UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Gustavo Cunha Guedes

TÍTULO DO TRABALHO

HERMES: Um *Software-modem* ADSL Baseado no *Framework* Ptolemy II

DM_17/2011

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém -Pará - Brasil
2011

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Gustavo Cunha Guedes

TÍTULO DO TRABALHO

HERMES: Um *Software-modem* ADSL Baseado no *Framework* Ptolemy II

Dissertação submetida à Banca Examinadora do Programa de Pós-graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Mestre em Engenharia Elétrica, ênfase em Telecomunicações.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

HERMES: UM *SOFTWARE-MODEM* ADSL BASEADO NO *FRAMEWORK* PTOLEMY II

G924h Guedes, Gustavo Cunha

HERMES: Um *Software-modem* ADSL Baseado no *Framework* Ptolemy II;
Gustavo Cunha Guedes; orientador, Aldebaro Barreto da Rocha Klautau Júnior. 2011.

Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia,
Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2011.

1. Software - desenvolvimento. 2. Modem. 3. Linhas digitais de assinantes.
4. Simulação (computadores). I. Orientador. II. Título.

CDD 22. ed. 005.1

UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

HERMES: UM *SOFTWARE-MODEM* ADSL BASEADO NO *FRAMEWORK* PTOLEMY II

AUTOR: GUSTAVO CUNHA GUEDES

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM 20/04/2011

BANCA EXAMINADORA:

.....

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior

ORIENTADOR - PPGEE/ITEC/UFPA

.....

Prof. Dr. Evaldo Gonçalves Pelaes

MEMBRO - PPGEE/ITEC/UFPA

.....

Prof. Dr. Claudomiro Souza Sales Junior

MEMBRO EXTERNO - ICEN/UFPA

VISTO:

.....

Prof. Dr. Marcus Vinícius Alves Nunes

COORDENADOR DO PPGEE/ITEC/UFPA

Dedicado ao meu amado pai,

Cleobino Alberto Guedes Junior,

com o amor e a saudade mais imensos.

Agradecimentos

Aos meus pais, Cleobino e Fátima, pelo carinho, amizade, apoio, disciplina, incentivo, exemplos e tantas outras atitudes que fizeram toda a diferença. Devo tudo a vocês. Pai, você pode não estar mais presente fisicamente, mas do meu coração e da minha memória nunca irá sair. Mãe, você é a pessoa mais incrível do mundo. Amo-os com todas as minhas forças.
Aos meus irmãos, Paulo e Rafael, por toda a nossa vida juntos, da infância mais tenra à mais instável adolescência; pelas nossas brincadeiras, aventuras e até mesmo as brigas; por serem meus exemplos de vida. Amo vocês.
À minha querida Cinthia, pelo companheirismo, apoio, amizade, afeto e por estar ao meu lado nos momentos em que mais precisei. Te amo.
À minha querida cunhada e irmã postiça, Cibele, por todo o carinho que só uma irmã poderia dar, e aos meus sogros Cláudia e Cláudio, por me receberem sempre de braços abertos no aconchego do seu lar.
À Malu, minha inseparável vira-latas; à Nina, minha doce siamesa; à Meg, minha moleca; à Maroca, minha tagarela; à Nikitinha e à Mium, meus anjos que se foram cedo demais; que se apropriaram de meu coração mole como verdadeiras filhas e que sempre conseguiram me encher de alegria com um simples olhar ou uma lambida.
Aos amigos da Faculdade de Engenharia de Computação e do LaPS/LEA/LPRAD: Daniella, Leide, Lilian, Adriana, Marcel, Elton, Kelly, Yomara, Claudia, Adalbery, Johelden, Fernanda, Muller, Carol, Nelson, José, Patrick, Igor, Eduardo, Marcio, Harney, Tales, Erick, Luciana, Silvia, Pedro, Bruno, Liviane, Marcelino, Diego, Edvar, João, Marco, Gilvan, Renan, Diogo, Diego Gomes, Roberto, Patricio, Ilan, Zampollo, Valquiria. Se esqueci de alguns, peço que não se sintam excluídos por conta de minha memória falha.
Ao meu orientador e grande amigo, Aldebaro, pelas oportunidades, orientação, suporte e, principalmente, pelos conselhos e constante disposição em ajudar. Aos membros da banca (e também bons amigos) que me avaliou: Professor Pelaes e Professor Claudomiro.
Enfim, agradeço a todas as pessoas que, direta ou indiretamente, estiveram envolvidas em minha formação, não apenas profissional e acadêmica, mas também de caráter e de humanidade.

Resumo

Esta dissertação tem como objetivo principal apresentar a implementação de um *software*-modem ADSL totalmente escrito em Java, utilizando o *framework* Ptolemy II, denominado de Hermes (de "*A* ***H****andy* ***E****xperimental softwa****R****e* ***M****od****E****m* ***S****ystem*"). Um *software*-modem é útil em ocasiões onde se precisem executar testes e simulações de sistemas de comunicação com um número grande de modens e quando os parâmetros desses sistemas precisem ser modificados com um alto grau de liberdade. Além disso, um *software*-modem possui características que tornam mais fácil a tarefa de acrescentar, eliminar, validar e analisar funções e algoritmos de processamento de sinais e de telecomunicações. Testes e simulações foram realizados para analisar a funcionalidade do Hermes, utilizando, inclusive, o Tracespan, um equipamento para análise não-intrusiva de redes DSL. A partir dos resultados obtidos em conjunto com o Tracespan, foi possível validar com sucesso as funções do Hermes.

PALAVRAS-CHAVES: software-modem, simulação, ptolemy, adsl, dmt.

Abstract

This dissertation has as main objective to present the implementation of an ADSL software-modem totally written in Java, using the Ptolemy II framework, named Hermes (from "A **H**andy **E**xperimental softwa**R**e **M**od**E**m **S**ystem"). A software-modem is useful in cases where one needs to perform tests and simulations of communication systems with a large number of modems and when systems' parameters must be changed with high degree of freedom. Besides, a software-modem has features that make it easy to add, eliminate, validate and evaluate signal processing and telecommunications functions and algorithms. Tests and simulations were performed to evaluate the Hermes functionality, including the use of Tracespan, a non-intrusive analisys equipment for DSL networks. From obtained results together with Tracespan, it was possible to successfully validate the Hermes functions.

KEYWORDS: software-modem, simulation, ptolemy, adsl, dmt.

Lista de Figuras

Lista de Tabelas

Glossário

ADSL	-	*Asymetric Digital Subscriber Line*
AFE	-	*Analog Front End*
AOC	-	*ADSL Overhead Control*
ATU-C	-	*ADSL Transceiver Unit - Central*
ATU-R	-	*ADSL Transceiver Unit - Remote*
AWGN	-	*Additive White Gaussian Noise*
CRC	-	*Cyclic Redundancy Check*
DMT	-	*Discrete Multi-Tone*
DSM	-	*Dynamic Spectrum Management*
DSLAM	-	*Digital Subscriber Line Access Multiplexer*
EOC	-	*Embedded Operations Channel*
FEC	-	*Forward Error Correction*
FEQ	-	*Frequency-domain Equalizer*
FFT	-	*Fast Fourier Transform*
FPGA	-	*Field-programmable Gate Array*
IEEE	-	*Institute of Electrical and Electronics Engineers*
ISI	-	*Intersymbol Interference*
ITU	-	*International Telecommunication Union*
ITU-T	-	*ITU Telecommunication Standardization Sector*
LMS	-	*Least Mean Square*
OAM	-	*Operation, Administration and Maintenance*
QAM	-	*Quadrature Amplitude Modulation*

SDF - *Synchronous Dataflow*

SNR - *Signal to Noise Ratio*

TEQ - *Time-domain Equalizer*

VDSL - *Very-high-bit-rate Digital Subscriber Line*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very High Speed Integrated Circuits*

Sumário

Capítulo 1

Introdução

A palavra modem é uma abreviação para "modulador e demodulador". Um *software*-modem, por sua vez, é um modem totalmente (ou quase totalmente) implementado em *software*. Ou seja, suas funções não são executadas por circuitos eletrônicos (que são chamados de *hardware*), mas sim por um programa de computador.

ADSL [3, 4, 5] é a sigla para *Asymetric Digital Subscriber Line*, uma variação dos sistemas DSL [6, 7] para provimento de internet em banda larga para usuários que possuam uma linha telefônica . Para isso, um modem ADSL é instalado na casa do usuário, enquanto outro modem ADSL (normalmente embutido em um conjunto de vários modens, 12 por exemplo, chamado de DSLAM - de *Digital Subscriber Line Access Multiplexer*) é instalado na operadora. O modem do usuário é referido como ATU-R (*ADSL Transceiver Unit - Remote*), enquanto o modem da operadora é referido como ATU-C (*ADSL Transceiver Unit - Central*).

1.1 Motivações

Um modem ADSL, seja ele ATU-R ou ATU-C, normalmente possui grande complexidade, necessitando desempenhar funções matemáticas, de controle, de processamento intensivo de sinais, máquinas de estados, dentre outras. Uma vantagem imediata da implementação de um modem ADSL em *hardware* [8] é que essas funções podem ser executadas em alta velocidade.

Entretanto, surge também uma desvantagem importante: a falta de flexibilidade, já que o *hardware* será otimizado para os parâmetros específicos do modem. Além disso, adicionar novas funções e capacidades torna-se um processo difícil e caro. A Tabela 1.1 ilustra uma

comparação entre algumas características de um *hardware*- e as de um *software*-modem.

Tabela 1.1: Características de um *hardware*-modem versus as de um *software*-modem.

Característica	***Hardware*-modem**	***Software*-modem**
Desempenho	melhor	pior
Adição de capacidades e funções	mais difícil	mais fácil
Modificação de parâmetros	mais difícil	mais fácil
Custo	maior	menor
Pontos de teste	fixos	adicionados conforme a necessidade
Prototipagem, testes e simulações	mais difícil	mais fácil

Atualmente existe uma demanda por taxas de dados cada vez maiores por conta dos usuários de banda larga, incluindo os usuários de sistemas DSL. Essas taxas ultrapassam os limites impostos pelas características das linhas (tipo de cabo, comprimento, etc) e pelas interferências causadas por ruídos externos (por exemplo, estações de rádio AM) e por outras linhas DSL contidas no mesmo cabo (interferência essa chamada de "fala cruzada' (ou *crosstalk*, em inglês) [9, 10, 11]).

Duas linhas de pesquisa têm sido desenvolvidas no sentido de se combater essas interferências e, consequentemente, superar esses limites.

Uma delas é a que estuda o ajuste (ou balanceamento) das potências por tom alocadas para as linhas presentes num *cabo*, de forma a otimizar os parâmetros de taxa, potência, margem ou outros, de todos os modens ou de um grupo deles. Essa linha de pesquisa é chamada de *Dynamic Spectrum Management* (DSM) [12].

A outra linha de pesquisa, *Vectoring*, trata da técnica de vetorização dos sinais transmitidos em um grupo de conexões DSL cujas linhas possuam *crosstalk* entre si. Dessa forma, os sinais enviados e recebidos pelo grupo de modens são agrupados em vetores e os canais, tanto os principais quanto os de *crostalk*, são vistos como uma matriz. Através da estimação precisa dos canais de *crosstalk* e de um sistema de sincronismo entre todas as transmissões do grupo, é possível cancelar (ou diminuir significativamente) os efeitos desse *crosstalk* através do uso de equalizadores nos receptores dos modens, possibilitando o alcance de taxas muito mais altas do que seria possível sem esse cancelamento [13].

Existe uma forte tendência de se utilizar ambas as técnicas em conjunto para se obter

resultados ainda melhores [14].

Tanto DSM quanto *Vectoring* são técnicas relativamente recentes e sua implementação prática ainda é muito difícil, principalmente pela ausência de uma plataforma de testes e simulações que auxilie na implementação de novos algoritmos, ajuste de parâmetros e análise de resultados de forma prática e simples.

1.2 Simulação de sistemas DSL

Alguns ambientes para simulação de sistemas de comunicação do tipo DSL podem ser citados aqui.

O FTW xDSL *simulation tool*, ou simplesmente FTW [15], é um *toolbox* escrito em Matlab para simulação de sistemas DSL, inclusive com a disponibilização de uma interface gráfica. O FTW se encontra atualmente na versão 3.1.

O projeto Ericsson/UFPA implementou o jDSLsim, um *software* para simulação de sistemas DSL totalmente escrito em Java, com suporte para cenários com várias linhas e considerando a existência de *crosstalk* entre elas. O *software* basicamente realiza o cálculo das taxas possíveis de serem alcançadas em cada linha em diferentes configurações de cenários.

Tranter [16] apresenta técnicas eficientes de simulação de sistemas de comunicação em geral, com algoritmos implementados em Matlab, porém com ênfase em aplicações *wireless*.

Evans [17] desenvolveu um *toolbox* em Matlab para simulação de canais e equalizadores no domínio do tempo (TEQ) para sistemas com modulação *Discrete Multi Tone* (DMT), que é o tipo de modulação adotado pelos sistemas ADSL.

1.3 Implementações de *software*-modens

Bellard [18] desenvolveu um projeto para Linux denominado Linmodem, com a pretensão de torná-lo uma alternativa de *driver* Linux para os modens do tipo “WinModem”, muito utilizados na década de 90. Tais modens eram basicamente AFE’s para *software*-modens escritos exclusivamente para Windows e, portanto, não podiam funcionar em sistemas Linux.

Porém, o projeto foi desenvolvido até a versão 0.2.5 e sua página foi atualizada pela última vez em 6 de março de 2000. Não foi colocado em prática (isto é, não pode ser utilizado como um *driver* para WinModens), funcionando apenas como uma ferramenta para pesquisa e desenvolvimento de algoritmos de telecomunicações.

1.4 O que é o Hermes?

Este trabalho apresenta um *software*-modem ADSL denominado de Hermes (*A* ***H****andy* ***E****xperimental softwa****R****e* ***M****od****E****m* ***S****ystem*). Tendo sido implementado através do *framework* Ptolemy II [19], o Hermes possui como principais características a flexibilidade de adição de funções e capacidades, ajustes de parâmetros, inclusão de pontos de teste (para visualização de sinais, variáveis, etc) e depuração.

Todos os blocos do Hermes foram desenvolvidos em conjunto com o grupo ERICSSON/UFPA, do qual o autor desta dissertação faz parte. Durante a sua implementação, a publicação [20] foi gerada.

Seu uso pode ser aplicado a situações em que seja necessário projetar, simular e testar sistemas ADSL, inclusive com um número grande de instâncias (ou seja, com vários modens individuais), modificando-se com frequência seus parâmetros.

Através do uso de um AFE (*Analog Front End*), um circuito eletrônico que faz a conversão digital/analógica dos sinais transmitidos pelo Hermes e a conversão analógica/digital dos sinais recebidos [21], é possível também utilizar o Hermes com linhas telefônicas reais, tornando possível analisar sua performance com maior realismo.

1.5 Estrutura do trabalho

Este trabalho está estruturado da seguinte maneira.

O Capítulo 2 explica as etapas de uma transmissão ADSL e apresenta o *framework* Ptolemy II , sobre o qual foi desenvolvido o Hermes.

O Capítulo 3 aborda o processo de projeto e implementação de cada bloco do Hermes,

dividindo-o em Transmissor, Receptor e Controle.

O Capítulo 4 demonstra algumas simulações feitas com o Hermes, incluindo simulações *offline* com auxílio do equipamento Tracespan^{TM}[2], e apresenta seus resultados.

Finalmente, no Capítulo 5 são apresentadas algumas considerações finais e possibilidades de trabalhos futuros.

Capítulo 2

O *Software-modem* Hermes

O **Hermes** é um *software-modem* ADSL implementado através do *framework* Ptolemy II. Uma visão geral do sistema no qual o Hermes é aplicado é ilustrada na Figura 2.1.

Figura 2.1: Visão geral do funcionamento do Hermes.

Como a Figura 2.1 mostra, o Hermes é executado pelo computador e pode se comunicar com outro modem (seja ele um modem real ou uma outra instância do Hermes) através da instalação, no computador, de um *Analog Front End* (AFE), um circuito eletrônico que faz a conversão digital/analógica dos sinais transmitidos pelo Hermes e a conversão analógica/digital dos sinais recebidos [21].

Como será mostrado no capítulo 4, é possível também executar o Hermes sem a necessidade de um canal real e dos AFE's, utilizando-se para isso um modelo de canal simulado no próprio Ptolemy.

Para entender como o Hermes foi implementado, é necessário primeiro saber como acontece

uma transmissão ADSL e em que consiste o *framework* Ptolemy II. Este capítulo traz uma breve explicação sobre as etapas de uma conexão ADSL e apresenta, em seguida, o Ptolemy II.

2.1 As etapas de uma transmissão ADSL

De acordo com [1], uma conexão ADSL atravessa basicamente duas etapas:

1. Inicialização - Quando o modem é ligado e executa algumas rotinas padrão para iniciar a transmissão/recepção;
2. *Showtime*[1] - Quando o modem entra no estado de transmissão e recepção propriamente ditas.

A etapa de inicialização é, por sua vez, dividida em 4 sub-etapas:

1. Sinalização (*Handshake*)
2. Treinamento (*Training*)
3. Análise de Canal (*Channel Analysis*)
4. Troca de informações (*Exchange*)

A figura 2.2 ilustra essas etapas, sendo elas descritas nas seções seguintes.

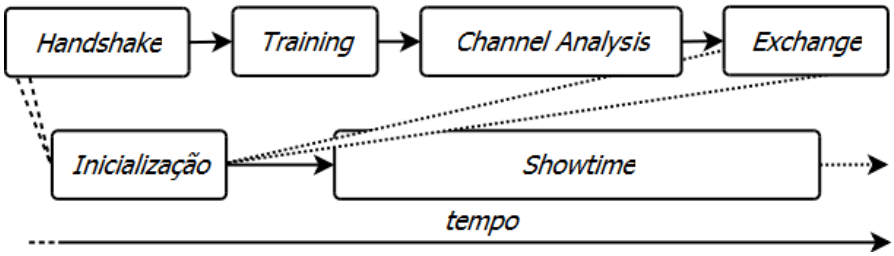


Figura 2.2: Ilustração das etapas de inicialização e *showtime* de uma transmissão ADSL.

[1]Não há uma tradução apropriada deste termo para o português. Detalhes sobre seu significado são fornecidos na Seção 2.1.5.

2.1.1 Sinalização

A etapa de sinalização (*handshake*) normalmente é iniciada pelo modem do usuário (ATU-R) assim que este é ligado. Entretanto, em alguns casos ela pode ser iniciada pelo modem da operadora (ATU-C). Essa etapa e seus protocolos são definidos em [22] e sua implementação não foi finalizada neste trabalho. Por este motivo, ela não será detalhada aqui. O leitor que desejar uma descrição dos procedimentos de *handshake* definidos nessa recomendação encontrará uma boa fonte de informações em [23].

2.1.2 Treinamento

Durante a etapa de treinamento (*training*), os equalizadores e sincronizadores dos receptores (tanto o do modem da operadora quanto o do usuário) estimam seus parâmetros de forma que possam, durante o andamento da transmissão, corrigir características indesejáveis do canal como rotação e fase por tom, resposta ao impulso maior que o prefixo cíclico, *offsets* de frequência e fase de clock (que não são características do canal), etc, de tal forma que a recepção do sinal seja otimizada.

Essa estimação de parâmetros é feita através da transmissão de uma sequência de treino por um dos modems e da comparação, por parte do outro modem, entre o que é recebido e o que se esperava receber.

A Figura 2.3 ilustra a linha do tempo da etapa de treinamento, simultaneamente para os modens ATU-C e ATU-R.

O nome de cada sub-fase na Figura 2.3 corresponde ao nome do sinal que está sendo enviado pelo modem durante aquela sub-fase. Por exemplo, na sub-fase C-REVERB1, o sinal C-REVERB1 está sendo enviado pelo modem ATU-C. Além disso, nesse mesmo instante o modem ATU-R está enviando o sinal R-REVERB1.

Os números abaixo dos sinais indicam a Seção em [1] que define esses sinais. Por exemplo, o sinal C-QUIET5 é definido na seção 10.4.9 de [1].

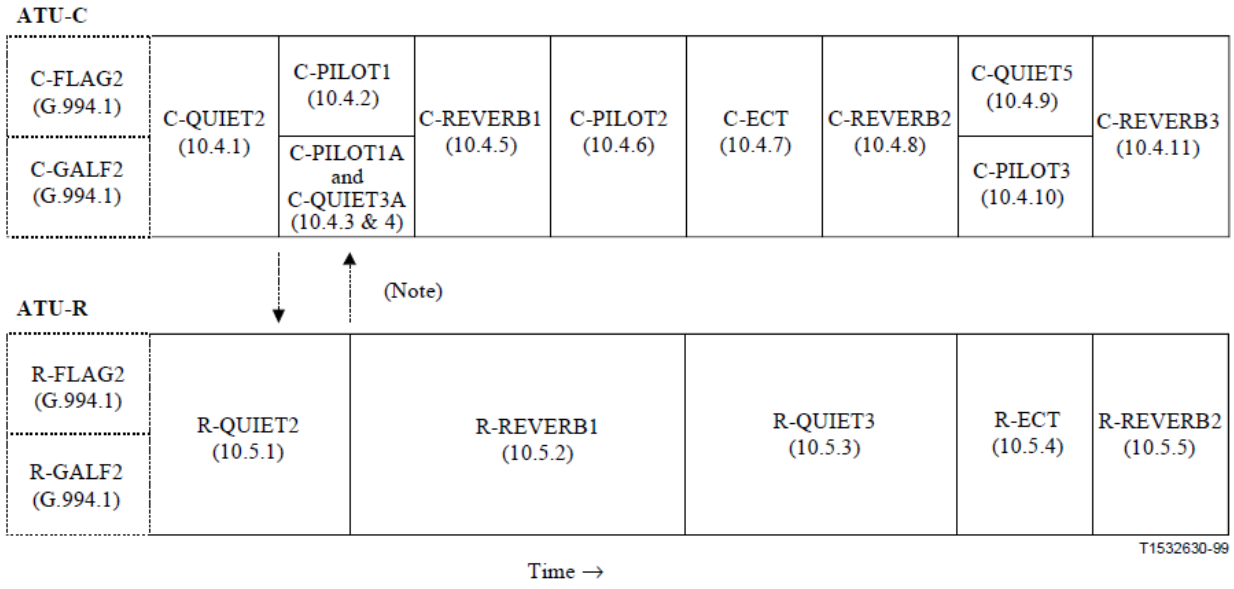


Figura 2.3: Linha do tempo da etapa de treinamento da inicialização (retirada de [1]).

2.1.3 Análise de canal

A etapa de análise de canal (*channel analysis*) se caracteriza pelo envio e recepção, por ambos ATU-C e ATU-R, de mensagens relativas a taxas possíveis/desejáveis, além da estimação da relação sinal/ruído (SNR) do canal.

A Figura 2.4 mostra a linha do tempo da etapa de *channel analysis*.

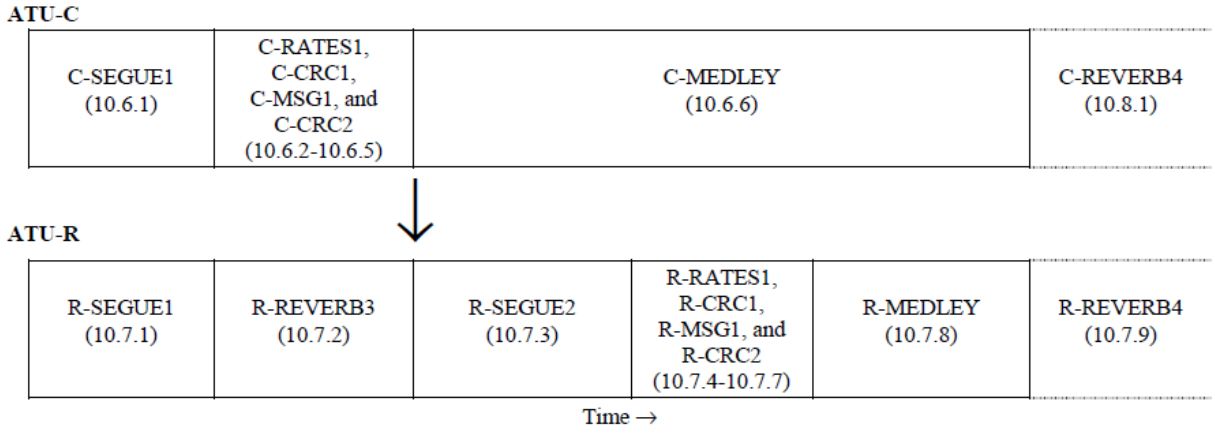


Figura 2.4: Linha do tempo da etapa de análise de canal da inicialização (retirada de [1]).

2.1.4 Troca de informações

Durante a etapa de troca de informações (*exchange*), o ATU-C e o ATU-R, através de trocas de mensagens, negociam parâmetros como as taxas com que vão transmitir, número de bits alocados para cada tom, potência de cada tom, número de bytes de paridade, valor de margem escolhido, etc.

A Figura 2.5 mostra a linha do tempo da etapa de *exchange*.

2.1.5 *Showtime*

Finalmente, na etapa de *showtime* os modens transmitem dados entre si de forma estável e com poucos erros[2], já fixados os parâmetros relativos a taxa de bits, tamanho do frame, profundidade (*depth*) do *interleaver*, etc.

Durante esta etapa, além de dados os modens podem também trocar informações estatísticas (com relação às características do canal, por exemplo), executar operações como *bit swapping*, dentre outras funções [24].

2.2 O *framework* Ptolemy II

O Ptolemy II é uma extensão do *framework* Ptolemy Classic [25], e foi criado na Universidade de Berkeley da California (UC Berkeley) em 1996 pelo grupo do Prof. Edward A. Lee. O Ptolemy Classic era escrito em C++ e foi substituído pelo Ptolemy II [26] após o lançamento de sua última versão, 0.7.1, em 12 de junho de 1998.

O Ptolemy II se encontra atualmente na versão 8.0.1 e consiste em um *framework* em código aberto, totalmente escrito em Java, para projetos e simulações orientados a "atores" (ou blocos). Cada ator possui um *software* implementado de forma a processar dados que sejam disponibilizados em suas portas de entrada e enviar os resultados para suas portas de saída.

Modelos podem ser gerados através da interconexão entre as portas desses atores, que dessa forma poderão trocar mensagens entre si. No caso de um sistema de processamento de sinais

[2] A taxa de erros de bits (BER, do inglês *bit error rate*) permitida para os sistemas ADSL é de 10^{-7}.

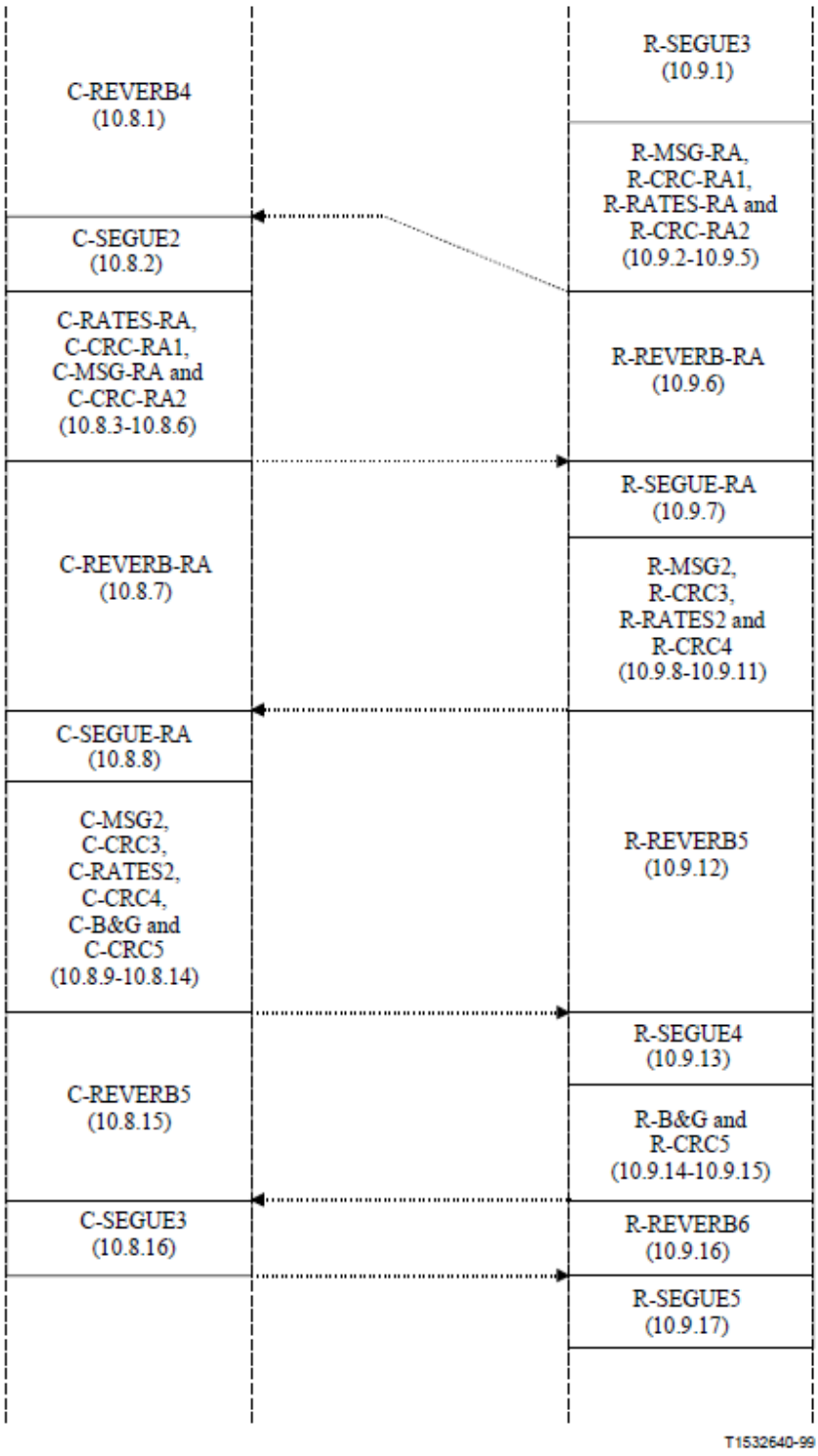


Figura 2.5: Linha do tempo da etapa de troca de informações da inicialização (retirada de [1]).

e comunicações [27], como, por exemplo, o *software*-modem deste trabalho, essas mensagens são os próprios sinais, e o *software* embarcado em cada ator representa algum algoritmo de

processamento de sinais.

Ele implementa o que se chama de "modelos de computação", que definem como os atores de um modelo são escalonados e executados. O modelo de computação mais apropriado para sistemas de processamento de sinais é o Synchronous Dataflow (SDF) e foi esse, portanto, o modelo escolhido para o Hermes. Uma explicação detalhada desses modelos de computação fugiria do escopo deste trabalho; porém, o Ptolemy II possui uma vasta documentação na *web*, e uma ótima fonte de referência sobre o seu funcionamento, incluindo os seus modelos de computação, pode ser encontrada em [28].

Cabe aqui ressaltar que o Ptolemy II é utilizado não apenas por pesquisadores e engenheiros, mas também por empresas de vários ramos. A Agilent Technologies, por exemplo, adaptou o código do Ptolemy II para desenvolver um simulador próprio de sistemas digitais, de rádio-frequência e microondas, o Advanced Design System (ADS) [29].

Os modelos do Ptolemy II podem ser visualizados, editados e simulados através do aplicativo Vergil (que faz parte do Ptolemy II). A Figura 2.6 mostra um modelo de exemplo do Ptolemy II aberto no Vergil.

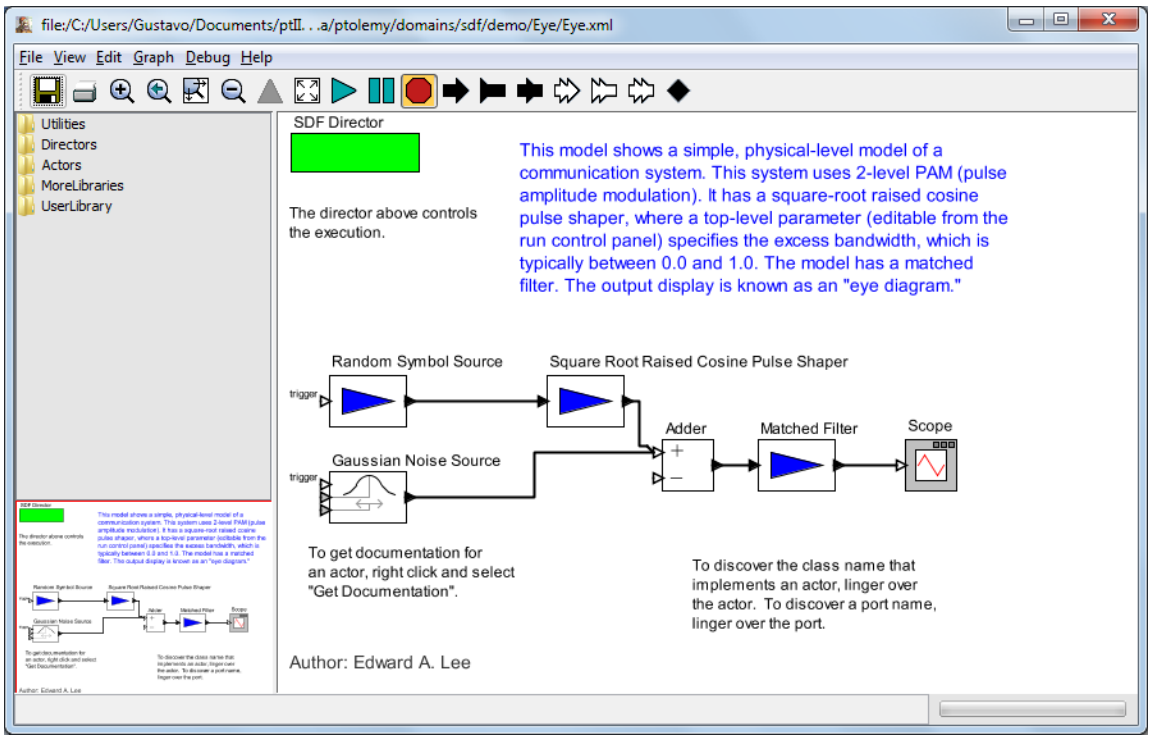


Figura 2.6: Modelo de exemplo do Ptolemy II.

A área maior é a área de edição do modelo; a área menor, à esquerda e abaixo, é a área de

visualização total do modelo; e a área à esquerda e acima é a biblioteca do Ptolemy II. Nela, estão disponíveis vários atores já implementados pelo grupo de desenvolvimento do Ptolemy II, o que facilita a criação de novos modelos.

Os modelos criados através do Vergil podem ser salvos em arquivos XML, assim como novas bibliotecas de atores criadas pelo usuário.

Capítulo 3

Projeto e implementação

O projeto e a implementação do Hermes podem ser divididos basicamente em 3 partes:

- Transmissor;
- Receptor;
- Controle dos blocos.

3.1 O Transmissor

O modelo de referência apresentado em [1] (Figura 3.1) define um transmissor ADSL, apresentando uma estrutura de blocos de processamento digital de sinais facilmente adaptável ao ambiente Ptolemy.

Esta seção descreve os blocos e funções desse transmissor, além dos detalhes de sua implementação.

3.1.1 Multiplexação e construção de quadros

Um transmissor ADSL precisa, primeiramente, agrupar alguns canais de dados de entrada em um fluxo único que é, então, armazenado e transformado num quadro (também conhecido como *frame*). Cada quadro é entregue, então, às etapas seguintes para serem

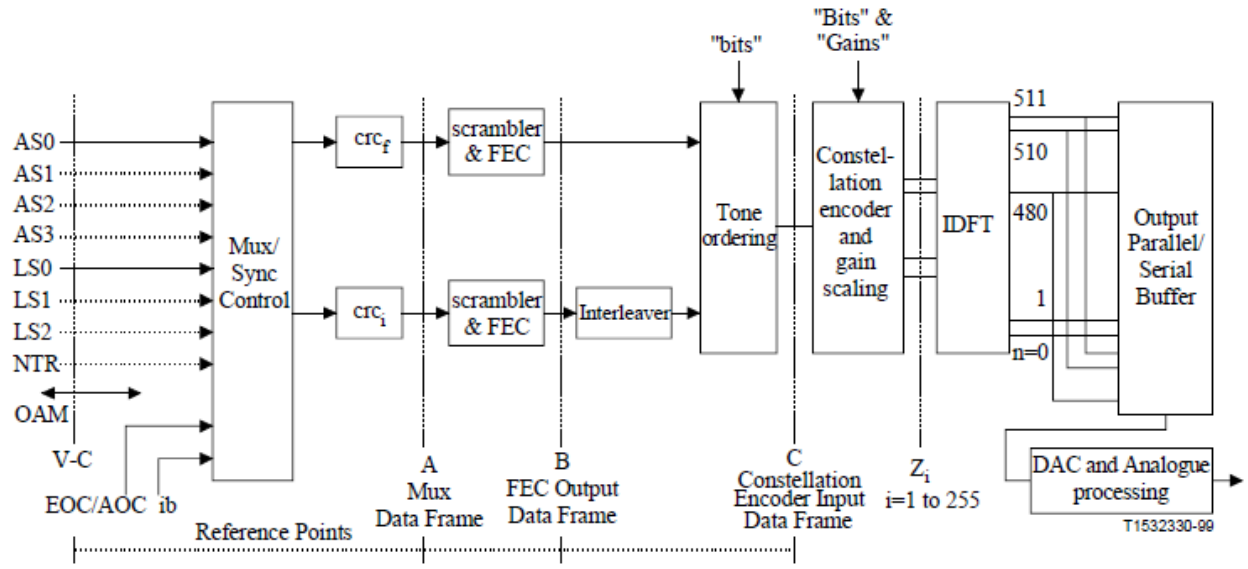


Figura 3.1: Modelo de referência do transmissor ADSL (retirado de [1]).

processados individualmente[1].

Os dados de entrada que precisam ser agrupados num quadro são os dados provenientes dos canais portadores e os dados de redundância, que são explicados nas seções a seguir.

3.1.1.1 Canais portadores

Em sistemas ADSL, os dados de entrada referentes às aplicações do usuário (navegadores de internet, comunicadores de mensagens instantâneas, etc) podem provir de 2 tipos de canais, denominados canais portadores (*bearer channels*): canais AS e canais LS. Os canais AS são unidirecionais e seus dados só podem fluir no sentido *downstream* - ou seja, do modem da operadora (ATU-C) para o modem do usuário (ATU-R); enquanto isso, os canais LS são bidirecionais - ou seja, seus dados podem fluir nas duas direções simultaneamente (*downstream* e *upstream*).

Os canais AS0 e LS0 são obrigatórios para o modem da operadora (ATU-C), sendo opcional a disponibilização de mais 3 canais AS (AS1, AS2 e AS3) e de mais 2 canais LS (LS1 e LS2).

[1]Essa operação também é denomidada de conversão serial-paralelo, já que os dados entram de forma serial (bit a bit) e saem de forma paralela (quadros).

Para o modem do usuário (ATU-R), apenas o canal LS0 é obrigatório, e os canais LS1 e LS2 são opcionais.

A taxa de dados individual para cada um desses canais pode variar em múltiplos inteiros de 32 kbps (*kilobits* por segundo), de acordo com a Tabela 3.1.

Tabela 3.1: Taxas de dados para os canais ASx e LSx.

Bearer channel	Taxa mínima (kbps)	Taxa máxima (kbps)
AS0	32	6144
AS1	32	4608
AS2	32	3072
AS3	32	1536
LS0	32	640
LS1	32	640
LS2	32	640

A taxa de 32kbps é obtida através de álgebra simples, baseado na taxa de símbolos adotada para os sistemas ADSL, que é de 4000 símbolos por segundo. Um símbolo ADSL correponde a um *frame*. Portanto, ao se dividir a taxa de cada canal por 4000, obtém-se o número de bits que cada canal ocupa em um quadro.

Por exemplo, o canal AS0 tem um número de bits por *frame* que pode variar de $32000/4000 = 8$ bits até $6144000/4000 = 1536$ bits, em múltiplos inteiros de $32000/4000 = 8$ bits.

A exigência de se utilizar múltiplos inteiros de 8 bits por quadro facilita a troca de informações com algoritmos que trabalhem com bytes ao invés de bits (como alguns códigos corretores de erros, por exemplo [30]).

3.1.1.2 Dados de redundância

Alguns dados adicionais precisam ser anexados aos dados de aplicações para formar os quadros. São dados de redundância, referidos como *ADSL system overhead* [31], que são listados a seguir:

- Canal de operações embarcadas (*Embedded Operations Channel* - EOC);
- Canal de controle de redundância (*ADSL Overhead Control Channel* - AOC);
- Bytes de checagem de redundância cíclica (*Cyclic Redundancy Check* - CRC);
- Bits de Operação, Administração e Manutenção (*Operation, Administration and Maintenance* - OAM);
- Bytes de redundância em avanço (*Forward Error Correction* - FEC).

Os bytes de redundância FEC, utilizando codificação Reed Solomon, são gerados pelo bloco ReedSolomonActor (descrito mais adiante) e inseridos em cada *frame*.

3.1.1.3 Construção de quadros

A construção de um quadro é feita a partir da junção estruturada dos dados dos canais portadores e de redundância. Essa junção é feita de modo a se produzir um "superquadro" formado por 68 quadros consecutivos. Em outras palavras, a cada 68 quadro um superquadro é formado.

Entre cada superquadro, um símbolo de sincronia é enviado. Esse símbolo não contém informação útil - apenas uma sequência de bits pseudo-aleatórios modulados através de QAM com 2 bits por tom, servindo unicamente para facilitar a sincronização do receptor com o transmissor.

A multiplexação e a construção de quadros foram implementadas através dos seguintes blocos:

- MultiplexerActor
- CRCActor
- ScramblerActor
- ReedSolomonActor
- InterleaverActor

Eles são ligados serialmente, como mostra a Figura 3.2; entretanto, suas taxas de processamento variam entre si.

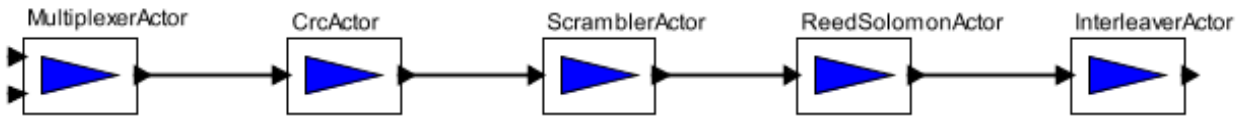


Figura 3.2: Os blocos do Hermes que implementam multiplexação e construção de quadros.

O funcionamento desses blocos é explicado a seguir.

O bloco MultiplexerActor recebe como entrada (em bits e serialmente) os dados provenientes dos canais ASx e LSx, além dos dados de redundância, e os monta numa estrutura de quadro, na forma de um vetor de bits. Quando esse vetor é totalmente preenchido, o quadro resultante é enviado para o bloco CRCActor (em forma de bytes).

O bloco CRCActor possui 3 parâmetros que devem ser ajustados durante a inicialização do modem:

- GenPoly: indica qual o polinômio gerador que será utilizado para o cálculo dos bits de CRC do superquadro. O polinômio gerador utilizado em ADSL está definido em [1];
- FrameLength: indica o tamanho de um quadro, em bytes;
- SuperframeLength: indica o tamanho de um superquadro, em número de quadros.

Esse bloco recebe bytes serialmente, diretamente do multiplexador, e os repassa à saída também serialmente, mas em forma de bits. Porém, ele mantém uma cópia dos bytes recebidos armazenada em um registro de tamanho $68 \times$ (*tamanho do quadro em bytes*). Dessa forma, quando os bytes preenchem um superquadro completo, é possível calcular o código CRC desse superquadro e concatená-lo aos seus últimos bits.

O bloco ScramblerActor recebe serialmente os bits enviados pelo CRCActor. Ele mantém um registro em fila do tipo FIFO (*first in first out*) dos últimos 23 bits recebidos, suficentes para se executar a operação de embaralhamento (*scrambling*). O embaralhamento consiste basicamente em aplicar a operação

$$b[k] = b[k-9] \oplus b[k-23]$$

aos bits atuais (no tempo discreto k), embaralhando-os, portanto, de uma forma controlada. O objetivo desse embaralhamento controlado é evitar que sequências longas de zeros ou uns sejam transmitidas, o que dificultaria a sincronização do receptor. Como será visto, no receptor esses bits são desembaralhados aplicando-se uma operação similar, recuperando-se assim os bits originais.

Após executar o embaralhamento, o ScramblerActor envia os bits resultantes, em forma de bytes, para o bloco ReedSolomonActor.

O bloco ReedSolomonActor recebe os bytes do ScramblerActor e, ao mesmo tempo em que preenche um registro com o tamanho da palavra-código utilizada (o qual é definido durante a inicialização), repassa esses bytes à saída. Após cada preenchimento do registro, bytes de paridade da palavra-código são calculados e enviados para a saída. O número de bytes de paridade também é definido durante a inicialização do modem.

Finalmente, o bloco InterleaverActor recebe os bytes enviados pelo ReedSolomonActor, entrelaça-os através de uma operação de convolução e envia o resultado para a saída serialmente em forma de bits.

3.1.2 Ordenação de tons

Os bits enviados pelo InterleaverActor estão prontos para serem separados em palavras individuais e modulados através de QAM. Entretanto, para maior robustez de transmissão, a Recomendação [1] pede que os bits sejam, primeiramente, ordenados de tal forma que os bits que passaram pelo *fast path* (ou seja, os bits que não foram entrelaçados) sejam modulados nos tons com constelações QAM mais densas (mais bits por símbolo), enquanto os bits provenientes do *interleaved path* sejam modulados nos tons com constelações mais esparsas (menos bits por símbolo), num processo denominado de "ordenação de tons" (*tone ordering*).

Dessa forma, os bits que não receberam a proteção do *interleaved path* têm essa defasagem compensada por serem alocados aos tons com menor incidência de *clipping* [6].

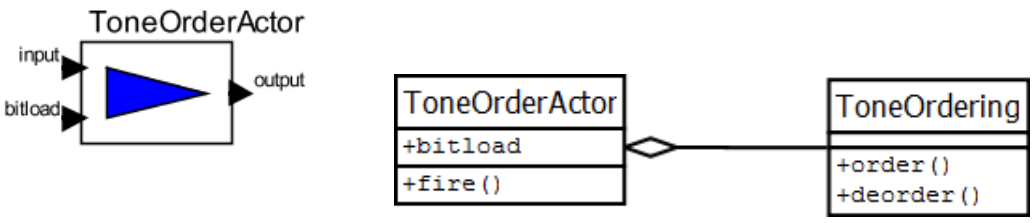
Para tons empatados em número de bits, a ordenação é feita pelo número do tom; por

exemplo, se os tons 60, 70 e 80 possuem o mesmo número de bits por símbolo, o tom 60 será alocado antes do tom 70 e este, por sua vez, será alocado antes do tom 80.

O bloco responsável pela ordenação de tons, denominado ToneOrderActor, foi implementado através de um algoritmo trivial. Basicamente, ele consiste em um *loop* que percorre todos os elementos de um vetor que contém a informação do número de bits que cada tom irá modular. Esse vetor é denominado *bitloading* (algo como "carregamento de bits") e é preenchido durante a inicialização do modem, sendo repassado ao ToneOrderActor através de uma porta de entrada.

Para cada iteração do loop, o tom com menor número de bits b (e menor numeração, em caso de empate) receberá os primeiros b bits que entraram no ToneOrderActor. O tom então é marcado como já percorrido e não será mais buscado nas próximas iterações. As outras iterações executam a mesma coisa, alocando os próximos b bits ao tom atual e assim por diante. Tons com zero bits são simplesmente ignorados.

A Figura 3.3(a) destaca o bloco ToneOrderActor e suas portas de entrada e saída, enquanto a Figura 3.3(b) mostra o diagrama UML de sua implementação.



(a) (b)

Figura 3.3: O bloco ToneOrderActor no Ptolemy II (a) e seu diagrama UML (b).

3.1.3 Modulação em amplitude e quadratura (QAM)

O tipo de modulação escolhido para os sistemas ADSL é a modulação em amplitude e quadratura, ou QAM - do inglês *Quadrature Amplitude Modulation* [32].

Cada tom deve ter um modulador QAM próprio, e o tamanho de sua constelação pode variar de 4 símbolos (2 bits por símbolo) a 32768 símbolos (15 bits por símbolo).

Os tipos de constelação utilizados variam dependendo do número de bits por símbolo:

- Para constelações pares - constelações de b bits por símbolo em que b é par -, utiliza-se o formato quadrado. Um exemplo trivial, com $b = 2$, $2^b = 4$ é mostrado na Figura 3.4.

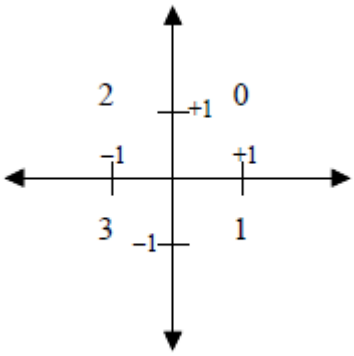


Figura 3.4: A constelação 4-QAM ADSL (retirado de [1]).

- Para constelações com b ímpar e maior que 3, usa-se um formato próximo a um quadrado, porém com os cantos vazios. Um exemplo com $b = 5$, $2^b = 32$ é mostrado na Figura 3.5.

	24	26	20	22	
19	09	11	01	03	17
18	08	10	00	02	16
31	13	15	05	07	29
30	12	14	04	06	28
	25	27	21	23	

Figura 3.5: A constelação 32-QAM ADSL (retirado de [1]).

- Para o caso especial da constelação 8-QAM, ou seja, com $b = 3$, $2^b = 8$, usa-se um formato semelhante a uma estrela retorcida, como se pode ver na Figura 3.6.

As constelações QAM em ADSL têm seus símbolos numerados de acordo com a codificação Gray, que permite que símbolos vizinhos difiram entre si em apenas um de seus bits [33, 34]. Dessa forma, torna-se mais fácil para o código corretor de erros (Reed Solomon, no caso dos sistemas ADSL) corrigir erros de bits causados quando o símbolo demodulado está incorreto, mas próximo do símbolo original.

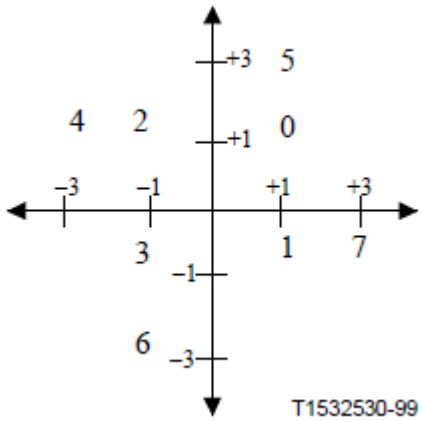


Figura 3.6: A constelação 8-QAM ADSL (retirado de [1]).

As constelações pares com $b > 2$ são geradas recursivamente a partir da constelação 4-QAM ($b = 2$) e as constelações ímpares com $b > 5$ são geradas recursivamente a partir da constelação 32-QAM ($b = 5$). Para isso, usa-se o modelo mostrado na Figura 3.7.

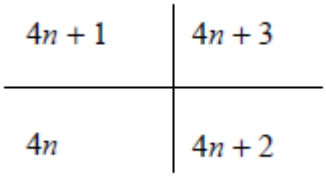


Figura 3.7: O modelo utilizado para gerar constelações QAM recursivamente em ADSL (retirado de [1]).

Dessa forma, para se gerar a constelação 128-QAM ($b = 7$), por exemplo, substituem-se os símbolos da constelação 32-QAM pelos símbolos calculados através do modelo da Figura 3.7. A costelação 256-QAM ($b = 8$) pode ser gerada a partir da constelação 128-QAM, através do mesmo método, o mesmo valendo para as constelações maiores.

Da mesma forma, a constelação 16-QAM ($b = 4$) pode ser gerada a partir da constelação 4-QAM; a constelação 64-QAM ($b = 6$) a partir da 16-QAM; e assim em diante.

O número máximo de bits por símbolo para os sistemas ADSL foi definido como 15 e o número mínimo como 2.

A Recomendação [1] também define constelações para codificação por treliça (*trellis coding*) [35]. Entretanto, ela não é mandatória e não foi implementada neste trabalho.

A modulação QAM foi implementada através de uma estrutura de banco de moduladores individuais. Para otimizar o uso de memória, apenas um modulador para cada número de bits (de 2 a 15) é instanciado durante a inicialização do *software*.

Cada modulador cria, então, uma matriz de números inteiros representando um constelação QAM (gerada recursivamente através do método explicado anteriormente) com o número de bits correspondente e com os índices de linha e coluna representando os eixos x e y dessa constelação.

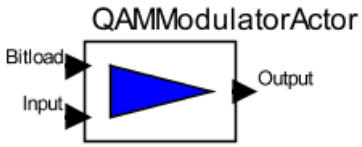
Esses índices são números inteiros consecutivos não negativos, enquanto as constelações têm seus eixos representados por números inteiros ímpares que podem ser tanto negativos quanto positivos. A conversão desses índices para os valores verdadeiros dos eixos foi, portanto, levada em conta na implementação dos moduladores.

Para modular uma palavra de b bits, a instância do modulador converte essa palavra em um número inteiro e faz uma busca simples nessa matriz, localizando os índices x e y que apontam para esse número inteiro e calculando o valor complexo correpondente.

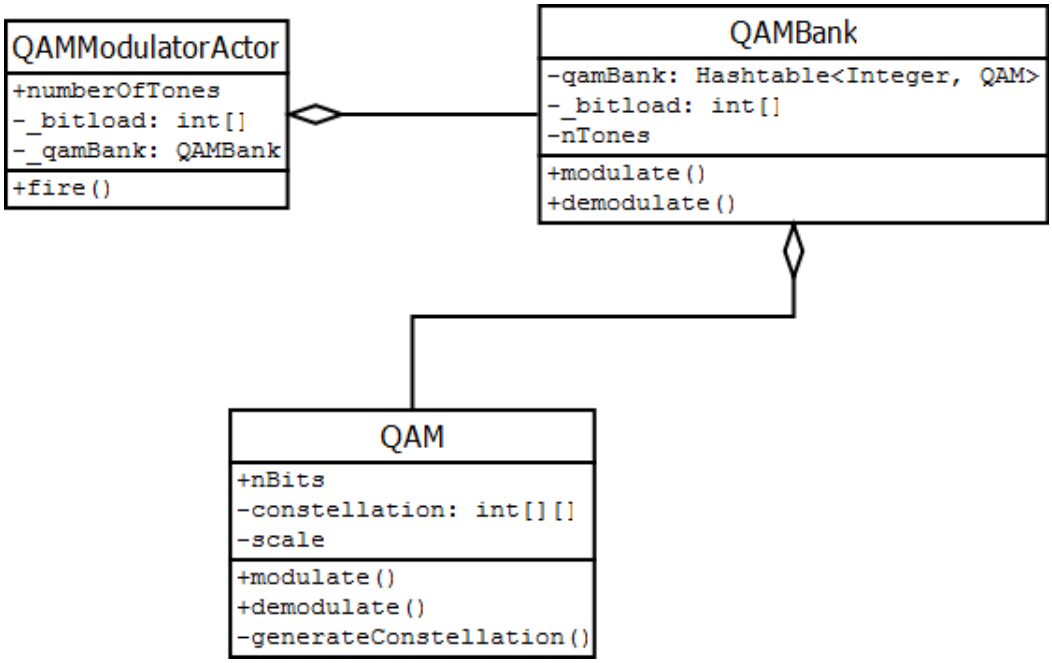
Para se fazer a modulação de todos os tons, segue-se o algoritmo descrito abaixo:

```
Para todos os tons t de 0 a 255 {
 b(t) = número de bits que o tom t carrega;
 Se b = 0 {
  Considere o valor complexo resultante da modulação igual a 0+0i;
 } Senão {
  Utilize a instância do modulador QAM de b bits para modular esse tom;
 }
}
```

A Figura 3.8(a) mostra o bloco QAMModulatorActor, enquanto a Figura 3.8(b) ilustra seu diagrama UML.



(a)



(b)

Figura 3.8: O bloco QAMModulatorActor no Ptolemy II (a) e seu diagrama UML (b).

3.1.4 Escalamento da densidade espectral de potência (PSD)

Após a modulação, os símbolos resultantes (números complexos) precisam ser escalados para que a potência alocada para cada tom seja obtida na transmissão. O conjunto das potências por tom numa transmissão ADSL é chamado de *power spectral density* (PSD, ou densidade espectral de potência em português) e é calculado durante a inicialização do modem, levando-se em conta estimações das características do canal, como sua atenuação e sua SNR.

A implementação do escalamento da PSD é trivial, multiplicando-se, elemento a elemento, um vetor de números reais (representando a PSD) por um vetor de números complexos (representando os símbolos QAM de saída). Essa multiplicação foi incorporada no bloco QAMModulatorActor.

3.1.5 Transformada rápida de Fourier inversa (IFFT)

As transformadas rápidas de Fourier direta (FFT) e inversa (IFFT) são a base de sistemas de modulação multi-portadora eficientes como DMT e OFDM [36, 37].

Diferentemente dos sistemas OFDM, entretanto, os sistemas DMT usam a banda base para transmissão da informação. Por isso, as amostras a serem enviadas devem ter parte imaginária nula. Para que essa condição seja satisfeita, é preciso modificar os símbolos de entrada de modo a dotá-los de simetria hermitiana. A simetria hermitiana para um vetor de números complexos é obtida anexando-se a ele mesmo sua versão rebatida e conjugada, como ilustra a Figura 3.9.

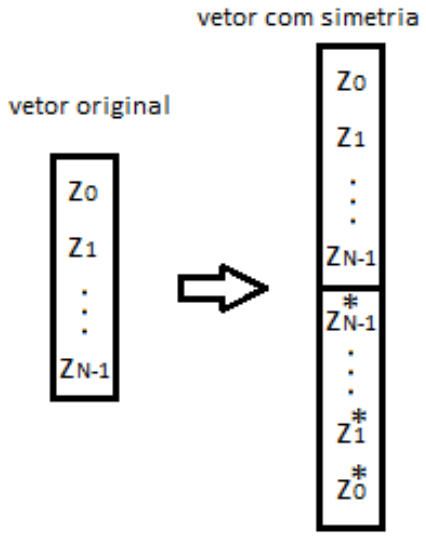


Figura 3.9: Obtenção de simetria hermitiana para um vetor de números complexos.

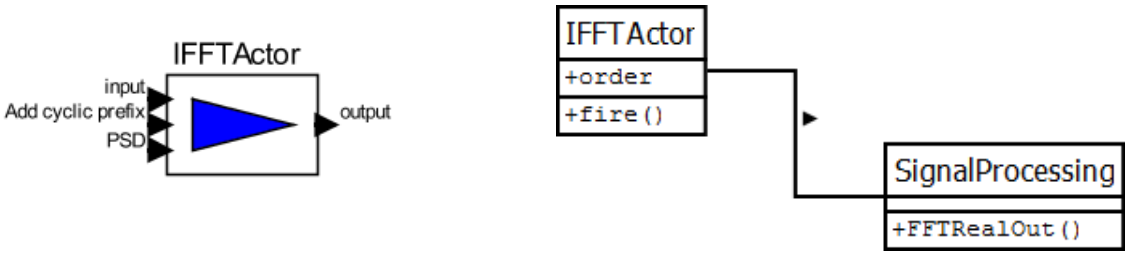
A implementação da IFFT foi feita através da criação do bloco IFFTActor que, por sua vez, utiliza uma classe pertencente ao PtolemyII, `SignalProcessing.java`, para executar a transformada.

Antes de executar a transformada, um método para criar um vetor de símbolos a partir do vetor original, porém com simetria hermitiana, é chamado. Isso garante que as amostras resultantes da IFFT terão parte imaginária nula.

Para executar a transformada propriamente dita, o IFFTActor chama o método estático `IFFTRealOut()` da classe `SignalProcessing.java`. Esse método foi escolhido por ser otimizado para realizar a IFFT de valores complexos quando se sabe previamente que sua

saída será real, pois ele realiza apenas as multiplicações necessárias (metade do número de multiplicações original), eliminando boa parte do processamento necessário e diminuindo, portanto, sua complexidade computacional e seu uso de memória.

A figura 3.10(a) mostra o bloco IFFTActor e suas portas, enquanto a Figura 3.10(b) mostra seu diagrama UML.



(a) (b)

Figura 3.10: O bloco IFFTActor no Ptolemy II (a) e seu diagrama UML (b).

3.1.6 Inserção de prefixo cíclico

O último passo para enviar a informação para fora do modem (através de um AFE, *Analog Front End*, que converterá as amostras digitais em um sinal de tensão a ser aplicado no canal telefônico) é inserir no início de cada símbolo um prefixo cíclico, que é nada mais do que uma cópia das últimas L amostras do símbolo. A Figura 3.11 ilustra esse procedimento.

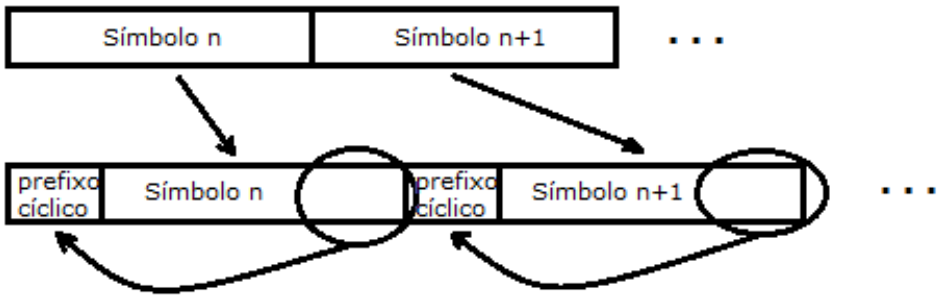


Figura 3.11: Processo de inserção do prefixo cíclico.

Seu objetivo é servir como um intervalo de guarda entre símbolos consecutivos, simplificando o processo de equalização do canal no receptor [38].

Em ADSL, o tamanho escolhido para o prefixo cíclico foi $L = 32$. Sua inserção foi incorporada ao bloco IFFTActor.

3.2 O Receptor

Não há em [1] um modelo de referência para o receptor ADSL; apenas o modelo do transmissor é apresentado (ver Figura 3.1). O receptor deste trabalho foi, então, projetado de forma a executar, inversamente, todos os procedimentos do transmissor, além de funções relacionadas a sincronização, equalização, etc, recuperando as informações enviadas pelo transmissor da forma mais robusta possível.

Esta Seção descreve os blocos do receptor do Hermes.

3.2.1 Sincronização de clock e de símbolo

A sincronização de clock e de símbolo ainda se encontra em fase de implementação. Seus passos consistem basicamente em:

1. Sincronização de clock: ajustar a frequência de amostragem do conversor analógico/digital do AFE, de modo a compensar flutuações de frequência no sinal transmitido pelo modem do outro lado;
2. Sincronização de símbolo: identificar qual amostra corresponde à amostra inicial de cada símbolo.

Apesar de serem aparentemente triviais, esses dois passos exigem a implementação de algoritmos relativamente complexos. Há livros inteiros sobre sincronização de sistemas de telecomunicações, inclusive para sistemas DMT/ADSL. Mais detalhes podem ser encontrados em [39].

3.2.2 Equalização no domínio do tempo (TEQ)

Após a sincronização, o sinal no domínio do tempo deve ser equalizado para eliminar a interferência inter-símbolos (ISI) causada pelo canal. Pelo fato de possuirem prefixo cíclico,

os símbolos recebidos do canal não têm tanta ISI quanto teriam sem o prefixo. Dessa forma, o TEQ não necessita equalizar totalmente o canal, mas apenas fazer com que seu comprimento L seja menor que o do prefixo cíclico, ou seja, $L < 32$. Esse processo é denominado de *channel shortening* (algo como "diminuição de canal" em português) e, por isso, muitas vezes o TEQ é referido não como um equalizador, mas como um *channel shortener*.

O TEQ é, basicamente, um filtro FIR através do qual o sinal recebido pelo modem ADSL passa, de forma que sua ISI seja eliminada através do encurtamento do canal. Os coeficientes desse filtro podem ser fixos ou (o que é mais comum) ajustados dinamicamente durante a inicialização e até mesmo em *showtime*, de forma a se adapatar às variações do canal. Nesse caso, o TEQ é um filtro adaptativo.

Esse ajuste dos coeficientes do TEQ deve ser feito através de algum algoritmo específico. Há vários algoritmos na literatura para treino de filtros adaptativos [40] e, consequentemente, de TEQ's, e essa área continua sendo objeto de intenso estudo atualmente.

Assim como a sincronização, a equalização no tempo ainda se encontra em fase de implementação neste trabalho.

3.2.3 Retirada do prefixo cíclico

Da mesma forma que a sua inserção, a retirada do prefixo cíclico é trivial e foi incorporada ao bloco FFTActor, que é descrito a seguir.

3.2.4 Transformada rápida de Fourier (FFT)

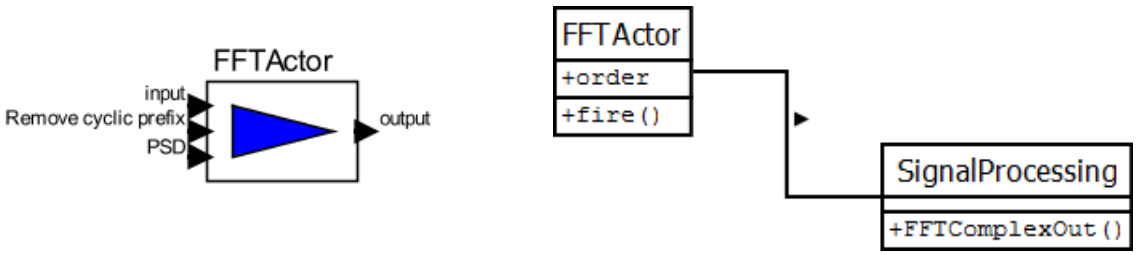
Após ter o prefixo cíclico retirado, os símbolos provenientes do canal devem ser levados para o domínio da frequência através de uma FFT (Transformada Rápida de Fourier). Após isso, metade dos símbolos de saída também devem ser descartados, pois são redundantes e foram inseridos apenas para se obter simetria hermitiana.

O bloco responsável por essas 3 operações consecutivas - retirar o prefixo cíclico, aplicar a FFT e descartar a informação de simetria hermitiana - é o FFTActor.

Sua implementação é semelhante à do bloco IFFTActor, também utilizando a classe

`SignalProcessing` do Ptolemy II; porém, o FFTActor utiliza o método `FFTComplexOut()`. Esse método é otimizado para receber entradas com parte imaginária nula.

A Figura 3.12(a) mostra o bloco FFTActor e suas portas. A Figura 3.12(b) mostra o diagrama UML de sua implementação.



(a) (b)

Figura 3.12: O bloco FFTActor no Ptolemy II (a) e seu diagrama UML (b).

3.2.5 Equalização no domínio da frequência (FEQ)

A equalização no domínio da frequência serve para completar o trabalho do equalizador no domínio do tempo (TEQ).

O TEQ, como explicado anteriormente, se encarrega de eliminar a ISI entre os símbolos no domínio do tempo. Porém, ele não elimina os efeitos de atenuação e de rotação de fase por tom do canal. Se esses efeitos não são eliminados, torna-se impossível executar a demodulação dos símbolos QAM de forma correta.

Para fazer este trabalho, o FEQ precisa, durante a inicialização, fazer uma estimação desses efeitos.

O FEQ implementado neste trabalho, representado pelo bloco FEQActor, utiliza o algoritmo do mínimo quadrado médio (*least mean square* - LMS) para estimar esses valores de atenuação (magnitude) e rotação (fase), que podem ser representados por números complexos.

Para isso, uma sequência pseudo-aleatória, denominada sequência de treino e previamente conhecida pelo transmissor e pelo receptor, é enviada várias vezes seguidas, em número suficiente para que se obtenha uma estimativa próxima o bastante do canal real.

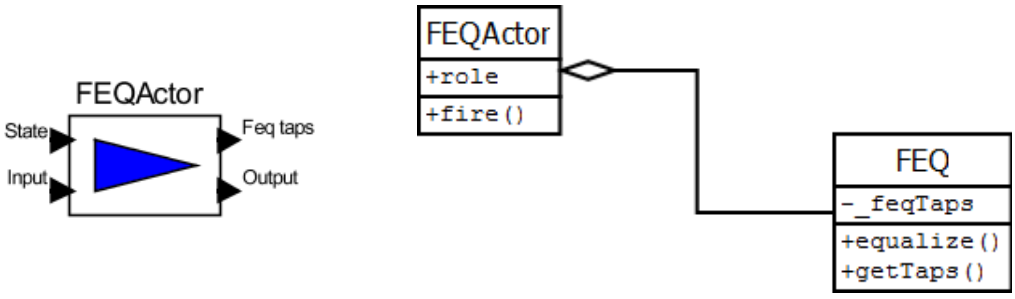
A estimativa LMS $\hat{H}_n$ para cada tom n é obtida iterativamente minimizando-se o erro e através das equações:

$$e_{n,k} = y_{n,k} - \hat{H}_{n,k-1}x_n$$

$$\hat{H}_{n,k} = \hat{H}_{n,k-1} + \mu e^*_{n,k}x_n$$

com k sendo a iteração atual, μ o fator de treino, y_n o símbolo recebido no tom n e x_n o símbolo de treino x no mesmo tom.

A Figura 3.13(a) mostra o bloco FEQActor e suas portas, enquanto a Figura 3.13(b) ilustra sua implementação em UML.



(a) (b)

Figura 3.13: O bloco FEQActor no Ptolemy II (a) e seu diagrama UML (b).

3.2.6 Re-escalamento da PSD

O re-escalamento da PSD é trivial e consiste basicamente em desfazer a operação de escalamento executada pelo bloco QAMModulatorActor, através de uma divisão elemento a elemento do vetor de símbolos de entrada pelo vetor da PSD. Essa operação foi incorporada ao bloco QAMDemodulatorActor, descrito mais adiante.

3.2.7 Estimação da razão sinal/ruído (SNR)

Como será visto mais adiante, uma das etapas de uma transmissão ADSL é a estimação das SNR's por tom do canal, para que o vetor de *bitloading* - que representa o número de bits

alocados para cada um dentre os N tons - possa ser calculado através da equação:

$$b_n = \log_2\left(1 + \frac{SNR_n}{\Gamma}\right),\ n = [1, N]$$

onde n é o índice do tom e Γ é a margem da SNR, definida pela operadora da transmissão.

Para realizar essa estimativa foi implementado o bloco SNREstimatorActor, que utiliza, por sua vez, a classe SNREstimator para realizar uma média aritmética simples das razões instantâneas entre a potência de cada símbolo recebido e a potência do erro entre esse símbolo e o símbolo de treino (ou seja, o símbolo que se esperava receber).

Matematicamente,

$$SNR_n = \frac{1}{K}\sum_{1}^{K}\frac{Y_{n,k}^2}{(Y_{n,k} - X_n)^2},\ n = [1, N]$$

onde Y é o símbolo recebido, X é o símbolo esperado, n é o índice do tom, k é o número do símbolo recebido e K é o total de símbolos. Em ADSL, o total de símbolos necessários para se estimar a SNR do canal é de 16384.

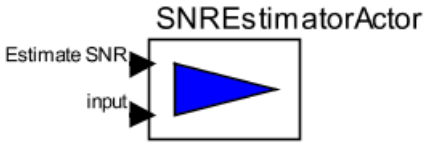
O bloco SNREstimatorActor possui duas portas de entrada: `input`, para receber os símbolos de entrada, e `Estimate SNR`, para receber um comando de ativação da estimação, término da estimação ou reinício.

A Figura 3.14(a) mostra o bloco SNREstimatorActor e suas portas, e a Figura 3.14(b) ilustra seu diagrama UML.

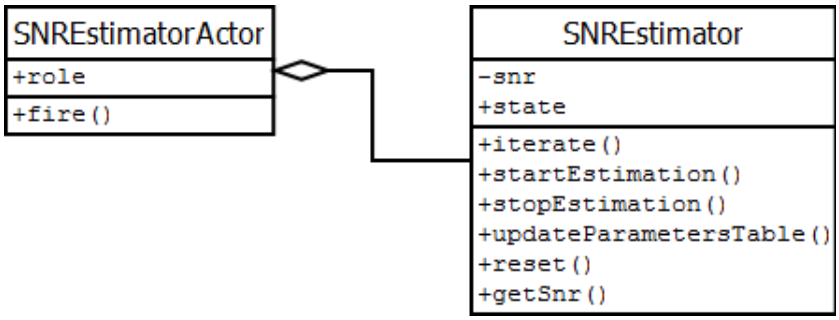
Note que esse bloco não possui nenhuma porta de saída. É que, assim que ele termina de estimar os valores de SNR por tom, escreve-os diretamente na tabela de parâmetros do modem, sem enviar nenhum tipo de dado para outros blocos.

3.2.8 Demodulação QAM

A demodulação QAM é executada pelo bloco QAMDemodulatorActor. Assim como o bloco QAMModulatorActor, ele utiliza a classe QAMBank para instanciar as classes QAM correspondentes às constelações de 2 a 15 bits. As classes QAM possuem, por sua vez, um método `demodulate()` que demodula um símbolo QAM e retorna a palavra correspondente a



(a)



(b)

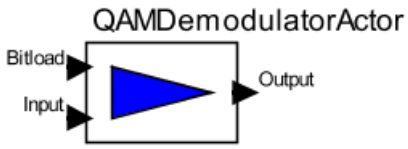
Figura 3.14: O bloco SNREstimatorActor no Ptolemy II (a) e seu diagrama UML (b).

essa demodulação.

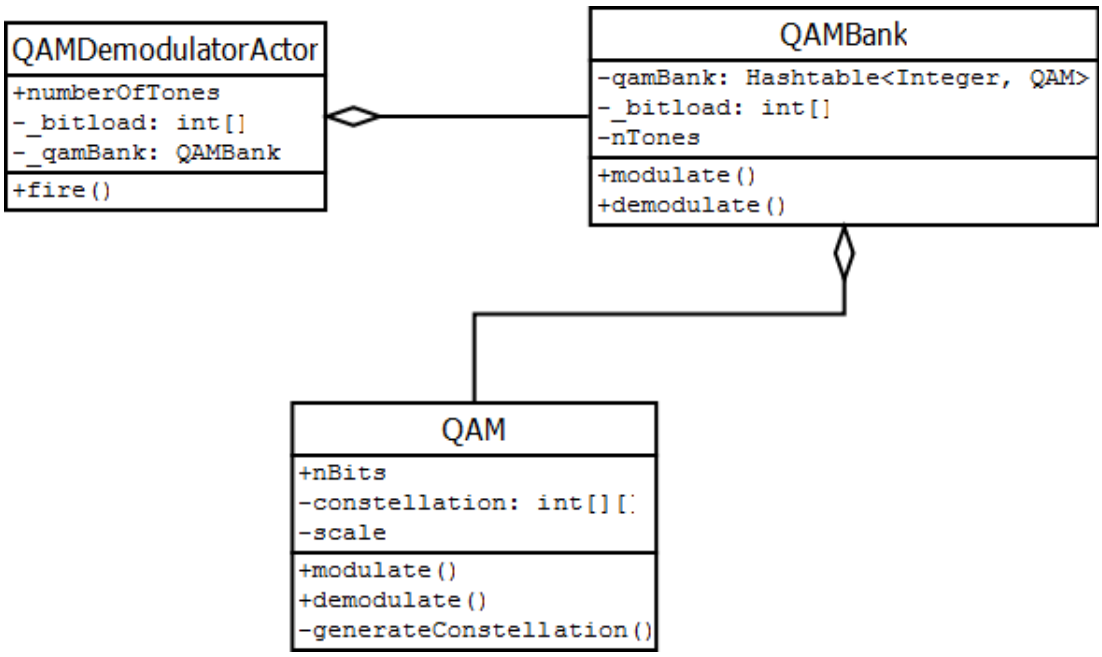
A demodulação QAM pode ser feita quantizando-se vetorialmente os símbolos recebidos, isto é, buscando-se, dentre todos os símbolos da constelação, o que estiver mais próximo (pelo critério da distância euclidiana) do símbolo recebido.

Entretanto, a demodulação através de busca simples de símbolos QAM em constelações de tamanho considerável (por exemplo, uma constelação de 10 bits por símbolo possui 1024 símbolos) pode ter um custo computacional muito alto; dessa forma, o método `demodulate()` da classe QAM foi desenvolvido de forma a buscar os símbolos através da quantização de seus valores em cada eixo (fase e quadratura). Assim, a mesma busca simples do exemplo anterior, que precisaria procurar por 1 dentre 1024 símbolos, é dividida em duas buscas, cada uma delas procurando por um valor dentre 32, o que diminui de forma considerável a sua complexidade.

A Figura 3.15(a) mostra o bloco QAMDemodulatorActor e a Figura 3.15(b) ilustra seu diagrama UML.



(a)



(b)

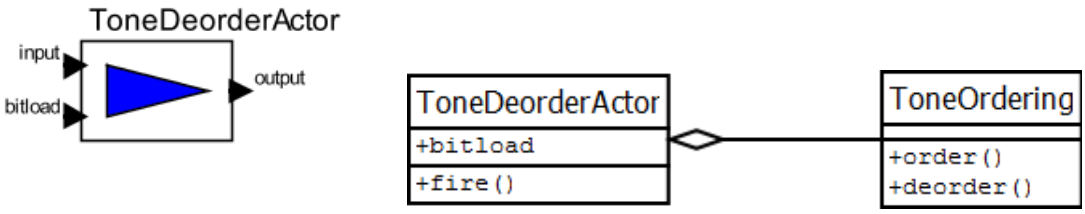
Figura 3.15: O bloco QAMDemodulatorActor no Ptolemy II (a) e seu diagrama UML (b).

3.2.9 Reordenação de tons

Após os símbolos de cada tom serem demodulados, as palavras precisam ser reordenadas para se recuperar o quadro original.

Essa tarefa é executada pelo bloco ToneDeorderActor e consiste simplesmente em executar a operação inversa à do bloco ToneOrderActor.

A Figura 3.16(a) mostra o bloco ToneDeorderActor e suas portas, enquanto a Figura 3.16(b) mostra o diagrama UML de sua implementação.



(a) (b)

Figura 3.16: O bloco ToneDeorderActor no Ptolemy II (a) e seu diagrama UML (b).

3.2.10 Desconstrução de quadros e demultiplexação

Depois de ser recuperado, cada grupo de 68 quadros consecutivos recebidos, representando 1 superquadro, deve ser desagrupado e decodificado de forma inversa à em que foi codificado no transmissor, ou seja, deve passar pelos blocos

- DeinterleaverActor;
- ReedSolomonDecoderActor;
- DescramblerActor;
- CRCDecoderActor;
- DemultiplexerActor.

nessa ordem. Cada um desses blocos executa a operação inversa ao seu correspondente no transmissor, como explicado a seguir.

O bloco DeinterleaverActor recupera os bytes desentrelaçados do frame original aplicando a mesma operação executada pelo bloco InterleaverActor.

O bloco ReedSolomonDecoderActor executa dois passos: primeiro, verifica se a palavra-código recebida contém algum erro, através da checagem de seus bits de paridade. Caso haja erros, tenta corrigí-lo para obter a mensagem original; caso contrário, envia a palavra como está, para que o bloco CRCDecoderActor se encarregue então de informar o modem que o superquadro contém um ou mais erros que não podem ser corrigidos.

O bloco DescramblerActor é responsável por desembaralhar os bits que foram embaralhados pelo bloco ScramblerActor no transmissor. Para isso, ele efetua a mesma operação de OU EXCLUSIVO executada pelo ScramblerActor, o que garante a recuperação dos bits originais.

O bloco CRCDecoderActor é responsável por checar se os superquadros recebidos contêm erros que não puderam ser corrigidos pelo ReedSolomonDecoderActor e informar isso às camadas superiores de rede, para que essas tomem alguma providência (como pedir o re-envio das mensagens corrompidas, por exemplo).

Finalmente, o bloco DemultiplexerActor efetua a separação entre os dados dos canais portadores e os dados de redundância, enviando-os para as portas correspondentes.

A Figura 3.17 mostra esses blocos conectados em série no Ptolemy. A área destacada por uma elipse tracejada indica blocos auxiliares utilizados para descartar os *depth bytes* iniciais provenientes do bloco *deinterleaver*, pois eles serão sempre nulos.

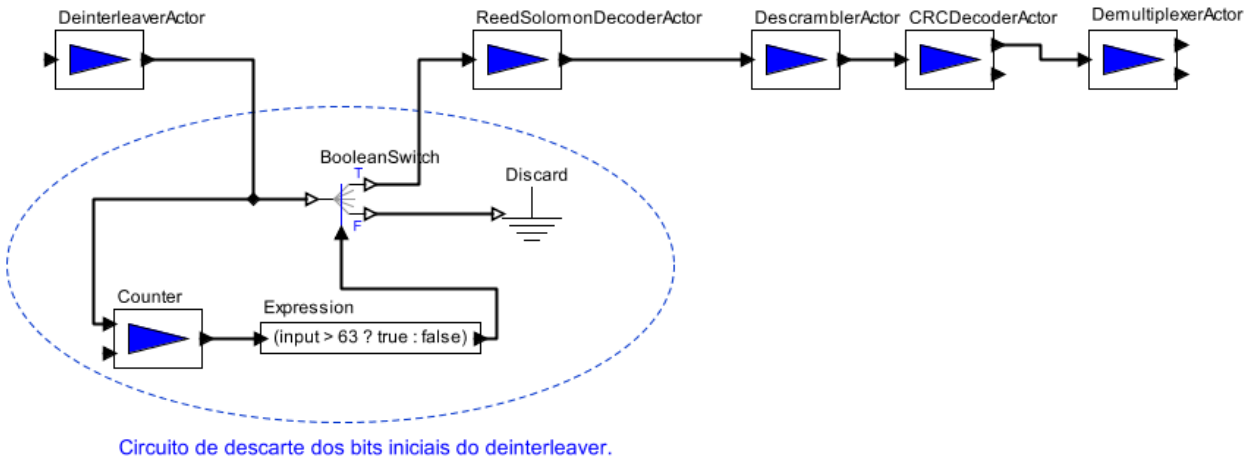


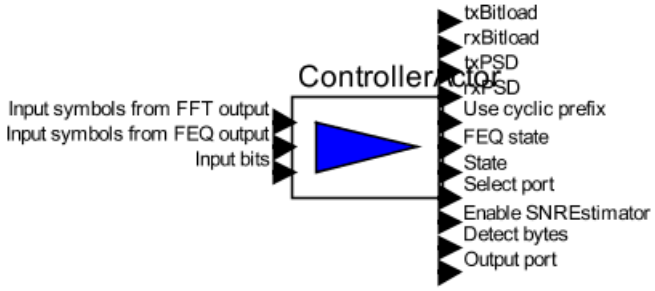
Figura 3.17: Blocos do Hermes que implementam a desconstrução dos quadros e demultiplexação.

3.3 Controle dos blocos

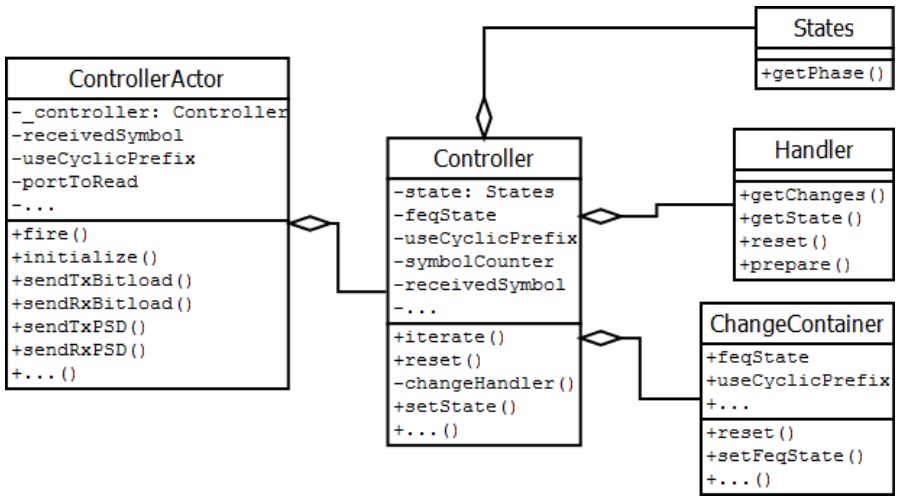
Para fazer com que todos os blocos executem suas tarefas de forma harmônica, além da definição em tempo real de seus parâmetros, acionamento de blocos específicos em

determinados momentos, dentre outras funções, é preciso que seu controle seja centralizado. Para isso, foi implementado o bloco ControllerActor.

As Figuras 3.18(a) e 3.18(b) mostram o bloco ControllerActor e seu diagrama UML (simplificado), respectivamente.



(a)



(b)

Figura 3.18: O bloco ControllerActor no Ptolemy II (a) e seu diagrama UML (b).

O ControllerActor implementa as operações necessárias para que sejam ativados os blocos corretos a cada instante da transmissão, de forma a codificar/modular as informações a serem transmitidas e demodular/decodificar as informações recebidas. Para isso, foi criado um sistema de máquina de estados e de manipuladores (*handlers*) que decide, a cada ciclo, qual sinal deve ser enviado, quais blocos devem ser ativados, qual potência e quantos bits devem ser alocados para cada tom, etc.

O bloco ControllerActor possui três portas de entrada:

- Uma para receber os símbolos QAM após a FFT (*Input symbols from FFT*);
- Uma para receber os símbolos QAM após o FEQ (*Input symbols from FEQ*);
- Uma para receber os bits após a decodificação dos símbolos QAM (*Input bits*).

Ele possui também 9 portas de saída:

- Bitload de transmissão (*txBitload*);
- Bitload de recepção (*rxBitload*);
- PSD (*Power Spectral Density*) de transmissão (*txPSD*);
- PSD de recepção (*rxPSD*);
- Porta para indicar se está usando prefixo cíclico (*Use cyclic preffix*);
- Porta para indicar o estado do FEQ (*FEQ state*);
- Porta para habilitar o estimador de SNR (*Enable SNREstimator*);
- Porta com os dados a serem modulados e transmitidos (*Output bits*);
- Porta para indicar se o modem está pronto para entrar em *showtime* (*Showtime ready*)

Capítulo 4

Simulações e resultados

Para testar e validar o Hermes, algumas simulações foram executadas, tanto *online* - ou seja, com duas instâncias do Hermes se comunicando entre si - quanto *offline* - com uma instância do Hermes recebendo o sinal digitalizado de uma transmissão já executada entre dois modens reais, através do equipamento Tracespan^{TM}.

Este capítulo descreve essas simulações e seus resultados.

4.1 Simulação *online* com ATU-C e ATU-R

A simulação *online* foi executada através da criação de duas instâncias do Hermes, uma representando o ATU-C e outra representando o ATU-R, além de dois blocos representando um canal telefônico bidirecional. O uso de dois blocos é necessário pelo fato de que o Ptolemy não permite que um bloco execute o processamento de um sinal nos dois sentidos simultaneamente, portanto um dos blocos implementa o sentido de *downstream* do canal, enquanto o outro implementa o sentido de *upstream*.

O modelo geral da simulação é mostrado na Figura 4.1.

Notam-se alguns detalhes importantes nesse modelo. O primeiro deles é que os modens ATU-C e ATU-R são implementados como instâncias de uma mesma classe de blocos, chamada aqui de ATU-x. A criação de classes de atores é um recurso do Ptolemy que permite reutilizar atores que contenham a mesma estrutura mas cujos parâmetros não sejam necessariamente iguais.

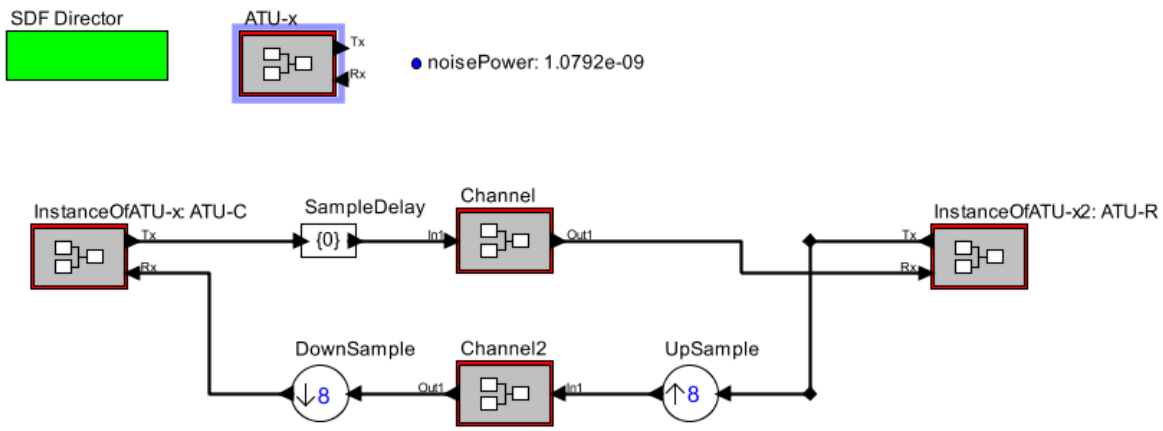


Figura 4.1: Modelo Ptolemy para simulação *online* do Hermes com ATU-C e ATU-R.

O segundo detalhe é a presença do bloco SampleDelay entre os blocos ATU-C e Channel. O bloco SampleDelay, como o nome sugere, insere um atraso entre sua entrada e sua saída. Ele é utilizado no domínio SDF do Ptolemy sempre que existe algum tipo de realimentação de sinal, para que não ocorra um *deadlock*[1]. Caso o SampleDelay não esteja presente nesses casos, o Ptolemy não inicializa a simulação, disparando uma exceção.

Outro ponto importante a ser observado é a inserção dos blocos UpSample e DownSample entre o ATU-R e o Channel2 e entre o Channel2 e o ATU-C, respectivamente. Esses blocos são inseridos para que a taxa com a qual o sinal de *upstream* (do ATU-R para o ATU-C) passa pelo canal seja igual à taxa do sinal de *downstream*, permitindo que os blocos Channel e Channel2 sejam projetados como um só canal.

Na Figura 4.2 é possível observar a estrutura interna do bloco ATU-C. Os blocos, parâmetros e junções de linhas foram destacados pelo Ptolemy para indicar que se trata de uma instância. O bloco ATU-R possui uma estrutura semelhante à do ATU-C, exceto pela diferença entre alguns parâmetros como a ordem da FFT e IFFT, número de tons, tamanho do prefixo cíclico, etc, e por isso é omitida aqui sua visão interna.

Os blocos representando o canal telefônico, Channel (sentido *downstream*) e Channel2 (sentido *upstream*), possuem a estrutura mostrada na Figura 4.3.

[1]No caso em questão, a realimentação ocorre porque o ATU-C gera um sinal que passa pelo bloco Channel, é processado pelo bloco ATU-R, depois pelo bloco Channel2 e, então, retorna ao bloco ATU-C.

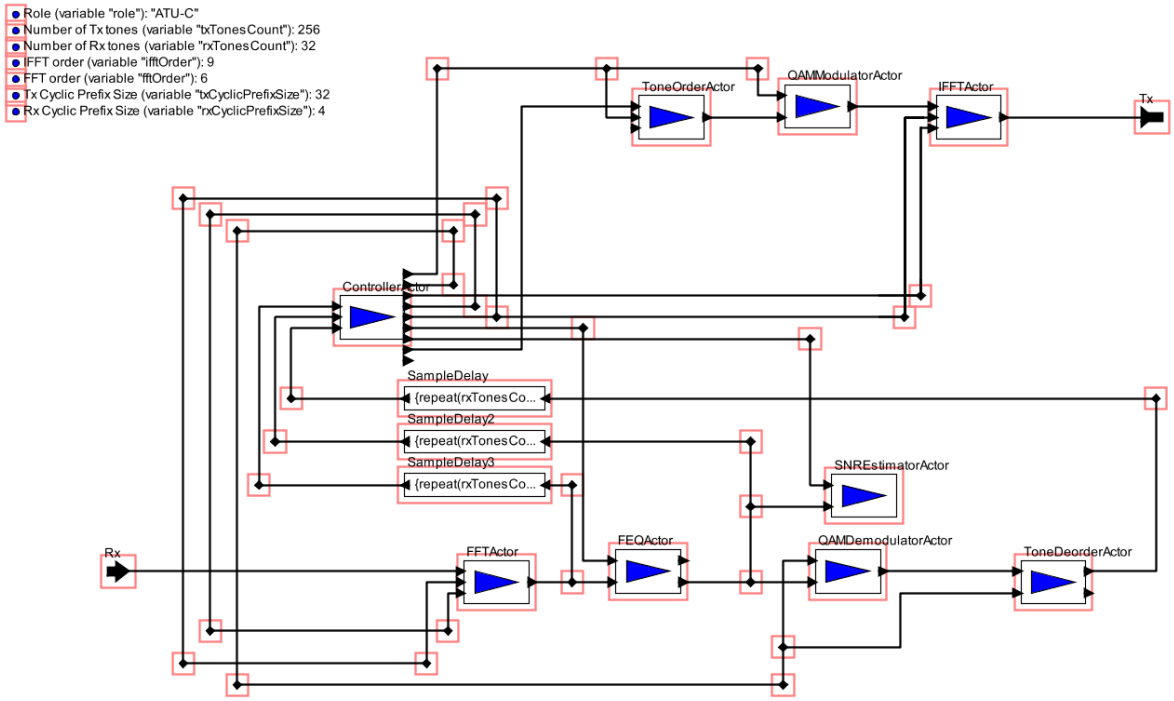


Figura 4.2: Estrutura interna do bloco ATU-C.

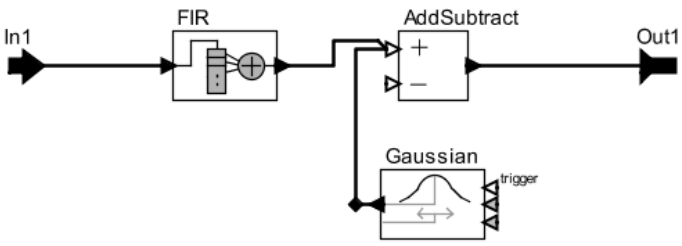
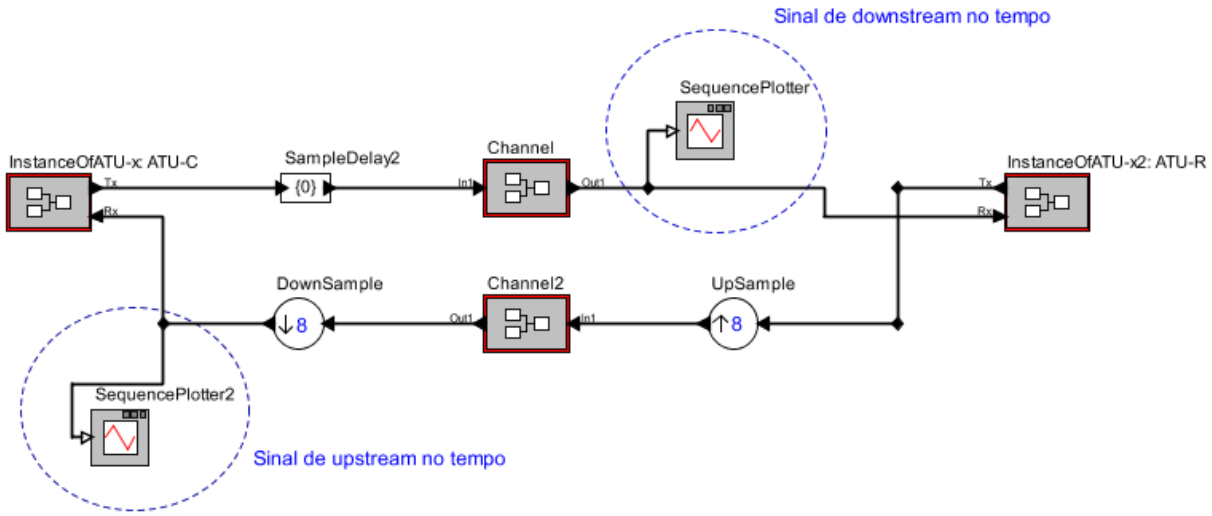


Figura 4.3: Estrutura interna dos blocos Channel e Channel2.

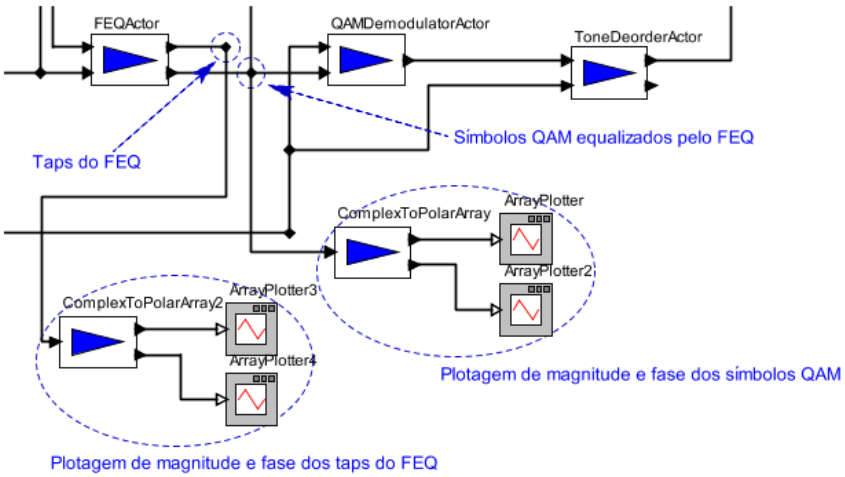
Tal estrutura consta basicamente de um filtro do tipo FIR (do inglês *Finite Impulse Response*, ou Resposta ao Impulso Infinita em português) e um bloco para adição de ruído gaussiano. Os *taps* do filtro FIR e a variância do ruído foram estimados de forma a reproduzir um canal telefônico real.

Os resultados da simulação foram observados através da inserção de blocos de plotagem de sinais e vetores, da própria biblioteca do Ptolemy, em alguns pontos de prova no modelo (tanto externa quanto internamente aos modens), como mostrado nas Figuras 4.4(a) e 4.4(b).

A Tabela 4.1 sumariza os sinais enviados pelo ATU-C e pelo ATU-R durante a fase de



(a)



(b)

Figura 4.4: Pontos de prova e plotagem de sinais para as simulações do Hermes, externa (a) e internamente aos modens (b).

inicialização.

4.1.1 Sinais

A Figura 4.5 mostra o início do envio do tom piloto do ATU-C para o ATU-R (C-PILOT1), indicando que quer iniciar uma conexão (assumindo-se que a etapa de sinalização já foi

Tabela 4.1: Sinais enviados pelo ATU-C e pelo ATU-R durante as subfases da inicialização.

ATU-C		**ATU-R**	
Training	C-PILOT1	***Training***	R-QUIET2
	C-REVERB1		R-QUIET2
	C-PILOT2		R-REVERB1
	C-ECT		R-QUIET3
	C-REVERB2		R-ECT
	C-PILOT3	***Channel Analisys***	R-SEGUE1
Channel Analisys	C-REVERB3		R-REVERB3
	C-SEGUE1		R-SEGUE2
	C-RATES1		R-RATES1
	C-CRC1		R-CRC1
	C-MSG1		R-MSG1
	C-CRC2		R-CRC2
Exchange	C-MEDLEY		R-MEDLEY
	C-REVERB4		R-REVERB4
	C-SEGUE2	***Exchange***	R-SEGUE3
	C-RATESRA		R-MSGRA
	C-CRCRA1		R-CRCRA1
	C-MSGRA		R-RATESRA
	C-CRCRA2		R-CRCRA2
	C-REVERBRA		R-REVERBRA
	C-SEGUERA		R-SEGUERA
	C-MSG2		R-MSG2
	C-CRC3		R-CRC3
	C-RATES2		R-RATES2
	C-CRC4		R-CRC4
	C-BEG		R-REVERB5
	C-CRC5		R-SEGUE4
	C-REVERB5		R-BEG
	C-SEGUE3		R-CRC5
			R-REVERB6
			R-SEGUE5

executa pelos dois modens). Nota-se a presença, principalmente antes do tom piloto, de ruído no sinal. Também é possível notar a interência entre símbolos (ISI) do canal, manifestando-se em forma de uma ligeira distorção no início do tom piloto. Na Figura 4.6, o mesmo sinal é mostrado, porém no domínio da frequência, podendo-se perceber claramente a presença de uma senóide num subcanal pouco acima do 50 (neste caso, o subcanal 64), enquanto os outros subcanais têm potência praticamente nula.

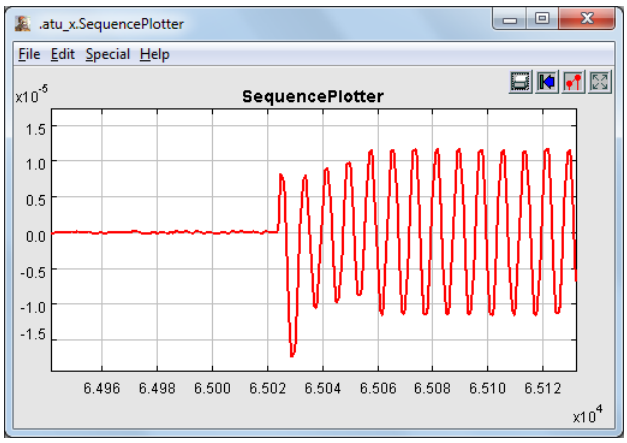


Figura 4.5: Início do envio do sinal C-PILOT1 pelo ATU-C.

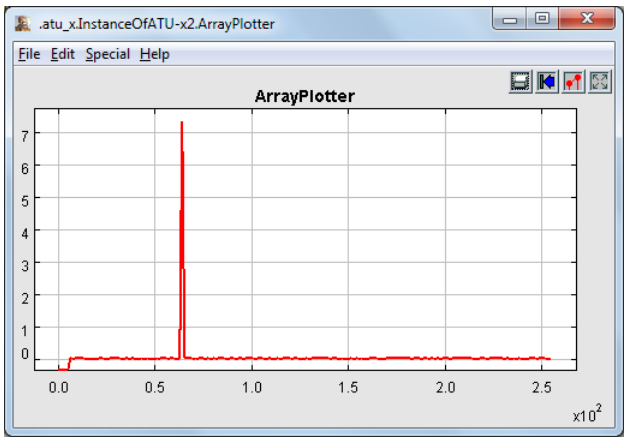
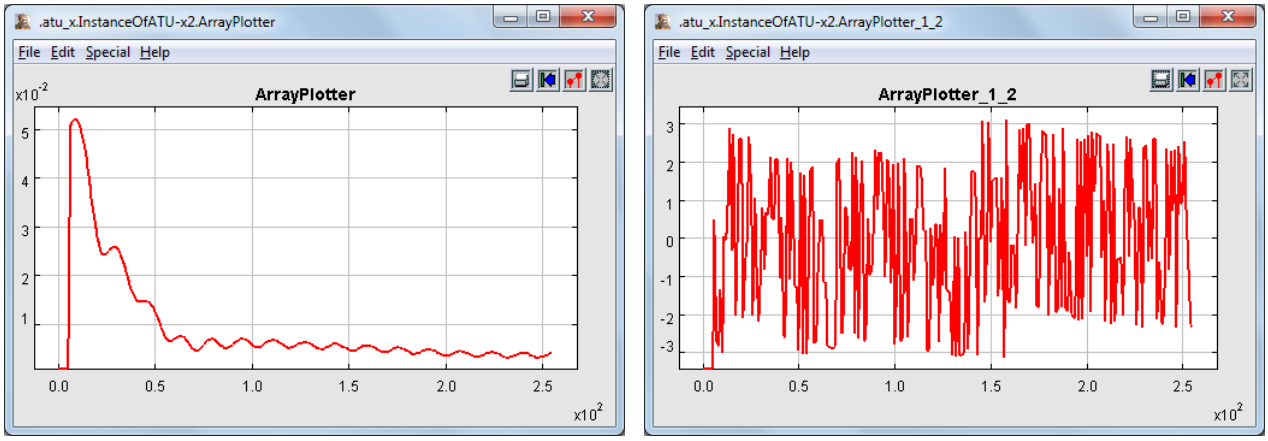


Figura 4.6: Sinal C-PILOT1 no domínio da frequência.

Após o sinal C-PILOT1, o ATU-C começa a enviar o sinal C-REVERB1. As Figuras 4.7(a) e 4.7(b) apresentam a magnitude e a fase, respectivamente, desse sinal na entrada do ATU-R. Notam-se os efeitos de atenuação da potência e de distorção da fase do sinal.

Vários sinais são enviados pelo ATU-C após o sinal C-REVERB1 (ver Tabela 4.1). Como muitos deles são semelhantes entre si (por exemplo, os sinais C-REVERBx, x = 1,2,..., são



(a) (b)

Figura 4.7: Magnitude (a) e fase (b) do sinal C-REVERB1 na entrada do ATU-R.

idênticos, assim como os sinais C-PILOTx, x = 1,2,..., etc), suas plotagens foram omitidas.[2]

Na Figura 4.8 é mostrado, por sua vez, o início do sinal R-REVERB1, enviado pelo ATU-R após detectar o tom piloto que foi enviado pelo ATU-C.

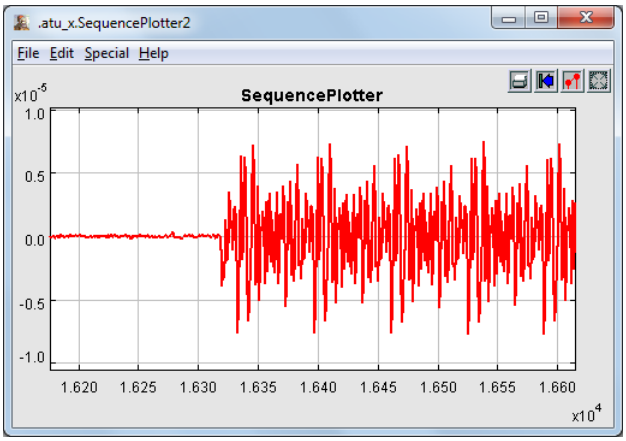
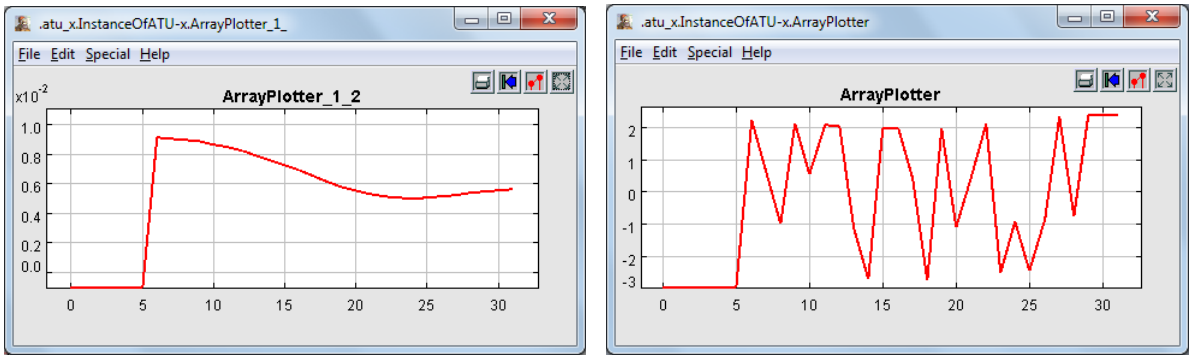


Figura 4.8: Início do envio do sinal R-REVERB1 pelo ATU-R.

Os gráficos de magnitude e fase desse mesmo sinal, obtido na entrada do receptor do ATU-C, são mostrados nas Figuras 4.9(a) e 4.9(b).

Assim como o ATU-C, o ATU-R envia muitos sinais durante a inicialização e vários deles são semelhantes entre si. Por isso, apenas o R-REVERB1 é mostrado aqui.

[2]O sinal C-ECT, reservado para treinamento do cancelador de eco, é opcional e não foi implementado no Hermes, consistindo apenas em silêncio (ou seja, a tensão na saída do modem é nula durante esse sinal).

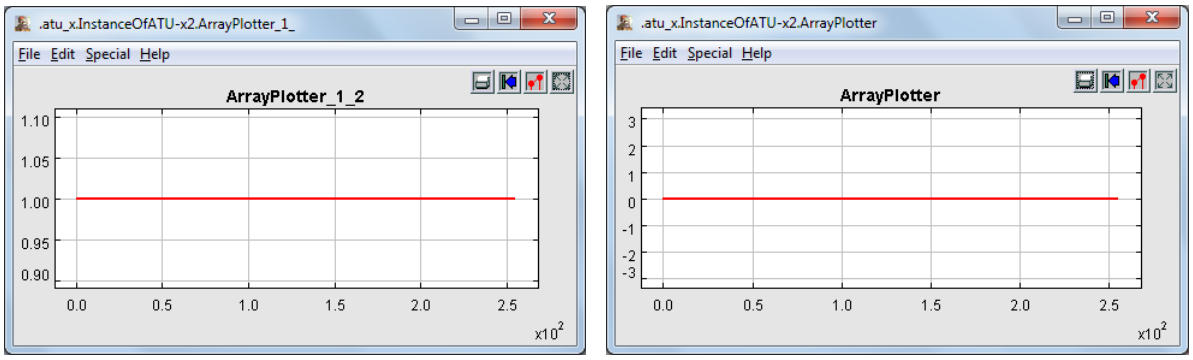


(a) (b)

Figura 4.9: Magnitude (a) e fase (b) do sinal R-REVERB1 na entrada do ATU-C.

4.1.2 Treinamento do FEQ

Como explicado no capítulo 3, durante a fase de treinamento os *taps* do FEQ são estimados de forma contínua, através do algoritmo LMS. Através das Figuras 4.10(a) e 4.10(b), 4.11(a) e 4.11(b), é possível observar a evolução dos *taps* do FEQ do ATU-R, em magnitude e fase, antes e depois dessa estimação.



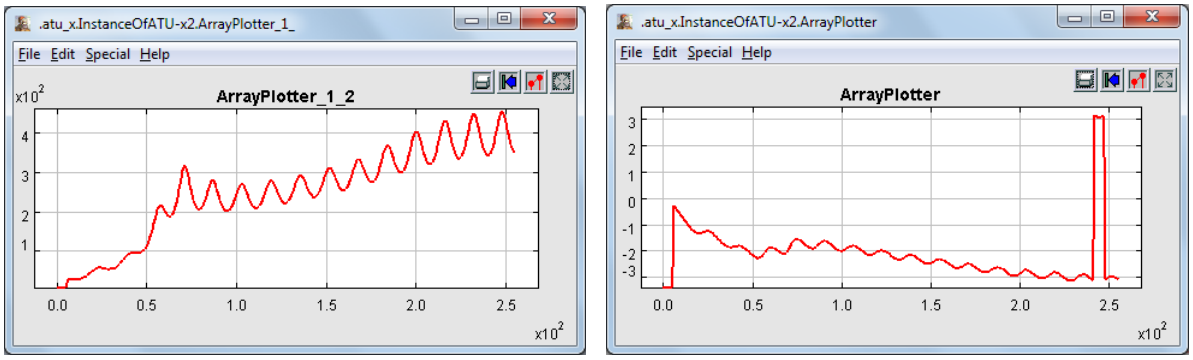
(a) (b)

Figura 4.10: Plotagem de magnitude (a) e fase (b) dos *taps* do FEQ, antes de sua estimação.

A evolução do FEQ do ATU-C ocorreu de modo semelhante e é omitida aqui.

4.2 Simulação *offline* com o Tracespan^{TM}

O Tracespan^{TM}(Figura 4.12) é um equipamento para análise não-intrusiva de redes ADSL e ADSL2+ [2]. Ele é conectado diretamente ao par trançado onde ocorrerá a transmissão, mas



(a) (b)

Figura 4.11: Plotagem de magnitude (a) e fase (b) dos *taps* do FEQ depois de sua estimação.

não é "enxergado" pelos modems, não interferindo, portanto, na qualidade da trasmissão.

Figura 4.12: O equipamento Tracespan^{TM} (foto retirada de [2]).

O Tracespan^{TM} digitaliza o sinal detectado nos pontos em que está conectado e o salva num arquivo binário. O usuário pode determinar um tempo máximo de digitalização (por exemplo, 2 minutos) ou finalizá-la no momento em que achar apropriado.

Depois que a digitalização termina e o arquivo binário contendo o sinal ADSL é salvo, o Tracespan^{TM} inicia uma análise multi-camada desse sinal, permitindo visualizar toda a transmissão em forma de sinal no tempo, símbolos na frequência (constelações QAM em cada tom) ou bits (inclusive divididos em categorias). Dessa forma, é possível verificar quais símbolos, bits ou sinais foram enviados durante cada momento da transmissão; se eles foram enviados de forma errada; se houve algum comportamento anormal de algum dos modems durante a transmissão, e qual a causa desse comportamento; se algum tipo de parâmetro foi modificado; etc.

Além disso, por possuir um receptor ADSL/ADSL2+ internamente, o Tracespan^{TM} possibilita a digitalização do sinal em vários pontos da recepção, como mostrado na Figura

4.13.

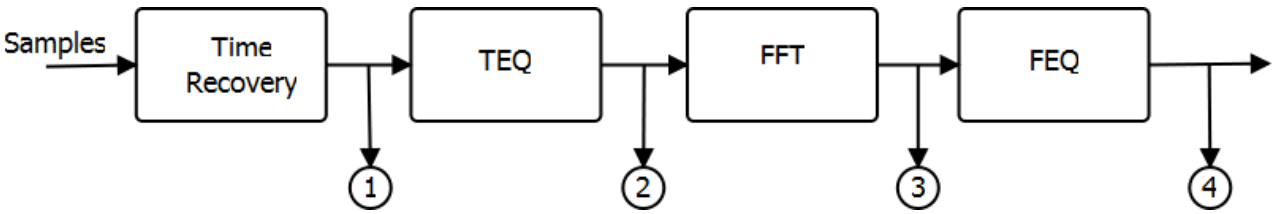


Figura 4.13: Pontos do receptor em que o Tracespan^{TM} permite digitalizar o sinal.

As simulações *offline* foram realizadas após a digitalização, através do Tracespan^{TM} de uma trasmissão real entre um DSLAM e um modem ADSL. Essa transmissão foi realizada no Laboratório de DSL da UFPA (LabDSL), utilizando:

- Um DSLAM modelo EDN312xp;
- Um modem ADSL Speedstream 4200;
- UM cabo telefônico de 2000m.

A Figura 4.14 mostra a configuração utilizada.

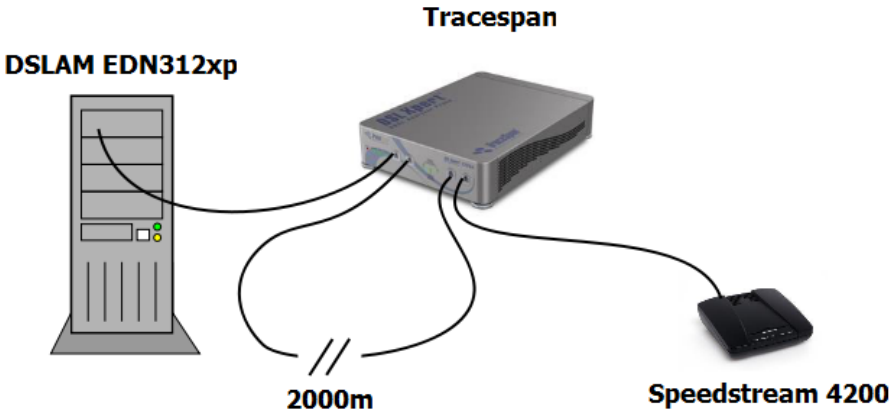


Figura 4.14: Configuração utilizada para digitalizar os sinais de *showtime* de uma transmissão entre um DSLAM e um modem ADSL reais.

Para fins de exemplo, é mostrado na Figura 4.15 o gráfico de magnitude dos primeiros 10000 símbolos DMT (pouco mais de 1 segundo) do sinal de inicialização exportado do Tracespan após o TEQ e levado para o domínio da frequência (isto é, fazendo-o passar por uma FFT).

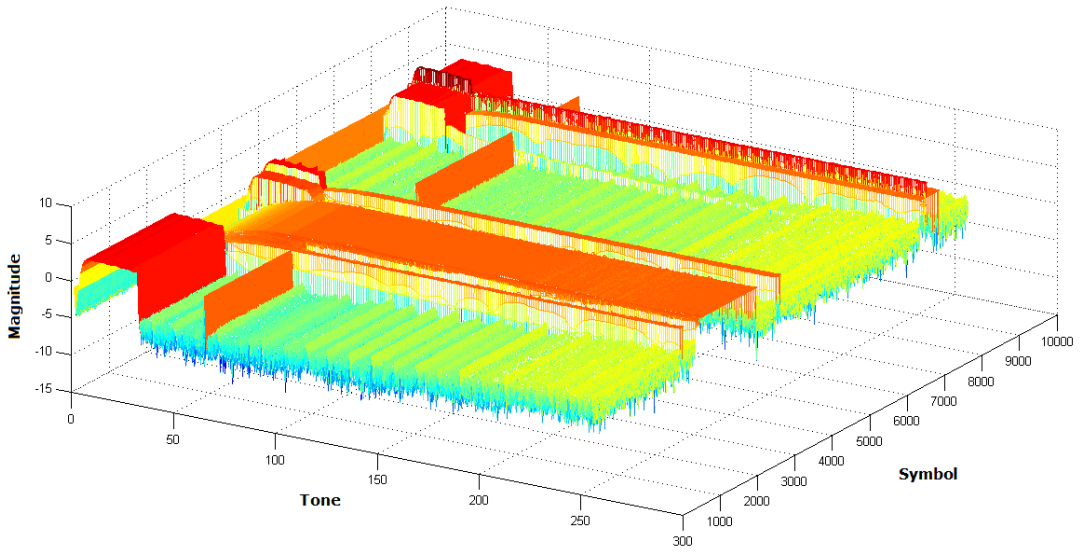


Figura 4.15: Gráfico de magnitude dos primeiros 10000 símbolos DMT do sinal digitalizado pelo Tracespan^{TM} no domínio da frequência.

No gráfico é possível analisar a presença do tom piloto, além de outros sinais de inicialização. Nota-se também que os símbolos na faixa de *upstream* (até o tom 32) têm magnitude maior que os símbolos de *downstream*, o que se explica pelo fato de o sinal ter sido obtido na extremidade da linha mais próxima ao ATU-R - não tendo o sinal de *upstream*, portanto, sofrido atenuação do canal (ao contrário do sinal de *downstream*).

Após a digitalização da transmissão, os dados foram exportados para arquivos binários utilizados para alimentar o Hermes. A Figura 4.16 ilustra os passos para se gerar as simulações com o Tracespan^{TM} e o Hermes.

4.2.1 Inicialização

As mensagens recebidas, as detecções de sinais e os parâmetros decodificados pelo Hermes durante a fase de inicialização estão sumarizados no Anexo A. Comparados com os resultados obtidos pelo próprio Tracespan durante a análise de camadas da transmissão digitalizada, os resultados apresentados pelo Hermes foram rigorosamente semelhantes, o que demonstrou a validade do receptor.

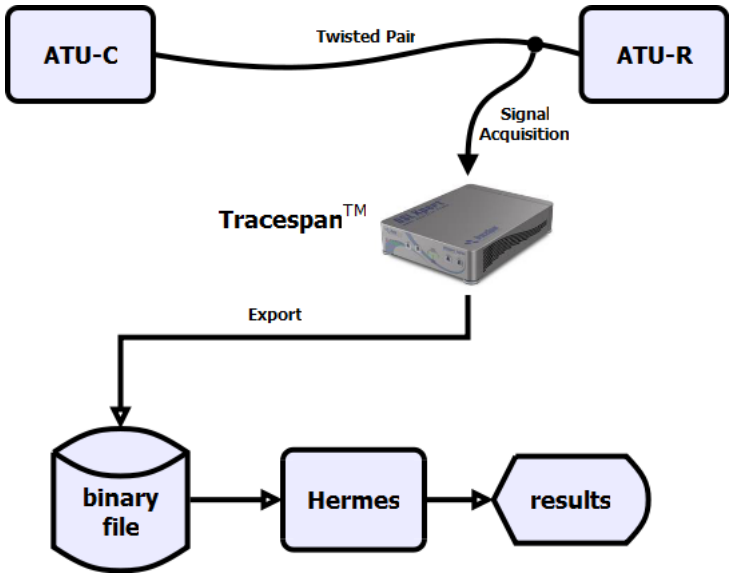


Figura 4.16: Passos para as simulações *offline* com o Tracespan™ e o Hermes.

4.2.2 *Showtime*

O Tracespan™ também possibilitou a execução de simulações *offline* com o Hermes durante a fase de *showtime*. Neste caso, porém, há mais possibilidades de obtenção de sinais digitalizados; além dos pontos em que se pode obter sinais temporais (antes do TEQ ou da FFT) e símbolos na frequência (depois da FFT ou do FEQ), mostrados na Figura 4.13, é possível também exportar sinais lógicos após o *interleaver*, nos pontos mostrados na Figura 4.17.

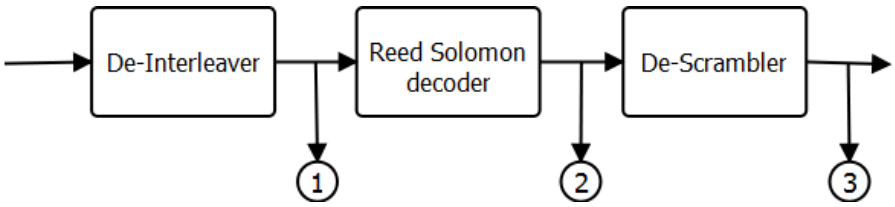


Figura 4.17: Pontos do receptor em que o Tracespan™ permite digitalizar os sinais lógicos, isto é, como quadros de bits.

Pra fins de simplicidade, uma simulação em *showtime* foi executada com o sinal de downstream obtido na saída do FEQ (ponto 4 na Figura 4.13).

Para esta simulação, foi utilizado o modelo da Figura 4.18.

Notam-se alguns pontos de prova nas saídas dos blocos de decodificação, conectados a mostradores (*displays*). Através desses mostradores, foi possível analisar os dados que

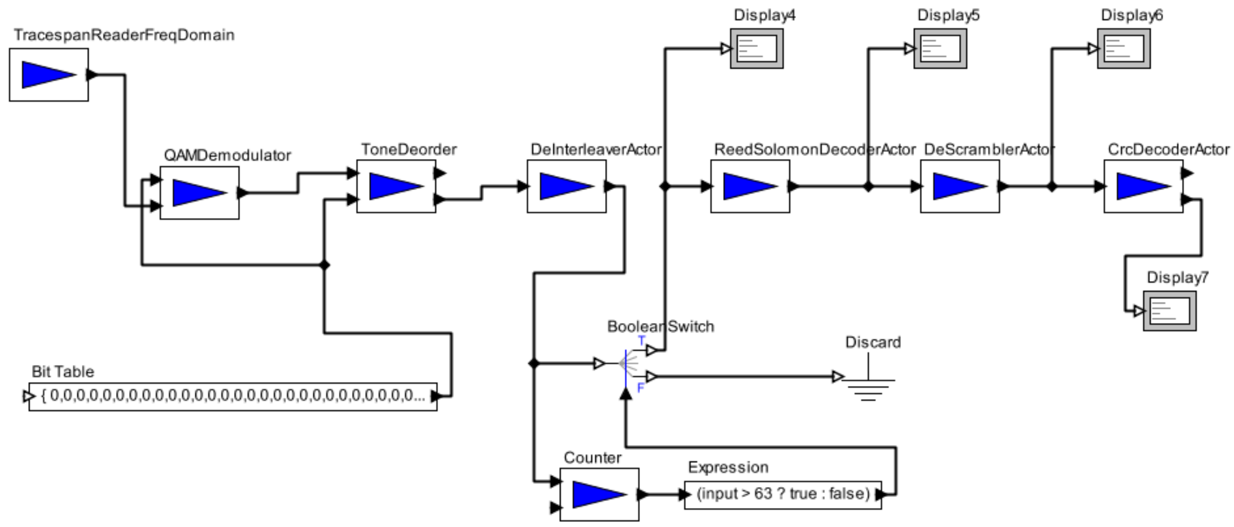


Figura 4.18: Modelo para simulação da fase de *showtime* utilizando os símbolos de *downstream* digitalizados pelo Tracespan^{TM} na saída do FEQ.

passaram por cada um dos blocos e compará-los com os dados digitalizados pelo próprio Tracespan^{TM} nos pontos 1, 2 e 3 da Figura 4.17.

Além disso, pode-se perceber que o vetor "BitTable" parece estar preenchido por zeros. Entretanto, isso se deve ao fato de que, para economizar espaço, o Vergil "trunca" os valores de um vetor quando ele é muito grande pra ser mostrado no modelo. O vetor BitTable tem 256 valores, correspondentes aos tons de *downstream*. Como os primeiros 32 tons de downstream não carregam bits (ou seja, seus valores no vetor BitTable são zero) e apenas os 28 primeiros valores são mostrados, tem-se a impressão de que o vetor inteiro é nulo, o que não é verdade.

Para executar essa simulação, o vetor BitTable foi preenchido com os valores obtidos através dos dados armazenados pelo Tracespan durante a fase de inicialização da mesma transmissão cuja fase de *showtime* foi simulada.

Nas Figuras 4.19, 4.20 e 4.21 são mostradas partes das saídas dos mostradores.

Todos os números correspondem a representações inteiras sem sinal de bytes, ou seja, valores de 0 a 255, obtidos nos pontos de provas de cada *display*. Cada linha corresponde a um quadro. Ao comparar esses valores com os valores digitalizados pelo Tracespan^{TM} verificou-se

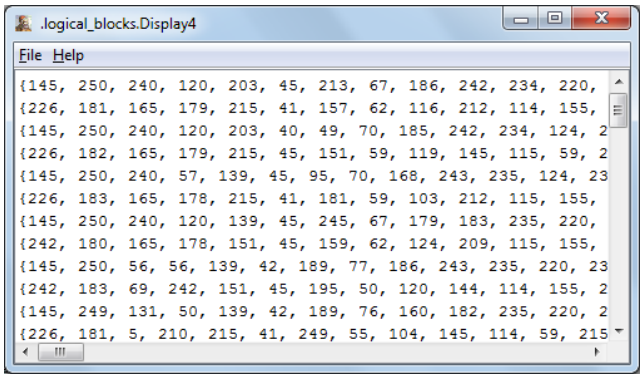


Figura 4.19: Parte dos dados obtidos na saída do bloco DeinterleaverActor durante a simulação de *showtime*.

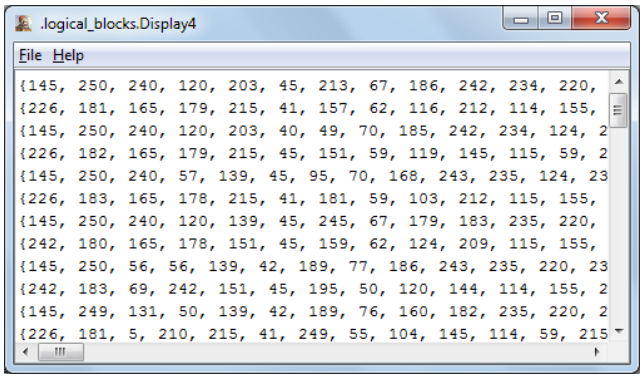


Figura 4.20: Parte dos dados obtidos na saída do bloco ReedSolomonDecoderActor durante a simulação de *showtime*.

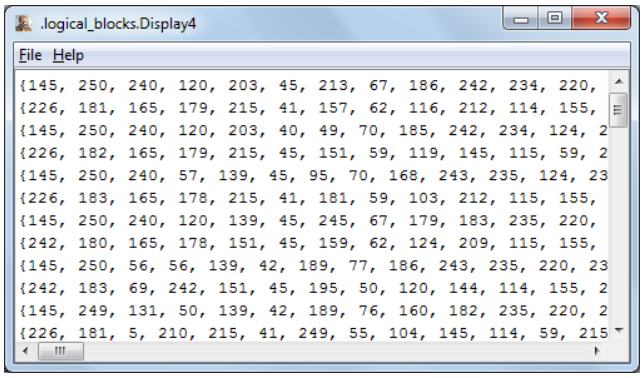


Figura 4.21: Parte dos dados obtidos na saída do bloco DescramblerActor durante a simulação de *showtime*.

que eles são idênticos, validando-se assim o receptor do Hermes durante a fase de *showtime*.

Capítulo 5

Conclusões

Nesta dissertação, foram apresentados os passos do projeto e implementação de um *software*-modem ADSL baseado no *framework* Ptolemy II, chamado de Hermes. Também foi realizada uma análise de seus blocos individuais e de seu funcionamento como um todo.

Simulações e resultados, inclusive com o uso do equipamento Tracespan^{TM} foram mostrados. Através da análise desses resultados, é possível concluir que o Hermes consegue executar todas as etapas de inicialização de um modem real, além de demodular e decodificar corretamente sinais de *showtime*. O uso do Tracespan^{TM} permitiu validar seus blocos, através da digitalização de uma transmissão real e da comparação dos parâmetros obtidos pelos modens reais com os obtidos pelo Hermes.

O Hermes continua em pleno desenvolvimento e os próximos passos relacionados a este trabalho são:

- Otimizar seu desempenho, através da refatoração de seu código e da substituição de funções computacionalmente custosas por funções equivalentes que sejam mais "leves";
- Implementar os blocos opcionais de acordo com a Recomendação ITU-T G.992.1 [1], como, por exemplo, a codificação por treliça;
- Desenvolvimento de um AFE para realizar a comunicação do Hermes com modens reais;
- Implementação do TEQ e do bloco de sincronismo;

- Implementação da fase de sinalização [22];
- Reutilizar o código do Hermes para a construção de um modem VDSL/VDSL2 e, posteriormente, de um modem capaz de alcançar taxa da ordem de Gigabits por segundo (Gbps). Devido à alta taxa de dados, este poderá ter implementadas suas funções mais simples em *software* e as mais complexas em *hardware*, através de tecnologias como VHDL, FPGA, dentre outras;

Referências Bibliográficas

[1] ITU-T, "Asymmetric Digital Subscriber Line (ADSL) transceivers," June 1999.

[2] TraceSpan^{TM}, "Tracespan products," http://www.tracespan.com, último acesso em 28/02/2011.

[3] D. Ginsburg, *Implementing ADSL*, Addison-Wesley, 1999.

[4] C. K. Summers, *ADSL Standards, Implementation and Architecture*, CRC Press, 1999.

[5] W. Goralski, *ADSL & DSL technologies*, Networking Professional's Library. Osborne/McGraw-Hill, 2002.

[6] T. Starr, J. M. Cioffi, and P. J. Silverman, *Understanding Digital Subscriber Line Technology*, Prentice-Hall, 1999.

[7] T. Starr, M. Sorbara, J. M. Cioffi, and P. J. Silverman, *DSL Advances*, Prentice-Hall, 2003.

[8] S.A. Tretter, *Communication System Design Using DSP Algorithms: With Laboratory Experiments for the TMS320C6713 DSK*, Information technology–transmission, processing, and storage. Springer, 2008.

[9] J.W. Cook, R.H. Kirkby, M.G. Booth, K.T. Foster, D.E.A. Clarke, and G. Young, "The noise and crosstalk environment for adsl and vdsl systems," *Communications Magazine, IEEE*, vol. 37, no. 5, pp. 73 –78, May 1999.

[10] C. Zeng and J.M. Cioffi, "Crosstalk cancellation in adsl systems," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, 2001.

[11] J. A. C. Bingham, *The DSL as a Medium for High-Speed Data*, pp. 21–58, John Wiley & Sons, Inc., 2001.

[12] E. Van den Bogaert, T. Bostoen, J. Van Elsen, R. Cendrillon, and M. Moonen, "DSM in practice: Iterative water-filling implemented on ADSL modems," in *Proc. IEEE Int. Conf. on Acoust., Speech and Sig. Processing (ICASSP)*, 2004.

[13] J. Cioffi, A. Chowdhery, H. Zou, and P. Silverman, "Mixed vectored and non-vectored VDSL2," 2010.

[14] J. Cioffi, M. Brady, V. Pourahmad, S. Jagannathan, W. Lee, Y. Kim, C. Chen, K. Seong, D. Yu, M. Ouzzif, H. Mariotte, R. Tarafi France Telecom R, G. Ginis, B. Lee, T. Chung, P. Silverman, and Assia Inc, "Vectored dsls with dsm: The road to ubiquitous gigabit dsls," .

[15] Telecommunications Research Center Vienna (FTW), "FTW's xDSL simulation tool," http://xdsl.ftw.at/xdslsimu/, último acesso em 14/06/2008.

[16] W. Tranter, K. Shanmugan, T. Rappaport, and Kurt Kosbar, *Principles of communication systems simulation with wireless applications*, Prentice Hall Press, Upper Saddle River, NJ, USA, first edition, 2003.

[17] G. Arslan, M. Ding, B. Lu, M. Milosevic, Z. Shen, and B. L. Evans, "Matlab dmtteq toolbox version 3.1," http://www.ece.utexas.edu/~bevans/projects/adsl/dmtteq/dmtteq.html.

[18] F. Bellard, "A generic linux soft modem," http://bellard.org/linmodem.html.

[19] Ptolemy II, "Ptolemy ii project," http://ptolemy.berkeley.edu/ptolemyII/, último acesso em 16/03/2011.

[20] G. C. Guedes, H. Abrahim, A. C. Gomes, A. Klautau, and B. Dortschy, "On the dynamic behavior of margin in dsl: Procedures and tools," *7th International Information and Telecommunication Technologies Symposium*, 2008.

[21] W.M.C. Sansen, J.H. Huijsing, and R.J. Plassche, *Analog circuit design: (X)DSL and other communication systems; RF MOST models; integrated filters and oscillators*, Kluwer Academic Publishers, 1999.

[22] ITU-T, "Handshake," January 2004.

[23] P. Golden, H. Dedieu, and K. Jacobsen, *Implementation and Applications of DSL Technology*, Auerbach Publications, Taylor & Francis Group, 2007.

[24] K.J. Kerpez, D.L. Waring, S. Galli, J. Dixon, and P. Madon, "Advanced dsl management," *Communications Magazine, IEEE*, vol. 41, no. 9, pp. 116 – 123, 2003.

[25] Ptolemy Classic, "Ptolemy classic project," http://ptolemy.berkeley.edu/ptolemyclassic/, último acesso em 16/03/2011.

[26] E. A. Lee, C. Hylands, J. Janneck, J. Davis II, J. Liu, X. Liu, S. Neuendorffer, S. nd Sachs M. Stewart, K. Vissers, and P. Whitaker, "Overview of the ptolemy project," Tech. Rep. UCB/ERL M01/11, EECS Department, University of California, Berkeley, 2001.

[27] Y. Zhou, "Communications systems modeling in ptolemy ii," Tech. Rep. UCB/ERL M03/53, EECS Department, University of California, Berkeley, 2003.

[28] Ptolemy II, "Ptolemy ii documentation," http://ptolemy.eecs.berkeley.edu/ptolemyII/designdoc.htm, último acesso em 16/03/2011.

[29] Agilent Technologies, "Agilent design system," http://www.agilent.com/find/eesof-ads, último acesso em 16/03/2011.

[30] S. Lin and D.J. Costello, *Error control coding: fundamentals and applications*, Pearson-Prentice Hall, 2004.

[31] P. Warrier and B. Kumar, *XDSL architecture*, ITPro collection. McGraw-Hill, 2000.

[32] L. Hanzo, W. Webb, and T. Keller, *Single- and multi-carrier quadrature amplitude modulation: principles and applications for personal communications, WLANs and broadcasting*, John Wiley & Sons, 2000.

[33] B. Fette, R. Aiello, and D. M. Dobkin, *Rf & Wireless Technologies*, Elsevier Science, 2007.

[34] W.C. Wong, R. Steele, and C.E. Sundberg, *Source-matched mobile communications*, Pentech, 1995.

[35] C. Schlegel and L. Perez, *Trellis coding*, IEEE Press, 1997.

[36] A.R.S. Bahai, B.R. Saltzberg, and M. Ergen, *Multi-carrier digital communications: theory and applications of OFDM*, Information technology–transmission, processing, and storage. Springer, 2004.

[37] K. Rao, D.N. Kim, and J.J. Hwang, *Fast Fourier Transform*, Signals and Communication Technology. Springer, 2008.

[38] W. Henkel, G. Tauböck, P. Ödling, P. O. Börjesson, and N. Petersson, "The cyclic prefix of ofdm/dmt - an analysis," 2002.

[39] U. Mengali and A.N. D'Andrea, *Synchronization techniques for digital receivers*, Applications of communications theory. Plenum Press, 1997.

[40] S. Haykin, *Adaptive filter theory*, Prentice-Hall information and system sciences series. Prentice Hall, 2002.

Anexo A

Sumário das mensagens recebidas, detecções de sinais e parâmetros decodificados pelo Hermes durante a fase de inicialização.

Treinamento

```
Waiting for C-PILOT1A...
Detected C-PILOT1A for 3075 symbol periods.
Change to R-REVERB1 state.
Sending 4096 R-REVERB1 symbols...
4096 R-REVERB1 symbols sent.
Change to R-QUIET3
Counting 2048 symbol periods in R-QUIET3 state...
2048 symbol periods counted.
Change to R-ECT state.
Sending 512 R-ECT symbols...
512 R-ECT symbols sent.
Change to R-REVERB2 state.
Sending 1024 R-REVERB2 symbols...
1024 R-REVERB2 symbols sent.
Change to R-SEGUE1 state and wait for C-SEGUE1.
Ok, 10 consecutive C-SEGUE1 symbols detected.
Change to R-REVERB3 state and start reading C-RATES1.
```

Análise de canal

```
Reading C-RATES1...
Preffix = 0x55555555
C-RATES1 preffix is OK.
Option 1:
```

```
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=128 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=16 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=0 S=1 I=1 FS(LS2)=0 RSf=0 RSi=0 S=1 I=1 FS(LS2)=0
Option 2:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=86 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=11 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=0 S=1 I=1 FS(LS2)=0 RSf=0 RSi=0 S=1 I=1 FS(LS2)=0
Option 3:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=44 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=6 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=0 S=1 I=1 FS(LS2)=0 RSf=0 RSi=0 S=1 I=1 FS(LS2)=0
Option 4:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=1 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=1 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=0 S=1 I=1 FS(LS2)=0 RSf=0 RSi=0 S=1 I=1 FS(LS2)=0
Finished decoding C-RATES1. Start receiving C-CRC1.
Reading C-CRC1...
C-CRC1 = 0x1c20
Finished decoding C-CRC1. Start receiving C-MSG1.
Reading C-MSG1...
C-MSG1 fields:
Maximum number of bits per tone: 15
Transmit PSD during initialization: 1
Framing mode: 3
NTR: 1
Overlapped spectrum option: 0
Trellis coding option: 1
Minimum required downstream SNR margin at initialization: 6
Finished decoding C-MSG1. Start receiving C-CRC2.
Reading C-CRC2...
C-CRC2 = 0xec5
Finished decoding C-CRC2. Sending 20 more R-REVERB3 symbols before changing to
 R-SEGUE2 state.
Ok, 20 more R-REVERB3 symbols sent.
Change to R-SEGUE2 state.
Sending R-SEGUE2 for 10 symbol periods.
Ok, 10 R-SEGUE2 symbols sent.
Change to state R-RATES1
```

```
Sending R-RATES1...
R-RATES1 sent. Change to state R-CRC1
Sending R-CRC1...
R-CRC1 sent. Change to state R-MSG1.
Sending R-MSG1...
R-MSG1 sent. Change to state R-CRC2.
Sending R-CRC2...
R-CRC2 sent. Change to state R-MEDLEY.
Sending R-MEDLEY for 16384 symbol periods, while using the received signal (C-MEDLEY)
 to estimate the downstream SNR.
R-MEDLEY sent. Change to state R-REVERB4.
Sending R-REVERB4 for 128 symbol periods...
R-REVERB4 sent. Change to state R-SEGUE3.
Sending R-SEGUE3 for 10 symbol periods...
Ok, 10 R-SEGUE3 symbols sent. Change to state R-MSG-RA.
```

Troca de informações

```
Sending R-MSG-RA...
R-MSG-RA sent. Change to state R-CRC-RA1.
Sending R-CRC-RA1...
R-CRC-RA1 sent. Change to state R-RATES-RA.
Sending R-RATES-RA...
R-RATES-RA sent. Change to state R-CRC-RA2.
Sending R-CRC-RA2...
R-CRC-RA2 sent. Change to state R-REVERB-RA and wait for C-SEGUE2.
Ok, 10 consecutive C-SEGUE2 signal(s) detected. Start receiving C-RATES-RA.
Reading C-RATES-RA...
Option 1:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=128 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=16 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=14 S=1 I=64 FS(LS2)=0 RSf=0 RSi=2 S=8 I=8 FS(LS2)=0
Option 2:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=128 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=16 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=14 S=1 I=64 FS(LS2)=0 RSf=0 RSi=2 S=8 I=8 FS(LS2)=0
Option 3:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
```

```
 BI Field: AS0=128 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=16 LS1(up)=0 LS2(up)=0
 RRSI Field: RSf=0 RSi=14 S=1 I=64 FS(LS2)=0 RSf=0 RSi=2 S=8 I=8 FS(LS2)=0
Option 4:
 BF Field: AS0=0 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=0 LS1(up)=0 LS2(up)=0
 BI Field: AS0=128 AS1=0 AS2=0 AS3=0 LS0=0 LS1=0 LS2=0 LS0(up)=16 LS1(up)=0 LS2(up)=0
RRSI Field: RSf=0 RSi=14 S=1 I=64 FS(LS2)=0 RSf=0 RSi=2 S=8 I=8 FS(LS2)=0
Finished decoding C-RATES-RA. Start receiving C-CRC-RA1.
Reading C-CRC-RA1...
C-CRC-RA1 = 0xb9b5
Finished decoding C-CRC-RA1. Start receiving C-MSG-RA.
Reading C-MSG-RA...
C-MSG-RA fields:
Maximum allowed DS ATU-R noise margin at initialization and in steady state: 6
Minimum required DS ATU-R noise margin in steady state: 58
New minimum required DS ATU-R noise margin at initialization: 6
Finished decoding C-MSG-RA. Start receiving C-CRC-RA2.
Reading C-CRC-RA2...
C-CRC-RA2 = 0x6452
Finished decoding C-CRC-RA2. Sending more 64 R-REVERB-RA symbols before changing to
 R-SEGUE-RA state.
Ok, 64 more R-REVERB-RA symbols sent. Change to R-SEGUE-RA state.
Sending R-SEGUE-RA...
R-SEGUE-RA sent. Change to R-MSG2 state.
Sending R-MSG2...
R-MSG2 sent. Change to R-CRC3 state.
Sending R-CRC3...
R-CRC3 sent. Change to R-RATES2 state.
Sending R-RATES2...
R-RATES2 sent. Change to R-CRC4 state.
Sending R-CRC4...
R-CRC4 sent. Change to R-REVERB5 state and wait for C-SEGUE-RA.
Ok, 10 consecutive C-SEGUE-RA signal(s) were detected. Start receiving C-MSG2.
Reading C-MSG2...
C-MSG2 fields:
Total number of bits supported: 358
Performance margin with selected option: 24
Estimated average loop attenuation: 11
Finished decoding C-MSG2. Start receiving C-CRC3.
```

```
Reading C-CRC3...
C-CRC3 = 0x9d38
Finished decoding C-CRC3. Start receiving C-RATES2.
Reading C-RATES2...
C-RATES2 option: 0x11
Finished decoding C-RATES2. Start receiving C-CRC4.
Reading C-CRC4...
C-CRC4 = 0x801
Finished decoding C-CRC4. Start receiving C-B\&G.
Reading C-B\&G...
Bits and gains table:
Tone: 1    Gain: 0.0        Number of bits: 0
Tone: 2    Gain: 0.0        Number of bits: 0
Tone: 3    Gain: 0.0        Number of bits: 0
Tone: 4    Gain: 0.0        Number of bits: 0
Tone: 5    Gain: 0.0        Number of bits: 0
Tone: 6    Gain: 0.0        Number of bits: 0
Tone: 7    Gain: 0.33        Number of bits: 3
Tone: 8    Gain: 0.33        Number of bits: 4
Tone: 9    Gain: 0.33        Number of bits: 5
Tone: 10    Gain: 0.33        Number of bits: 6
Tone: 11    Gain: 0.31        Number of bits: 6
Tone: 12    Gain: 0.33        Number of bits: 7
Tone: 13    Gain: 0.33        Number of bits: 8
Tone: 14    Gain: 0.32        Number of bits: 7
Tone: 15    Gain: 0.33        Number of bits: 8
Tone: 16    Gain: 0.33        Number of bits: 7
Tone: 17    Gain: 0.33        Number of bits: 7
Tone: 18    Gain: 0.33        Number of bits: 7
Tone: 19    Gain: 0.33        Number of bits: 5
Tone: 20    Gain: 0.33        Number of bits: 6
Tone: 21    Gain: 0.33        Number of bits: 8
Tone: 22    Gain: 0.30        Number of bits: 8
Tone: 23    Gain: 0.31        Number of bits: 8
Tone: 24    Gain: 0.33        Number of bits: 9
Tone: 25    Gain: 0.33        Number of bits: 9
Tone: 26    Gain: 0.31        Number of bits: 8
Tone: 27    Gain: 0.33        Number of bits: 8
```

```
Tone: 28    Gain: 0.33        Number of bits: 7
Tone: 29    Gain: 0.31        Number of bits: 6
Tone: 30    Gain: 0.33        Number of bits: 6
Tone: 31    Gain: 0.33        Number of bits: 6
Finished decoding C-B\&G. Start receiving C-CRC5.
Reading C-CRC5...
C-CRC5 = 0xd921
Finished decoding C-CRC5. Send more 64 R-REVERB5 symbols before changing to R-SEGUE4
 state.
Ok, 64 more R-REVERB5 symbols sent. Change to R-SEGUE4 state.
R-SEGUE4 sent. Change to R-B\&G state.
Sending R-B\&G...
R-B\&G sent. Change to R-CRC5 state.
Sending R-CRC5...
R-CRC5 sent. Change to R-REVERB6 state and wait for C-SEGUE3.
Ok, 10 consecutive C-SEGUE3 signal(s) were detected. Initialization was successful.
 Sending 1895 more R-REVERB6 symbols befores changing to SHOWTIME.
```