 5.6 - Diagrama de classes (UML) simplificado do módulo *kernel*

A classe “CSimulation” é onde se encontra o *engine* de simulação. Já a classe “CActor” é a classe que faz a interface com o modelo. Para o *engine* de simulação, um componente é na verdade um objeto de “CActor”. Em termos de objetos, existe um objeto da classe “CSimulation” para vários objetos da classe “CActor”.

Já a classe “CLogger” é responsável por enviar mensagens para a interface, com informações a respeito de status, erros, avisos e *debug*.

Além dessas três classes, existem ainda funções (na linguagem C) agrupadas em um arquivo para portar o módulo *kernel* em uma biblioteca dinâmica, DLL. Essa DLL é executada a partir do programa principal (interface gráfica) assim que for requisitado o início do processo de simulação. Mas antes de iniciar o *kernel*, o programa de interface precisa transcrever a topologia criada graficamente pelo usuário, assim como os parâmetros de cada componente, no arquivo de simulação.

No início da simulação, o *kernel* monta uma estrutura de instâncias de modelos (objetos de “CActor”), de acordo com a descrição da topologia no arquivo de simulação. Cada instância de modelo recebe os respectivos parâmetros, também providos pelo arquivo de simulação.

Nesse ponto, é iniciado o procedimento da simulação a eventos discretos adaptada descrita no capítulo 4. Assim, o primeiro passo é invocar a rotina de ignição de todos os modelos de componentes ativos, a fim de popular a fila de mensagens com mensagens iniciais.

A fila de mensagens é implementada através de uma fila de prioridades, onde a prioridade de cada elemento da fila é dada pelo tempo associado da mensagem. Toda vez que uma mensagem é inserida na fila, a mesma é reordenada, obedecendo a prioridade do tempo associado. Assim, a mensagem de menor tempo associado sempre será a primeira da fila.

Após a população inicial da fila de mensagens, o *engine* de simulação é iniciado. O *engine* de simulação é implementado através de um laço de execução, conhecido como *looping*. A cada iteração do *looping*, a primeira mensagem da fila é retirada, o tempo global de simulação é atualizado, e o método de processamento da mensagem, presente no modelo, é invocado. Esse método é invocado através da instância do modelo criada no início da

simulação. Esse mesmo método deve retornar as novas mensagens geradas, que serão inseridas na fila de mensagens.

No processamento da mensagem, a instância do modelo deve armazenar em variáveis o seu estado atual, e também guardar os resultados de interesse para o tempo de simulação corrente (igual ao tempo associado da mensagem corrente) para serem utilizados ou exportados futuramente.

A cada iteração, o tempo global de simulação deve ser comparado com o tempo limite estabelecido pelo usuário. Quando esse tempo for atingido, a simulação deve ser encerrada. No processo de encerramento da simulação, todos os modelos devem armazenar em arquivos textuais os resultados finais da simulação. Esses arquivos são disponibilizados para a interface, a fim de que o usuário possa visualizá-la de forma gráfica.

Também é no final da simulação que os cálculos no domínio da freqüência são realizados. Isso é de se esperar, pois não é possível fazer cálculos no domínio da freqüência referentes a instantes de tempo. É necessário se ter pelo menos uma janela temporal.

5.3.4 Modelos

Um modelo é a implementação de um componente no ambiente de simulação. É o modelo que contém a descrição matemática do componente, e que consome e produz as mensagens. Dentre os modelos criados neste trabalho estão: laser semicondutor DFB, fotodiodo, amplificador óptico semicondutor (SOA), gerador de bits e *driver* de corrente. Os modelos implementados referentes a componentes ópticos geralmente usam a intensidade da luz e o comprimento de onda para a modelagem. Mas modelos podem ser implementados usando outras informações, como por exemplo, a polarização da luz. Isso é possível graças à forma expansível no qual o LightSim foi projetado.

Cada modelo é uma DLL. Durante a simulação, o *kernel* executa a DLL para que o modelo possa processar a mensagem e gerar outras. A interface gráfica também executa a DLL para obter informações a respeito dos parâmetros do modelo para serem exibidos ao usuário na janela de parâmetros do componente.

Todos os modelos possuem o mesmo conjunto de classes, e diferem apenas na implementação de alguns métodos. Nesses métodos estão a definição dos parâmetros do

componente, seus valores padrão, o modelo matemático, o processamento para geração de novas mensagens, e eventuais tratamentos de dados para exportação dos resultados finais.

A Figura 5.7 mostra o diagrama de classes do módulo de modelos.

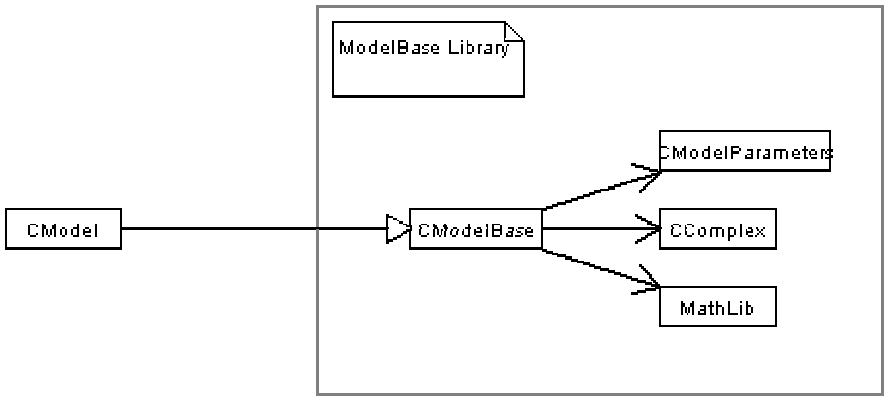

Figura 5.7 - Diagrama de classes (UML) simplificado do módulo de modelos

As classes “CModelBase”, “CModelParameters”, “CComplex” e “MathLib” são reunidos em uma biblioteca estática (LIB) chamada ModelBase. Isso é porque essas classes implementam o comportamento comum a todo modelo.

“MathLib” não é na verdade uma classe, mas um conjunto de funções matemáticas disponíveis ao modelo. A classe “CComplex” é o encapsulamento de números complexos e suas operações. A classe “CModelParameters” implementa o encapsulamento dos parâmetros do modelo (também presente na interface gráfica). Por fim, a classe “CModelBase” implementa as funções de acesso comum aos modelos.

Apenas a classe “CModel”, que é herdeira de “CModelBase”, possui informação específica para cada modelo. Nesta classe estão os métodos cujas implementações ficam a cargo do criador do modelo, a saber:

- “CModel” – esse é o construtor da classe. Nesse método, devem ser definidos o nome do modelo, o número de portas de entrada e saída, e os dados que serão exportados e que serão mostrados na interface gráfica como resultados

de simulação. Outras variáveis também podem ser inicializadas nesse método.

- "SetParameters" – nesse método, devem ser definidos todos os parâmetros do modelo. Cada parâmetro deve ter um nome, variável de armazenamento, valor máximo e mínimo (se for o caso), valor default, e conjunto de opções possíveis (se for o caso). No momento de mostrar os parâmetros do modelo na janela de parâmetros, a interface gráfica executará esse método para obter as informações necessárias.
- "Initialize" – esse método é chamado no início da simulação, a fim de que os modelos de componentes ativos possam fornecer as mensagens iniciais para a fila de mensagens.
- "ProcessEvent" – esse é o método principal do modelo, onde ocorre o processamento da mensagem. O método recebe como argumento o evento incidente, e deve retornar, através de outro argumento, as mensagens produzidas.
- "PostSimProcess" – esse método é chamado no final da simulação, e sua finalidade é de processar dados no domínio da freqüência, como por exemplo, utilizando a transformada rápida de Fourier (FFT).

A criação de um modelo consiste em construir os códigos os métodos acima com a modelagem e parâmetros específicos do modelo, e compilar juntamente com a biblioteca "ModelBase" para gerar uma DLL. Essa DLL deve ser colocada em um diretório predefinido e comum ao *kernel* e à interface gráfica. Se no mesmo diretório existir uma imagem BMP 32x32 com o mesmo nome da DLL, essa será usada como imagem do modelo pela interface.

5.4 Implementação de um modelo: laser semicondutor DFB

Para ilustrar a implementação de um modelo no LightSim, o caso do laser semicondutor DFB é tomado como exemplo e explicado a seguir. Sua simulação já foi

proposta de forma simples no capítulo 2, mas será abordada a seguir com mais detalhes e de forma a se adequar à implementação do método de simulação no LightSim.

Ao mesmo tempo em que o laser semicondutor é um dos modelos mais complexos de serem implementados, é o mais importante para o LightSim, pois é a origem do sinal óptico (mensagens) que propaga por todo o sistema. Praticamente todo sistema simulado possui um laser semicondutor em sua topologia.

O modelo matemático do laser semicondutor DFB monomodo é formado pelas equações de taxa, a saber [2]:

$$\frac{dS(t)}{dt} = \Gamma G(t)(N(t) - N_{tr})S(t) - \frac{P(t)}{\tau_p} + \frac{\Gamma \beta N(t)}{\tau_n} \qquad (5.1)$$

$$\frac{dN(t)}{dt} = \frac{I(t)}{eV} - G(t)(N(t) - N_{tr})S(t) - \frac{N(t)}{\tau_n} \qquad (5.2)$$

$$\frac{d\phi(t)}{dt} = \frac{1}{2}\alpha_{lin}\left[\Gamma v_g a_0 (N(t) - N_{tr}) - \frac{1}{\tau_p}\right] \qquad (5.3)$$

$$G(t) = \frac{v_g a_0}{1 + \varepsilon S(t)} \qquad (5.4)$$

onde $S(t)$ é a densidade de fótons; $N(t)$ é a densidade de portadores (elétrons); $\phi(t)$ é a fase; $G(t)$ é o ganho; Γ é o fator de confinamento; v_g é a velocidade de grupo (c/n_g); a_0 é o coeficiente de ganho; ε é o fator de compressão do ganho; N_{tr} é a densidade de portadores (elétrons) na transparência; $\tau_p = (v_g (\alpha_m + \alpha_{int}))^{-1}$ é o tempo de vida do fóton; α_m é o coeficiente de perdas do espelho; α_{int} é o coeficiente de perdas internas; τ_n é o tempo de vida do elétron (portador); β é a fração da emissão espontânea; I é a corrente injetada; e é a carga do elétron; V é o volume da região ativa; α_{lin} é o fator de largura de linha.

A potência óptica de saída pode ser obtida através da seguinte relação[2]:

$$P(t) = \frac{V \eta h \nu}{2\Gamma \tau_p} S(t) \qquad (5.5)$$

onde *P(t)* é a potência óptica, *h* é a constante de Planck, η é a eficiência quântica total e *v* é a freqüência da luz.

Durante o processamento do modelo, essas equações diferenciais devem ser resolvidas para poder gerar as outras mensagens e os resultados finais da simulação. Os próprios valores da densidade de fótons, densidade de portadores e fase são resultados interessantes, e podem fazer parte dos resultados finais da simulação. O processamento para resolver as equações de taxa e a geração de novas mensagens deve estar no método "ProcessEvent", como descrito em 5.2.4.

Para resolver as equações de taxa, utiliza-se o método numérico de Runge-Kutta de quarta ordem. A execução desse método é composta por iterações, e a cada iteração, um novo passo de cálculos é dado. A cada iteração, novos valores para a densidade de fótons, densidade de portadores e fase são gerados, e esses novos valores são usados para gerar as novas mensagens ópticas na saída óptica do componente.

No exemplo de simulação de um laser DFB mostrado no capítulo 2, foi considerado que um laço de execução (*looping*) era responsável pelas iterações de Runge-Kutta. Mas se esse laço for implementado com "for" ou "while", o processo irá ocupar todo o processamento do software para si, "travando" o resto da simulação. Uma solução seria implementar uma *thread* para esse laço.

Mas existe uma solução mais simples e não menos eficiente: as "auto-mensagens". Essas são mensagens que partem do modelo e são destinadas ao próprio modelo, mas com um tempo associado maior que aquele na qual foram geradas. Uma "auto-mensagem" não carrega dados relevantes a respeito de um sinal, mas apenas indica o instante de tempo futuro para gerar novas mensagens ópticas em sua saída. Ao receber uma "auto-mensagem", o processo deve executar uma iteração Runge-Kutta, gerando mensagens em sua saída óptica. Também deve gerar e enviar uma nova "auto-mensagem" para disparar a próxima iteração, em um momento futuro. Dessa forma, o laser entra em um *looping* infinito de geração de mensagens.

Portanto, o modelo do laser semicondutor DFB pode receber dois tipos de mensagens: elétrica, com dados da corrente de alimentação; e a "auto-mensagem". Quanto a mensagem com dados da corrente de alimentação é recebida, a única ação tomada é a atualização da variável de estado de corrente. Quando uma "auto-mensagem" é recebida, a

iteração de Runge-Kutta é executada tendo como dados de entrada o valor atual da corrente (estado) e os valores da última iteração. Assim, quebra-se o sincronismo entre o recebimento de uma mensagem elétrica de corrente com a geração das mensagens ópticas de saída.

As figuras 5.8 e 5.9 mostram diagramas ressaltando essa quebra de sincronismo:

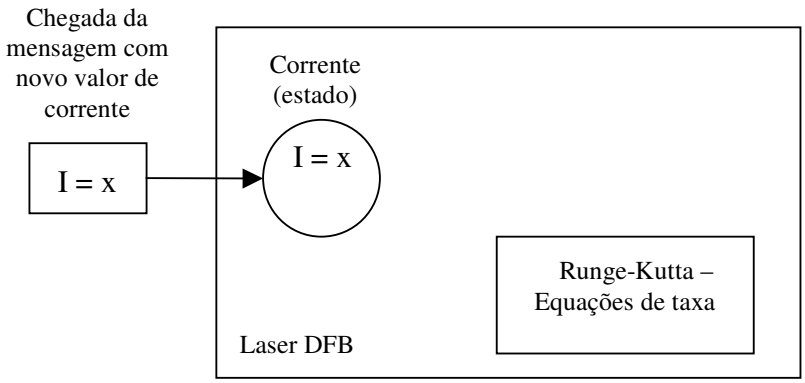

Figura 5.8 – Processamento envolvido na chegada de uma mensagem elétrica com o novo valor de corrente

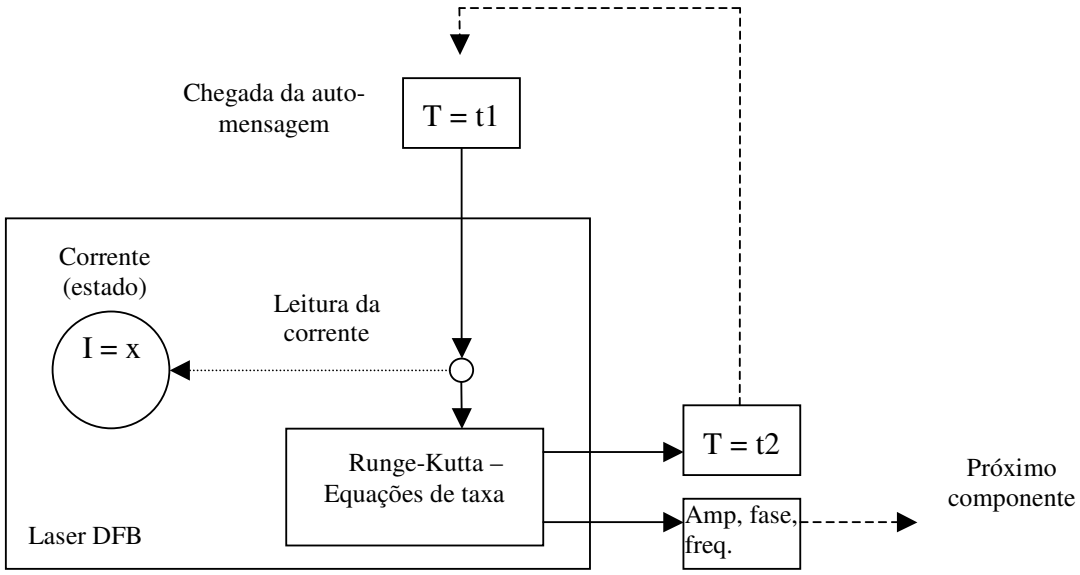

Figura 5.9 – Processamento envolvido na chegada de uma auto-mensagem

O intervalo de tempo entre uma “auto-mensagem” e outra (que é igual ao intervalo entre uma iteração e outra) não pode ser qualquer. Essa deve ser pequena o suficiente para se ter uma convergência no método numérico de Runge-Kutta. Porém, se esse intervalo for feito pequeno demais, a precisão da simulação será muito grande, e o processo de simulação pode demorar muito.

Para ajudar na determinação do melhor intervalo de tempo, é implementada a técnica de passo adaptativo, que calcula o maior passo possível entre uma iteração e outra, sempre mantendo o nível de erro dentro de um limite pré-estabelecido. Cada passo é na verdade o intervalo de tempo entre uma “auto-mensagem” e outra.

5.5 Conclusão

Usando a simulação a eventos discretos orientado a processos, descrito no capítulo 4, implementou-se o simulador LightSim. Esse é um software que foi projetado de forma modular, com a intenção de facilitar a inclusão de modelos no simulador, e também facilitar a manutenção e expansão do software em si.

O LightSim é dividido em três módulos principais, que são: interface grafica, *kernel* e modelos. A interface gráfica é dependente do sistema operacional, pois usa recursos gráficos, tais como janelas, botões, e outros componentes gráficos. O *kernel* é o módulo que contém o *engine* de simulação. Quanto aos modelos, cada um é uma DLL separada, de tal forma que para adicionar um modelo ao LightSim, basta compilar essa DLL e incluí-la na pasta específica; não é necessário recompilar todo o código fonte de todo o software.

6 Resultados

6.1 Introdução

No capítulo 5 foi abordada a implementação da simulação de sistemas de comunicação óptica usando simulação a eventos discretos, conforme explicado no capítulo 4. Essa implementação resultou no simulador LightSim, que é um software executável em um ambiente PC. Portanto, os resultados das simulações do LightSim expressam o desenvolvimento de todo o conjunto.

Neste capítulo, são mostrados resultados de simulações para dois casos: simulação de um laser DFB, e simulação de um sistema com realimentação. No primeiro caso, é mostrada inicialmente a simulação de um laser DFB modelado pelas equações de taxa tradicionais (eq. 5.1 a 5.3), modulado por uma seqüência de pulsos de corrente (bits). Depois, é mostrada a simulação de um laser DFB com uma modelagem alternativa, mas comparando com resultados validados por outro trabalho, a fim de se comprovar o correto funcionamento da implementação do método de simulação no LightSim.

O segundo caso é mais complexo. É simulado um OPLL (*Optical Phase-Lock Loop*), que é um sistema com realimentação. Esse é justamente um dos objetivos desse trabalho.

É importante ressaltar que a intenção das simulações mostradas e das comparações é de validar o método de simulação proposto e sua implementação, e não a modelagem do sistema em questão. Por isso, não é feita uma análise quantitativa dos resultados, mas qualitativa.

6.2 Simulação de um laser DFB

6.2.1 Seqüência de bits

A figura 6.1 mostra a topologia usada para simular um laser DFB modulado diretamente.

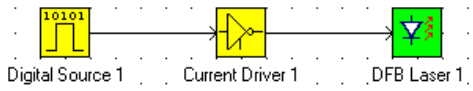

Figura 6.1 – Topologia para simulação do laser DFB

Uma fonte digital (Digital Source 1) fornece uma seqüência de bits para o sistema. Apesar de ser capaz de fornecer bits na forma de pulsos super-gaussianos ou solitrônica, por exemplo, ela está configurada de tal forma que, em sua saída, haja uma seqüência de bits na forma de um trem de pulsos de tensão idealmente retangulares. Sua função no sistema é simplesmente fornecer a seqüência de bits ideais, sem distorções ou ruídos.

Esta fonte digital está configurada com os parâmetros da tabelaTabela 6.1.

Tabela 6.1 – parâmetros da fonte digital

Parâmetro	Valor
Taxa de bits (Gbits/s)	2,5
Tensão do nível lógico “1” (V)	0,01
Tensão do nível lógico “0” (V)	0,0
Seqüência de bits	1010011010000 (não repetitivo)
Formato do pulso	Retangular
Codificação	NRZ (Non Return to Zero)

Para modular o laser DFB com a seqüência de bits gerada, é necessário converter o trem de pulsos de tensão (seqüência de bits), fornecida pela fonte digital, em um trem de pulsos de corrente. Para isso é utilizado um *driver* de corrente (Current Driver 1). Ao fazer a conversão em corrente, é adicionada uma corrente constante de polarização. Assim, garante-se que o nível mínimo de corrente, correspondente ao nível lógico “0”, terá pelo menos o valor da corrente de polarização.

O modelo de *driver* também leva em consideração uma largura de banda finita a fim de adicionar mais realismo ao modelo, já que nenhum dispositivo elétrico real tem uma largura de banda infinita.

Os parâmetros do *driver* de corrente são mostrados na tabela Tabela 6.2.

Tabela 6.2 – parâmetros do *driver* de corrente

Parâmetro	Valor
Condutância (A/V)	1
Corrente de polarização (mA)	40
Largura de banda (GHz)	3,5

A corrente modulada proveniente do *driver* de corrente alimenta o laser DFB (DFB laser 1). Esse, por sua vez, emite luz em sua saída óptica, modulada de acordo com a corrente de entrada.

Seus parâmetros são mostrados na tabela Tabela 6.3.

Tabela 6.3 – parâmetros do laser DFB

Parâmetro	Valor
Comprimento de onda central (m)	$1,3 \times 10^{-6}$
Comprimento da cavidade (m)	$2,5 \times 10^{-4}$
Largura da cavidade (m)	$2,0 \times 10^{-6}$
Espessura da cavidade (m)	$2,0 \times 10^{-7}$
Fator de confinamento	0,3
Índice de refração de grupo	4,0
Fator de ganho de largura de linha	5,0
Coeficiente de perda interna (m^{-1})	4000
Perda dos espelhos (m^{-1})	4500
Coeficiente de ganho diferencial	$2,5 \times 10^{-20}$
Densidade de portadores na transparência (m^{-3})	$1,0 \times 10^{24}$
Fator de recombinação não radioativa (s^{-1})	$1,0 \times 10^{18}$
Fator de recombinação radioativa (m^3/s)	$1,0 \times 10^{-16}$
Fator de recombinação Auger (m^6/s)	$3,0 \times 10^{-41}$
Fator de emissão espontânea	$1,0 \times 10^{-5}$
Fator de compressão de ganho	$5,0 \times 10^{-23}$
Tempo de vida do elétron (s)	$2,2 \times 10^{-9}$
Tempo de vida do fóton (s)	$1,6 \times 10^{-12}$

O laser DFB também é configurado para não simular o transiente, ou seja, quando a simulação começar, o número de portadores já está a um nível de operação em regime.

Simulando com um tempo de simulação de 5 ns, os seguintes resultados são obtidos.

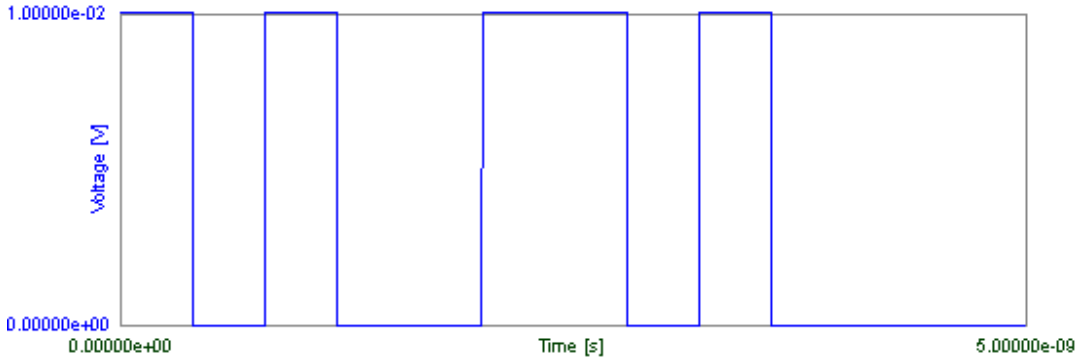

Figura 6.2 – Seqüência de bits na saída da fonte digital (“1010011010000”). Seqüência de bits ideais; nível alto = 0,010 V; nível baixo = 0 V.

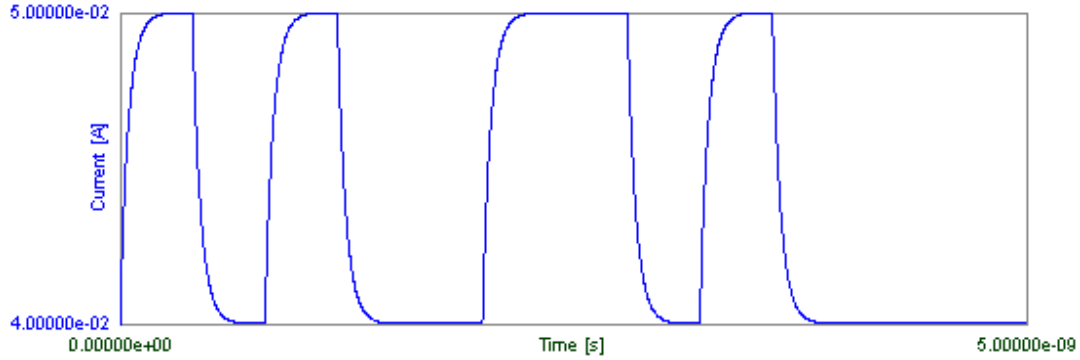

Figura 6.3 – Corrente na saída do *driver* de corrente. Corrente de polarização = 40 mA; condutância = 1 A/V; largura de banda = 3,5 GHz.

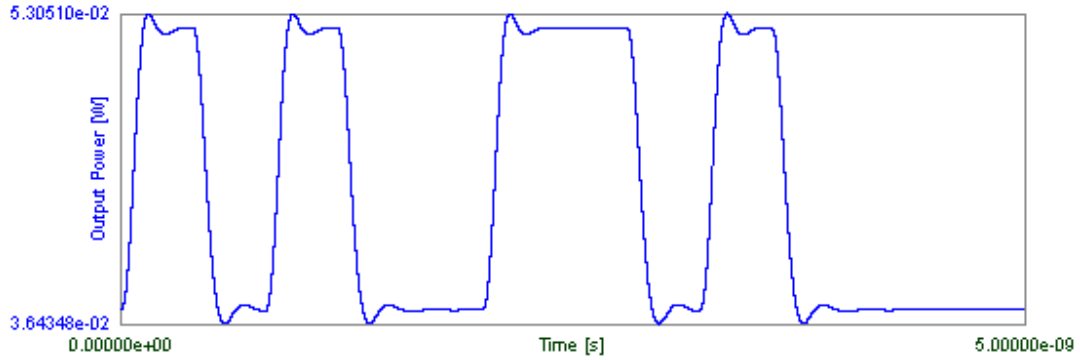

Figura 6.4 – Potência de saída do laser DFB

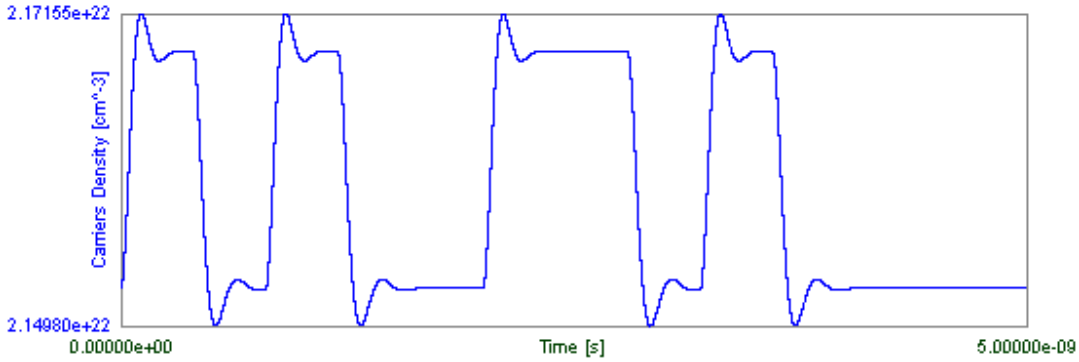

Figura 6.5 – Densidade de portadores do laser DFB

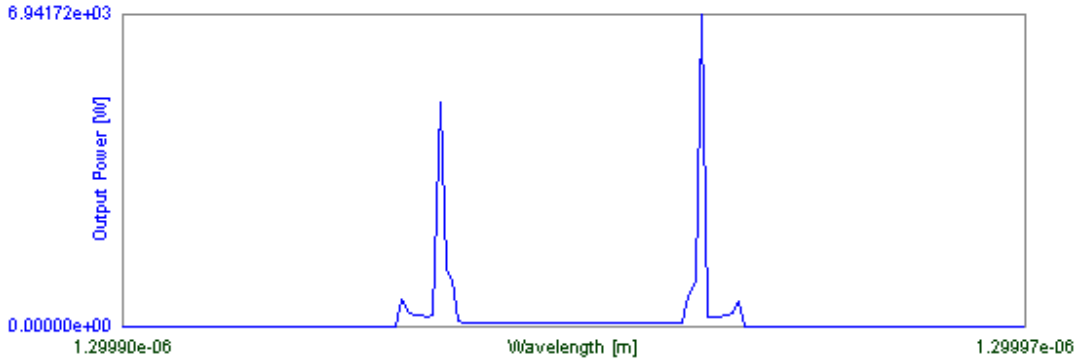

Figura 6.6 – Distribuição espectral de potência óptica (comprimento de onda)

Nos resultados para o laser DFB, principalmente nas Figuras 6.4 e 6.5 (potência óptica e densidade de portadores, respectivamente), nota-se a presença das oscilações de relaxação toda vez que há uma borda de subida (transição do bit "0" para o bit "1"). Como o laser está operando com uma corrente mínima acima da corrente de limiar (I_{th}), na borda de descida (transição do bit "1" para o bit "0") também se nota oscilações.

A Figura 6.6 mostra a distribuição espectral de potência óptica, dado em comprimento de onda. Nota-se que há dois picos, correspondentes ao nível alto e baixo de corrente. A variação de comprimento de onda ocorre devido à proporcionalidade entre o nível de corrente e a freqüência de operação do laser. Esse comportamento será mais explorado no próximo exemplo.

6.2.2 Comparação com resultados validados

Bjerkan, Royset, Hafskjoer e Myhre [10] apresentam uma nova forma para as equações de taxa do laser DFB, onde são usados parâmetros que podem ser estimados a partir de medidas experimentais, de uma forma mais direta. É demonstrado o método usado para se fazer a estimativa desses parâmetros, e, com a nova forma das equações de taxa, são feitas simulações. Os resultados finais da simulação são comparados com resultados experimentais, e nota-se que há boa semelhança entre eles, especialmente quando a corrente de polarização do laser DFB está acima do limiar.

Pode-se considerar que esses resultados são válidos e que servem de referência para se fazer uma comparação com os resultados de simulação do LightSim, usando-se o mesmo modelo. A concordância entre os resultados de [10] e os resultados das simulações do LightSim, usando-se o mesmo modelo de laser e esquemas de simulação, traz mais confiabilidade para o LightSim.

A nova modelagem do laser DFB proposta pode ser derivada das equações 5.1, 5.2 e 5.3, e são escritas da seguinte forma [10]:

$$\frac{dP(t)}{dt} = \frac{B\tau_n I_{th}(X(t)-1)+\frac{1}{\tau_p}}{1+FB\tau_p\tau_c P(t)}P(t) - \frac{P(t)}{\tau_p} + \frac{I_S I_{th} B\tau_n}{F}X(t) \qquad (6.1)$$

$$\frac{dX(t)}{dt} = \frac{I(t)}{I_{th}\tau_n} - \frac{FB\tau_p(X(t)-1)+\frac{F}{I_{th}\tau_n}}{1+FB\tau_p\tau_c P(t)}P(t) - \frac{X(t)}{\tau_n} \qquad (6.2)$$

$$\frac{d\phi(t)}{dt} = \frac{\alpha}{2}B\tau_n I_{th}(X(t)-1) \qquad (6.3)$$

onde [10]:

$$B = \frac{\Gamma g_0}{eV} \qquad (6.4)$$

$$\tau_c = \frac{\varepsilon}{g_0} \qquad (6.5)$$

$$F = \frac{2e\lambda}{hc\eta} \qquad (6.6)$$

$$I_S = \frac{\beta}{B\tau_n\tau_p} \qquad (6.7)$$

$$I_{th} = \frac{eVN_{th}}{\tau_n} \qquad (6.8)$$

são os novos parâmetros; *P(t)* é a potência óptica, *X(t)* é a densidade de portadores relativa ($N(t)/N_{th}$), $\phi(t)$ é a fase, g_0 é o coeficiente de ganho e N_{th} é a densidade de portadores no limiar.

O sistema real montado, e, portanto, também o sistema simulado, é composto por um gerador de corrente com forma de onda senoidal que alimenta um laser DFB. O sinal óptico emitido pelo laser é fotodetectado e então analisado por um osciloscópio. O fotodetector e o osciloscópio introduzem uma limitação na largura de banda, o que é representado por filtros na simulação.

Transportando esse esquema para o LightSim, tem-se a seguinte topologia:

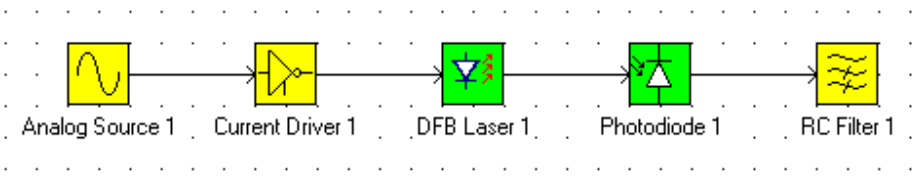

Figura 6.7 – Topologia para simulação do laser DFB alimentado com corrente cuja forma de onda é senoidal.

Para a simulação no LightSim, o laser DFB foi implementado usando-se o novo modelo proposto (equações 6.1 a 6.3).

Foram usados três lasers diferentes, e, portanto, três conjuntos de parâmetros diferentes foram estimados. Esses parâmetros são mostrados na tabela Tabela 6.4.

Tabela 6.4 – conjuntos de parâmetros estimados para os três lasers [10]

Parâmetro	Laser A	Laser B	Laser C
λ (nm)	1545,2	1555,4	1547,1
I_{th} (mA)	17,70	11,09	19,02
I_S (uA)	0,51	0,11	0,13
F (A/W)	23,4	15,4	29,4
B (GHz^2/mA)	116,5	149,2	102,4
K (ps)	418	487	597
τ_n (ns)	0,30	0,37	0,21
α	5,2	3,2	2,8
τ_c (ps)	3,9	5,6	4,9
τ_p (ps)	6,7	6,7	10,2

Nas figuras Figura 6.8, Figura 6.9 e Figura 6.10, a parte "(a)" mostra o resultado de [10], onde a linha tracejada é o resultado de simulação e a linha contínua é a medida experimental. Já a parte "(b)" mostra o resultado da simulação no LightSim. As comparações com os resultados do LighSim devem ser feitas levando-se em consideração a simulação de [10] (linha tracejada). Para facilitar a notação, I_0 é a corrente de *off-set* da forma de onda senoidal, I_m é a amplitude da onda senoidal, e f é a freqüência usada no gerador de corrente.

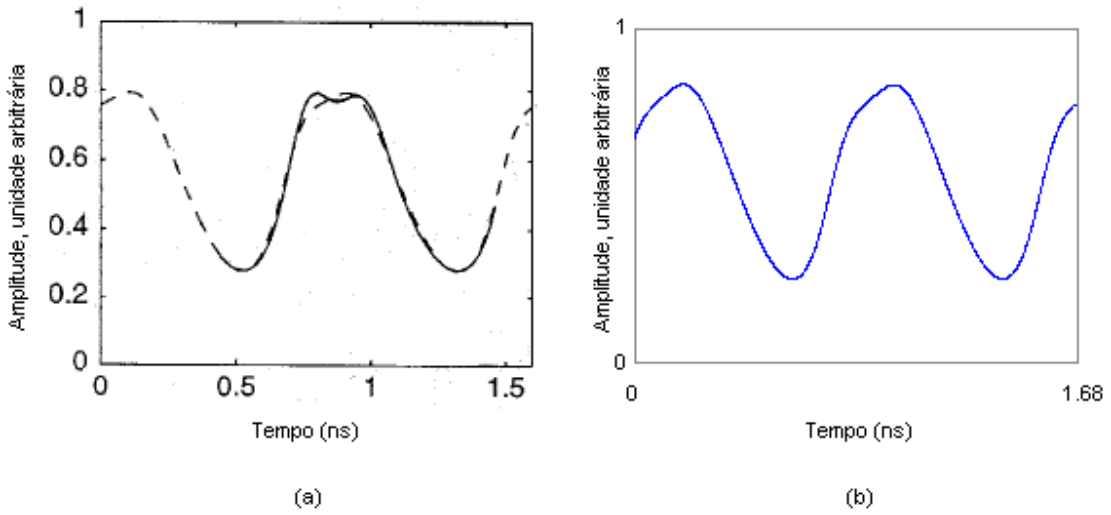

Figura 6.8 – Comparação entre simulações e medida do laser A, com I_0 = 22,0 mA, I_m = 2,0 mA e f = 1,25 GHz. a) resultados de [10]; linha tracejada: simulação; linha contínua: medida. b) resultado da simulação no LightSim.

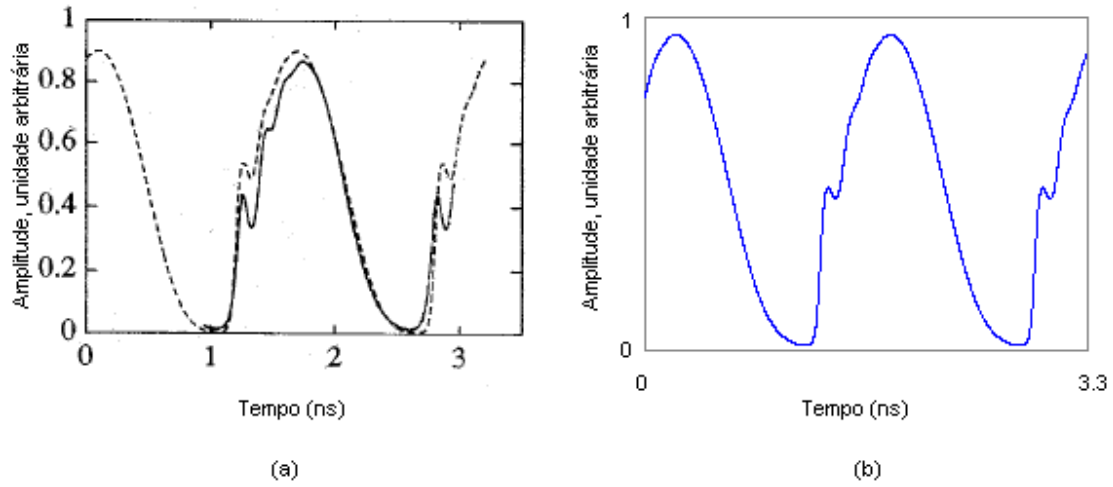

Figura 6.9 - Comparação entre simulações e medida do laser B, com I_0 = 18,1 mA, I_m = 7,2 mA e f = 625 MHz. a) resultados de [10]; linha tracejada: simulação; linha contínua: medida. b) resultado da simulação no LightSim.

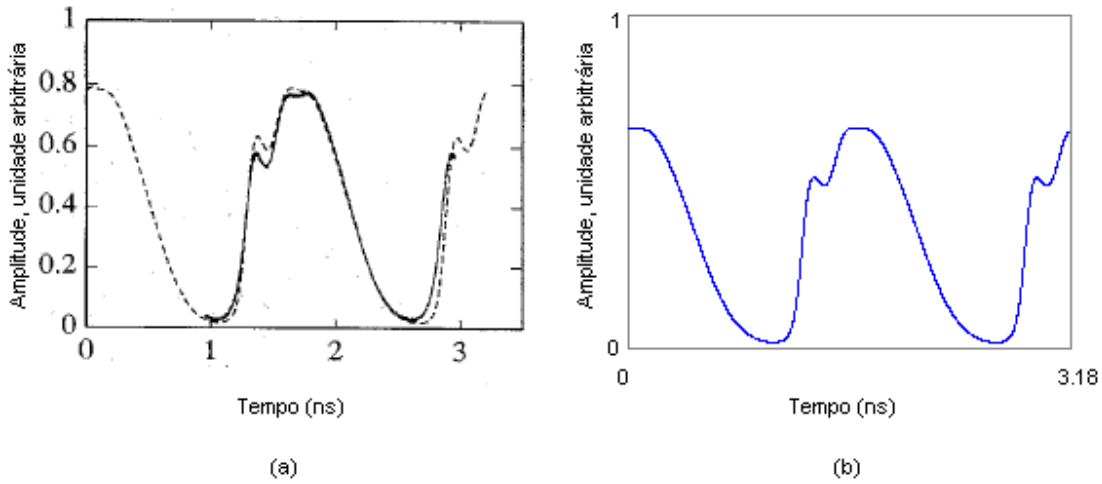

Figura 6.10 - Comparação entre simulações e medida do laser C, com I_0 = 23,0 mA, I_m = 4,0 mA e f = 625 MHz. a) resultados de [10]; linha tracejada: simulação; linha contínua: medida. b) resultado da simulação no LightSim.

Em cada um dos três casos, nota-se que os resultados das simulações são parecidos. Isso traz uma maior confiabilidade à implementação do método de simulação no LightSim e ao método de simulação em si.

É importante ressaltar que o objetivo dessas comparações não é verificar o correto funcionamento do modelo de laser DFB implementado ou do método numérico para resolver as equações de taxa. O objetivo é verificar o correto funcionamento da técnica de simulação implementado no LightSim.

6.3 Simulação de um sistema com realimentação

Um dos requisitos estabelecidos para o simulador desenvolvido é a capacidade de simular sistemas com realimentação. Conforme visto no item 4.5.3, a técnica de simulação a eventos discretos adaptada permite fazer essa simulação.

Para verificar essa capacidade, é simulado um sistema realimentado, mostrado na figura Figura 6.11:

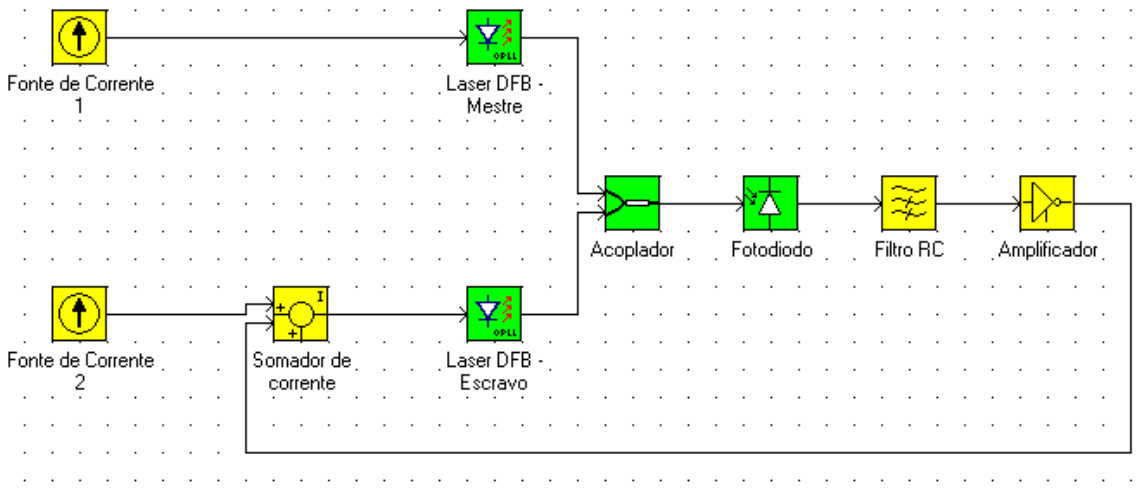

Figura 6.11 – Topologia de um OPLL homódino simples

Essa é a topologia de um OPLL (*Optical Phase-Lock Loop*) homódino simples. Sua finalidade é fazer com que a freqüência da luz emitida pelo laser escravo (laser DFB - Escravo) se iguale à freqüência de um sinal de luz de referência, proveniente nesse sistema pelo laser mestre (laser DFB – Mestre).

A alteração da freqüência do laser DFB é possível alterando-se o valor de sua corrente de alimentação. Uma variação da corrente de alimentação provoca não apenas uma variação no número de fótons em sua saída (intensidade de luz), mas também no número de portadores na região ativa do laser. A variação no número de portadores faz variar o índice de refração do meio, que, por sua vez, faz variar a fase da luz, e, conseqüentemente sua freqüência [3]. Essa relação entre número de fótons, número de portadores e fase é expressa pelas equações de taxa do laser DFB. De maneira geral, uma corrente maior causa uma freqüência maior. (Veja figura Figura 6.6.)

Na topologia da Figura 6.11, tanto o laser escravo como o laser mestre são alimentados com uma corrente de polarização DC de 40mA, mas têm freqüências de operação diferentes entre si. O sinal de luz do laser mestre e o sinal de luz do laser escravo são acoplados pelo acoplador óptico, gerando um sinal de batimento com diversos componentes em freqüência. Esse batimento é detectado pelo fotodiodo que transforma o sinal óptico em elétrico. Para essa simulação, o modelo implementado para o fotodiodo elimina componentes de freqüência indesejados. A corrente proveniente do fotodetector

passa por um filtro passa-baixa, para eliminar variações abruptas, e por um amplificador, para ajustar a corrente resultante a um nível desejado.

Essa corrente final é a corrente de controle do laser escravo, que, somada com a corrente de polarização, constitui a corrente de alimentação. Se a freqüência do laser escravo for menor que a do mestre, a corrente de controle deve ter um nível médio positivo para poder elevar a freqüência, e vice-e-versa.

As figuras Figura 6.12 e Figura 6.13 mostram os resultados da simulação considerando duas separações de freqüências entre laser mestre e escravo. Em ambos os casos, a freqüência de operação do laser mestre é de 230,609583 THz (comprimento de onda = 1,3μm), mas com uma corrente de polarização constante de 40mA a freqüência resultante é de 230,618645 THz (1,2999522μm).

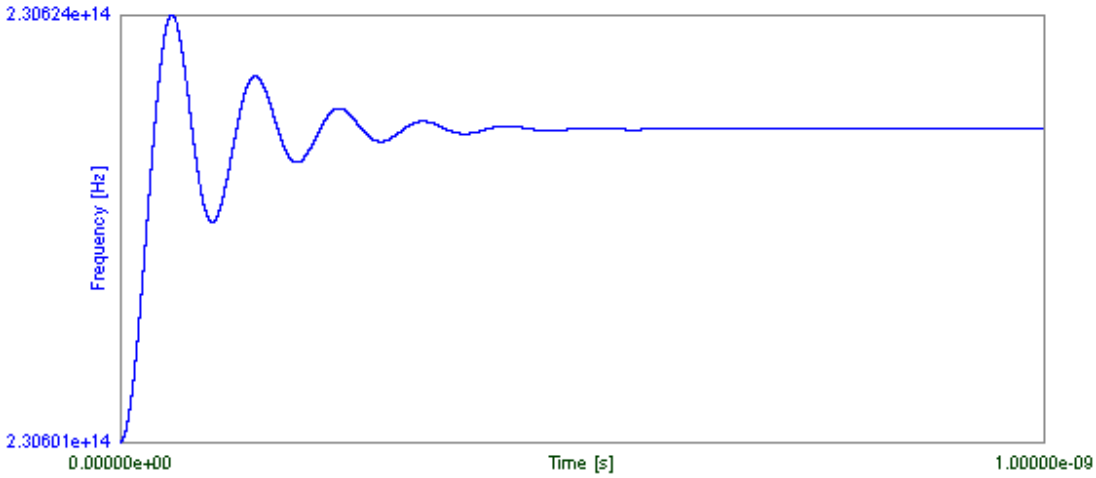

Figura 6.12 – Freqüência do laser escravo, inicialmente em 230,600907 THz (1,300048μm)

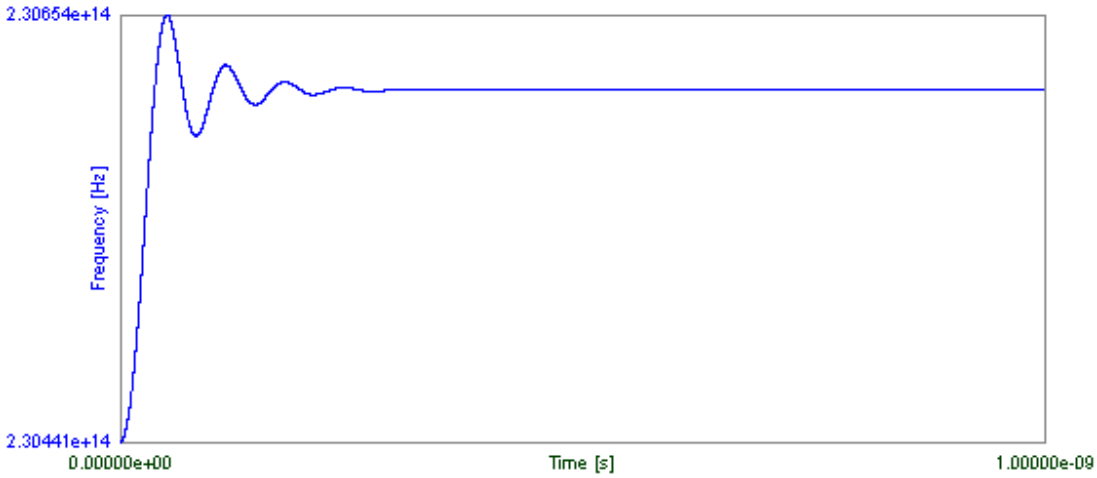

Figura 6.13 – Freqüência do laser escravo inicialmente em 230,441390 THz (1,300949μm)

Na figura Figura 6.12, o laser escravo está inicialmente em 230,600907 THz (1,300048μm) e estabilizou em 230,618058 THz (1,299952μm), enquanto que na figura Figura 6.13, o laser escravo está inicialmente em 230,441390 THz (1,300949μm) e estabilizou em 230,616387 THz (1,299962μm).

Nota-se que, em ambos os casos, a freqüência do laser escravo saiu de seu valor inicial e estabilizou no valor da freqüência do laser mestre, após algumas oscilações com amortecimento. Este é um sistema de controle em malha fechada e a amplitude das oscilações é controlada pelo ganho aplicado, e o amortecimento é controlado pelo filtro passa-baixa.

6.4 Conclusão

O objetivo das simulações anteriores é comprovar o correto funcionamento da simulação a eventos discretos implementado no simulador. Por isso, detalhes de modelagem matemática e análises quantitativas não são abordados.

Em ambos os casos, os resultados mostram concordância com o que é observado na prática. No caso do laser DFB, onde foi feita uma comparação com resultados validados em outro trabalho, essa concordância é mais explícita.

A simulação do OPLL mostrou que o LightSim é capaz de simular topologias que apresentam realimentação, o que é um dos objetivos desse trabalho.

Com esses resultados, pode-se assegurar que o método e o simulador são capazes de simular sistemas de comunicação óptica cuja topologia apresenta, ou não, realimentação.

7 Conclusão

O método de simulação a eventos discretos se mostra adequado à simulação de sistemas de comunicação óptica no nível físico. Considerando o ambiente PC monoprocessado, apresenta vantagens significativas em relação a outros métodos analisados.

Porém, a maneira tradicional de abordagem da simulação a eventos discretos considera apenas um único processo, o que não é muito adequado para simulação de sistemas de comunicações ópticas, onde o sistema modelado é composto por vários processos interligados. Mas adotando a orientação a processos e mensagens na simulação a eventos discretos, chega-se a uma solução que possibilita a simulação de sistemas com vários componentes, onde cada componente pode ter modelagens matemáticas tão complexas como o laser semicondutor DFB.

A fim de evitar erros de causalidade, é necessário adotar uma fila de mensagens global e única para todo o sistema simulado. Ao ordenar as mensagens na fila pelo tempo associado a cada uma delas, garante-se que a mensagem a ser processada, que é a primeira da fila, sempre será a de menor tempo associado.

A implementação da simulação a eventos discretos com orientação a processos possibilitou não somente a simulação de topologias ponto-a-ponto (enlaces ópticos), mas também topologias que apresentam realimentação, como é o caso do OPLL óptico.

O software de simulação implementado, LightSim, foi desenvolvido de tal maneira a facilitar a inclusão de novos modelos, a manutenção e expansão do software. Para isso, o software foi projetado de forma modular. Há três módulos gerais no LightSim: a interface gráfica, o *kernel*, e os componentes. Na interface gráfica há um intenso uso de recursos gráficos, a fim de facilitar o entendimento da ferramenta pelo usuário. Esse módulo é totalmente dependente do sistema operacional já que usa recursos gráficos.

Já o módulo *kernel* é independente de recursos do sistema operacional. É nesse módulo que se encontra o motor (*engine*) de simulação. O *kernel* recebe da interface os comandos, parâmetros e topologia montada pelo usuário. Na execução da simulação, cada modelo presente na topologia montada pelo usuário é invocado pelo *kernel* quando há uma mensagem destinada a ele.

Os modelos são DLLs, e, portanto, ao serem construídos e adicionados ao LightSim, não precisam que todo o software seja recompilado. Assim o desenvolvedor de modelos não precisa ter conhecimento de todo o código do LightSim, mas apenas de algumas funções padronizadas para o modelo.

Os resultados de simulação para o laser DFB mostram uma coerência com os resultados esperados teoricamente. Ao se modular diretamente um laser DFB com um trem de pulsos de corrente, a simulação mostrou que na saída óptica do laser DFB havia um trem de pulsos ópticos, com as oscilações de relaxação, o que era esperado. E ao se simular um modelo de laser DFB usado em outro trabalho, cujos resultados são validados, nota-se a similaridade entre esses e os resultados do LightSim.

Já na simulação do OPLL simples homódino, foi possível observar o deslocamento da freqüência do laser escravo, estabilizando aproximadamente na freqüência do laser mestre. A topologia do OPLL apresenta realimentação, o que serviu para comprovar o correto funcionamento da técnica de simulação para este caso.

Nas simulações mostradas, não houve uma constante preocupação com os resultados quantitativos. A intenção nos dois casos é de comprovar que o método de simulação a eventos discretos com orientação a processos é capaz de simular sistemas de comunicação óptica na camada física; e também que a implementação do simulador está correta.

Com os resultados apresentados, todos os objetivos traçados no início do trabalho foram alcançados: foi desenvolvido um simulador de sistemas de comunicação óptica na camada física, foi mostrado o uso da técnica de simulação a eventos discretos (com orientação a processos) implementado no simulador, e foram eliminadas as limitações do trabalho anterior.

8 Referências

[1] Matiuso, J. A., "*Proposta de um ambiente de simulação no domínio de fluxo de dados contínuos*", Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Agosto de 2002.

[2] Rossi, S. M., "*PC-LASER:Um Software para Simulação de Lasers Semicondutores*", Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Janeiro de 1994.

[3] Gonçalves, Marcos S., "*Análise da Evolução Temporal do Processo de Aquisição de Laços de Travamento de Fase Óptica Homódinos*", Tese de mestrado, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Julho de 2002.

[4] Schriber, T.J. e Brunner, D.T., "Inside Discrete-Event Simulation Software: How It Works and Why It Matters", *Proceedings of the 2004 Winter Simulation Conference*, pp. 142-152.

[5] Cassandras, C. G., "*Discrete Event Systems: Modeling and Performance Analysis*", Richard D. Irwin and Aksen Associates Inc. Publishers, Boston, MA, 1993.

[6] Lee, E. A. and Messerschmitt, D. G., "Synchronous Data Flow", *Proceedings of the IEEE*, pp. 1235-1245 , **Vol 75**, No. 9, September 1987.

[7] Fujimoto, R. M., "Parallel Discrete Event Simulation*", Proceedings of the 1989 Winter Simulation Conference*, pp. 19-28.

[8] Ptolemy II – "*Heterogeneous Concurrent Modeling and Design in Java - Volume 3: Ptolemy II Domains*", Department of Electrical Engineering and Computer Science, University of California at Berkeley, June 2004.

[9] Agrawal, G. P., "*Fiber-Optic Communication Systems*", John Wiley & Sons, Inc., New York, 2002.

[10] Bjerkan, L.; Royset, A.; Hafskjoer, L.; Myhre, D., "Measurement of Laser Parameters for Simulation of High-Speed Fiberoptic Systems", *Journal of Lightwave Technology*, pp. 839-850, **Vol 14**, No. 8, May 1996.