FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Implementação de Algoritmos em FPGA para Estimação de Sinal em Sistemas Ópticos Coerentes

Nuno José de Moura Pinto

Tese submetida no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientador: Henrique Salgado

Co-orientador: João Canas Ferreira

Junho de 2009

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2008/2009

A Dissertação intitulada

"IMPLEMENTAÇÃO DE ALGORITMOS EM FPGA PARA ESTIMAÇÃO DE SINAL EM SISTEMAS ÓPTICOS COERENTES"

foi aprovada em provas realizadas em 20/Julho/2009

o júri

Presidente Professor Doutor José Manuel Martins Ferreira
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Henrique Manuel de Castro Faria Salgado
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor João Paulo de Castro Canas Ferreira
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - **NUNO JOSÉ DE MOURA PINTO**

Faculdade de Engenharia da Universidade do Porto

Resumo

Os sistemas ópticos coerentes têm vindo a ganhar relativa importância nas comunicações por fibra óptica. Uma grande vantagem destes sistemas, comparativamente aos sistemas de modulação em intensidade (IM/DD), consiste na possibilidade de transmissão utilizando vários tipos de modulação. Estas tendem a apresentar melhorias significativas na transmissão de sinal, as quais são acompanhadas por uma maior complexidade ao nível do transmissor e do receptor.

Para a detecção coerente são usados no receptor um oscilador local (LO) e uma PLL. Uma forma de ultrapassar a dependência da PLL passa pela detecção coerente através do processamento digital do sinal. Posto isto, é possível implementar algoritmos em sistemas reconfiguráveis onde o processamento é feito em tempo real, permitindo a sincronização da portadora óptica com o LO sem recorrer ao uso de PLLs.

O objectivo deste trabalho consiste na implementação em plataformas FPGA de algoritmos adaptativos com vista à estimação do sinal transmitido através do canal dispersivo, como é o caso da fibra óptica.

Estes algoritmos permitem a compensação da dispersão cromática da fibra, melhorando o sistema em termos de sensibilidade no receptor e taxas de transmissão.

De entre os algoritmos já existentes foram escolhidos os algoritmos CMA *(Constant Modulus Algorithm)* e LMS *(Least Mean Square)* , cuja particularidade consiste na auto-adaptação do seu desenvolvimento às características do sinal recebido, permitindo desta forma a equalização do mesmo.

Para a implementação desses algoritmos é usada a ferramenta de desenvolvimento *System Generator*, cujas funcionalidades potenciam o desenvolvimento dos algoritmos a alto nível. Outra vantagem desta ferramenta de desenvolvimento é a optimização dos recursos da FPGA, permitindo a criação de sistemas eficientes, abstraindo o utilizador dos aspectos lógicos inerentes ao desenvolvimento do projecto.

Para cada algoritmo foram criadas duas configurações, uma sequencial e outra paralela, com o objectivo de avaliar a utilização de recursos, bem como o desempenho de cada uma. Na versão sequencial, a ocupação da FPGA é menor quando comparando com a versão paralela, no entanto, esta última permite ter uma latência 60% menor que a sequencial.

Na versão paralela são obtidas taxas de transmissão de 11 MSímbolos/s, enquanto que na versão sequencial são 4 MSímbolos/s, para a Virtex 5 XC5VLX330T a operar a frequências próximas de 110 MHz.

Foram também implementados compensadores de dispersão para 200 km e 500 km para a fibra óptica SSMF.

Cada equalizador, juntamente com o compensador, foram testados para sistemas de transmissão com modulação 4QAM e 16QAM. Os resultados são bastante promissores, justificando a aposta nesta tecnologia a fim de tornar estes sistemas mais eficientes.

Abstract

The importance of coherent optical systems has been increasing in the field of optical communications. The prime advantage of these systems, in comparison to intensity modulation systems (IM/DD), lies on the ability to transmit data employing different kinds of modulation. As a result, these techniques help to improve significantly the quality of the transmission, at the cost, though, of a higher hardware complexity in the reader and receiver.

To apply coherent detection, receivers often make use of PLL and Local Oscillators (LO). In order to avoid such complex and expensive components, it is possible to apply, in their place, digital signal processing techniques. Such techniques are based on algorithms which are prone to be implemented in hardware platforms, exploring real-time architectures, hence, achieving also a satisfactory synchronization between the optical carrier and LO. These algorithms also compensate for the chromatic dispersion present in this type of communication, fact that contributes to improve the system's sensibility and throughput.

The scope of this work is to accommodate previous designs of these algorithms in reconfigurable FPGA hardware platforms to achieve a good estimation of the transmitted signal through a dispersive channel such as an optical fibre.

The CMA (Constant Modulus Algorithm) and LMS (Least Mean Square) algorithms were selected from various researched techniques. The most relevant characteristic of the picked algorithms lies on their ability to auto-adapt their calculus process to achieve an adequate equalization of the received signal.

The implementation of these algorithms is performed using an high level approach tool named System Generator, which is a MATLAB plugin.

The use of this tool permits an efficient optimization of the FPGA resources, when developing dedicated hardware systems, as it removes from the user concern any logical issues associated to the traditional HDL approach.

The algorithms were implemented using two main approaches: sequential and parallel. Both were implemented with the purpose of evaluating the performance achieved by each one. When comparing to the parallel implementation, the sequential architecture uses less FPGA resources. However, the parallel version performs with a latency which is 60% lower than the latter.

The transmission rates achieved by the parallel and the sequential designs are respectively, 11 MSymbols/s and 4 MSymbols, when operating at 110 MHz frequency in a Virtex 5 XC5VLX330T . The work performed also comprehends the implementation of compensators to neutralize the dispersion regarding the use of the optical fibre SSMF for 200 Km and 500 Km.

Each equalizer, along with the associated compensator, was tested towards transmission modulations of 4QAM and 16QAM. The results revealed to be promising, justifying the emphasis on this technology to make these systems more efficient.

Agradecimentos

Esta tese foi o culminar de um trabalho para a qual contribuíram várias pessoas, directa e indirectamente, ao longo deste semestre.

Em primeiro lugar quero agradecer aos meus pais, José e Ester, pela força e coragem de todos os dias. Uma palavra de apreço para a minha irmã, Tânia, que indirectamente contribuiu para este trabalho.

À minha querida namorada, Marta, agradeço todo o esforço, trabalho e paciência. Sem ti, o caminho a percorrer teria sido mais longo e difícil. Obrigada pela amizade, pelas mensagens de apoio, foram muito muito importantes, assim como tu és para mim.

Quero agradecer também aos meus orientadores, ao Professor Henrique Salgado pela disponibilidade e apoio dado, e ao Professor João Canas Ferreira também pela disponibilidade e sugestões que foram muito importantes nas alturas certas.

Ao Engenheiro Luís Pessoa, um muito obrigado pela disponibilidade, e paciência ao longo deste trabalho, a ajuda dele foi muito importante.

Por último, quero agradecer aos meus colegas e amigos de laboratório pelo companheirismo e ajuda, que foi muito importante pelo que passo a cita-los: Alfredo Moreira, Carlos Resende, João Rodrigues, João Pereira, João Santos, Manuel Reis e Pedro Santos.

Nuno Pinto

"A experiência é algo que se tem,
quando já não se precisa dela..."

Anónimo

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

Lista de Abreviaturas

A/D	*Analog-to-Digital*
APD	*Avalanche Photo Diode*
ASK	*Amplitude Shift Keying*
ASIC	*Application Specific Integrated Circuit*
ASR	*Addressable Shift Register*
BRAM	*Block RAM*
CMA	*Constant Modulus Algorithm*
DSP	*Digital Signal Processing*
FPGA	*Field Programmable Gate Array*
FF	*Feedforward*
IFFT	*Inverse Fast Fourier Transformation*
IM/DD	*Intensity Modulation Direct Detection*
LED	*Light Emitting Diode*
LMS	*Least Mean Square*
LUT	*LookUp Tables*
LO	*Local Oscillator*
NRZ	*Non Return to Zero*
OPLL	*Optical Phase-Locked Loop*
PLL	*Phase-Locked Loop*
PSK	*Phase Shift Keying*
PIN	*diode P-type, Intrinsic, N-type*
QAM	*Quadrature Amplitude Modulation*
QPSK	*Quadrature Phase Shift Keying*
RAM	*Random Access Memory*
SRL	*Shift Register LUT*
SSMF	*Standard Single-Mode optical Fiber*
VCO	*Voltage Controlled Oscillator*

Lista de Símbolos

ω	Frequência angular
ω_1	Frequência do sinal
ω_2	Frequência do LO
D	Coeficiente de dispersão da fibra
λ	Comprimento de onda
z	Distância de transmissão
c	Velocidade da luz
μ	Passo de adaptação dos algoritmos

Capítulo 1

Introdução

Num sistema óptico, tal como o próprio nome indica, a transmissão de informação é feita através de fibra óptica. A fibra óptica é um canal de transmissão que estabelece a comunicação entre o transmissor e o receptor óptico através de ondas electromagnéticas.

O transmissor − também conhecido por fonte de luz ou fonte óptica − é responsável pela conversão do sinal eléctrico em sinal óptico. A fonte óptica é um diodo semicondutor (Laser ou LED) cuja intensidade luminosa é modulada pela intensidade do sinal eléctrico, através da variação da corrente eléctrica injectada neste.

O receptor óptico é constituído por um detector de luz, também designado por fotodetector, que tem a função de converter o sinal óptico recebido da fibra num sinal eléctrico correspondente ao original enviado. Os detectores mais usados são os fotodiodos PIN e APD − *Avalanche Photo Diode*.

Como qualquer canal, a fibra óptica também introduz penalidades, nomeadamente a dispersão cromática e a dispersão de modo de polarização. A transmissão de impulsos de informação em banda-base em sistemas com modulação de intensidade e detecção directa (IM/DD), combinada com uma largura espectral finita da fonte óptica, a dispersão da fibra vai provocar o alargamento desses impulsos, visto que diferentes frequências propagam-se a velocidades diferentes.

Ao passar para sistemas ópticos coerentes outras modulações mais complexas (como QAM, QPSK) podem ser implementadas [9]. Porém, a detecção coerente do sinal vai impor ao sistema complexidade e custo, visto ser necessário o uso de uma PLL − *Phase Lock-Loop* − óptica para sincronizar a frequência do Oscilador Local (LO) com a portadora óptica do sinal.

Para ultrapassar esta limitação foi estudada, recentemente, a possibilidade de converter e processar o sinal no domínio digital em sistemas ópticos coerentes. Desta forma é possível estimar a fase do LO evitando o uso de PLLs e lasers de largura espectral reduzida, o que simplifica bastante a estrutura do receptor óptico [10]. É neste contexto que surge a necessidade de implementar algoritmos de estimação de fase usando FPGAs, pois no campo das FPGAs o processamento de sinal é exequível em tempo real [11].

Têm sido realizados vários estudos e investigação nesta área, mas esta técnica é ainda bastante recente. Prevê-se que esta tecnologia esteja disponível em sistemas comerciais a curto/médio prazo (5 a 10 anos).

1.1 Motivação

Nos sistemas ópticos IM/DD usa-se apenas uma modulação, que é a de intensidade de luz, onde o bit "1" é representado por presença de luz e o "0" por ausência da mesma. Nos sistemas ópticos coerentes existe uma variedade de modulações que podem ser usadas: por exemplo, a modulação em fase (PSK), ou a modulação em amplitude (ASK), ou ainda, a modulação em fase e quadratura (QPSK ou QAM).

As vantagens do uso de sistemas ópticos coerentes incidem sobretudo nesta variedade de modulações que se tem ao dispor para a transmissão de informação, traduzindo-se num aumento da sensibilidade do receptor [12] e eficiência espectral e, portanto, maiores capacidades de transmissão.

No entanto, uma das desvantagens da detecção coerente é a necessidade duma PLL Óptica (OPLL). Esta OPLL provoca um aumento do grau de complexidade do sistema que, por sua vez, se traduz num acréscimo de custos inviabilizando qualquer aplicação prática.

Actualmente, as investigações que estão em curso para ultrapassar esta situação, indicam que amostrando o sinal usando conversores A/D de alta velocidade para posterior processamento do sinal no domínio digital, é possível compensar penalidades que o sinal sofre ao longo do sistema de transmissão.

É possível utilizar DSP − Processamento Digital de Sinal − para estimação de fase da portadora e, simultaneamente, compensação da dispersão cromática da fibra. A estimação de fase da portadora é essencial na detecção coerente para sincronizar o sinal recebido com o sinal do LO. Para além de ser possível estimar a fase da portadora é ainda possível compensar a dispersão introduzida pela fibra através do uso de algoritmos adequados [10]. Por este motivo, a eficiência no receptor, o alcance e/ou a taxa de transmissão podem ser alargados.

1.2 Objectivos da tese de dissertação

A implementação destes algoritmos pode ser realizada através do uso de FPGAs e em tempo real [11]. Assim, o propósito principal deste trabalho é a implementação em FPGAs de algoritmos já desenvolvidos para estimação do sinal transmitido, quando este atravessa um canal dispersivo (fibra óptica), em sistemas ópticos coerentes.

Seguem-se os objectivos desta tese de dissertação de mestrado:

- Avaliação da melhor plataforma (Verilog, *System Generator*) de implementação em FPGA através do estudo de tutoriais;
- Estudar *scripts* MATLAB já existentes por forma a gerar o sinal a tratar pela FPGA;

- Implementar o algoritmo proposto;
- Avaliar o desempenho do sistema, bem como os limites de funcionamento;
- Especificar uma plataforma FPGA adequada para a tarefa proposta, tendo em conta a velocidade de processamento pretendida.

A secção seguinte descreve a organização deste documento, seguindo-se as contribuições.

1.3 Estrutura da tese de dissertação

Após esta introdução ao tema, enquadramento dos sistemas ópticos coerentes, motivação e objectivos do projecto apresenta-se no capítulo 2 que se segue, o estado da arte em Sistemas Ópticos Coerentes e detecção coerente usando DSP. Nesse capítulo é também abordada a equalização adaptativa descrevendo-se os algoritmos CMA e LMS a implementar em *System Generator*. No final do capítulo 2 é discutida a compensação da dispersão cromática da fibra óptica.

A descrição da arquitectura de FPGAs é feita no capítulo 3, juntamente com as funcionalidades da ferramenta *System Generator*. São aqui apresentadas considerações a ter em linha de conta na implementação de algoritmos em FPGA. No final do capítulo é referida a funcionalidade de co-simulação em hardware do *System Generator*, funcionalidade esta que vai permitir a implementação na FPGA do hardware desenvolvido.

Seguidamente são apresentadas as etapas de implementação dos algoritmos CMA e LMS no capítulo 4. São descritas duas configurações para cada algoritmo, bem como, o módulo desenvolvido para compensar a dispersão. A concluir este capítulo surge a implementação do conceito paralelismo nos algoritmos.

Os capítulos 5 e 6 apresentam os resultados obtidos e as conclusões finais, respectivamente. No capítulo 6 é também apresentado o trabalho futuro.

1.4 Contribuições

- Implementação de algoritmos em FPGA para detecção de fase e compensação da dispersão material da fibra óptica.
- Desenvolvimento, validação e implementação de duas versões para cada algoritmo: configuração sequencial e configuração paralela.
- Na configuração sequencial, uma taxa de transmissão de 4 MSímbolos/s é conseguida.
- Para a configuração paralela, o atractivo valor de 11 MSímbolos/s é conseguido para a taxa de transmissão nesta configuração.
- Implementação de algoritmos em paralelismo, com vista a melhorar os limites de funcionamento do sistema.

- Apresentação de um artigo no Simpósio Nacional: *Symposium on Enabling Optical Networks*, Nokia Siemens Networks Portugal, 26 Junho, 2009.

Capítulo 2

Equalização adaptativa em sistemas ópticos coerentes

2.1 Introdução

Neste capítulo é apresentado o estado da arte de sistemas ópticos coerentes, cuja principal vantagem é poder-se usar vários tipos de modulações. O uso dessas modulações permite obter óptimos níveis de sensibilidade no receptor, o que possibilita o aumento do débito binário e/ou da distância a transmitir, ou seja, a capacidade dum sistema óptico é medida através da taxa de transmissão $\times$ distância.

Este capítulo começa por descrever os elementos de um sistema óptico de Modulação em Intensidade e Detecção Directa (IM/DD). De seguida, é apresentado o conceito de sistemas ópticos coerentes onde é descrita a transmissão e os tipos de recepção possíveis com estes sistemas. Posteriormente, exibe-se as diferenças entre a detecção heterodina e a detecção homodina.

Na secção 2.4 é apresentada a detecção coerente usando DSP. É descrito o receptor e os componentes que o constituem.

A Equalização adaptativa é abordada na secção 2.5 onde são descritos os algoritmos CMA e LMS.

E por fim, na última secção expõem-se as técnicas de compensação da dispersão.

2.2 Sistemas ópticos IM/DD

Os elementos básicos que constituem um sistema óptico são o transmissor, a fibra óptica e o receptor, como ilustra a Figura 2.1 que se segue.

O transmissor ou fonte de luz (à esquerda na Figura 2.1) é responsável pela conversão do sinal eléctrico em sinal óptico. A fonte de luz é um diodo semicondutor ou díodo LED *(Light Emitting Diode)* cuja intensidade luminosa emitida é modulada pelo sinal eléctrico através da variação da

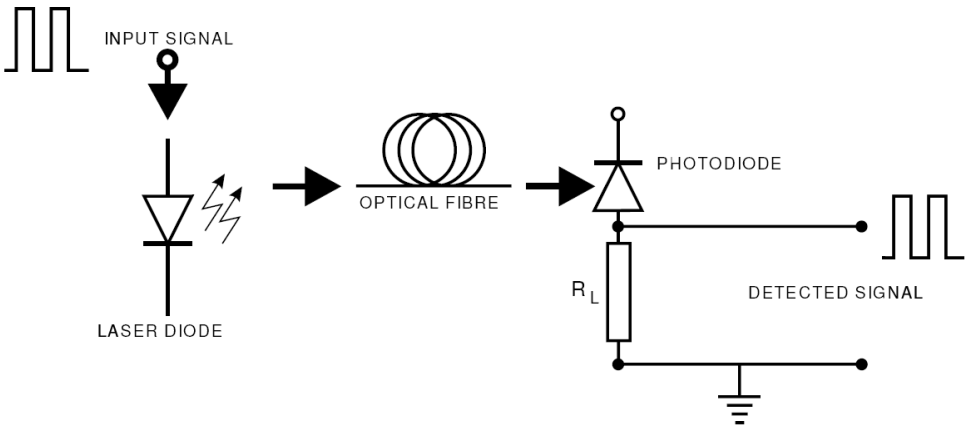


Figura 2.1: Elementos constituintes de um sistema óptico IM/DD [1].

corrente eléctrica injectada no semicondutor. A conversão referida é, habitualmente, denominada de modulação em intensidade para posterior detecção directa (conversão óptico-eléctrica por intermédio dum fotodiodo) em que a corrente gerada no semicondutor é directamente proporcional à potência óptica.

Depois da conversão do sinal eléctrico em sinal óptico, é feito o envio da informação através do canal de transmissão que é a fibra óptica. Como qualquer canal a fibra também introduz penalidades, nomeadamente, a dispersão cromática e a dispersão de modo de polarização. A dispersão da fibra combinada com uma largura espectral finita da fonte óptica vai provocar, na transmissão banda-base de impulsos de informação nestes sistemas, o alargamento desses mesmos impulsos, visto que diferentes frequências se vão propagar a velocidades diferentes. Este fenómeno vai gerar a interferência intersimbólica, aspecto conhecido nas telecomunicações e que é importante evitá-lo.

O receptor óptico (à direita na Figura 2.1) é um detector de luz que tem a função de receber a informação transmitida através da fibra óptica. Outra função do receptor é converter o sinal óptico proveniente da fibra num sinal eléctrico com características próximas do sinal originalmente enviado pelo transmissor. Os detectores mais usados são os fotodiodos PIN e APD. Nos receptores de detecção directa (dos sistemas IM/DD) são usados estes fotodiodos, no entanto a sensibilidade destes receptores, principalmente dos que usam fotodiodos PIN, é baixa o que limita o sistema.

Surgem, então, os sistemas ópticos com recepção coerente em que se conseguem melhorias significativas a nível da sensibilidade do receptor e consequente aumento da taxa de transmissão e da distância entre repetidores.

2.3 Sistemas ópticos coerentes

Os sistemas ópticos coerentes são sistemas que permitem efectuar a transmissão com grande flexibilidade, uma vez que fazem o uso de modulações mais complexas que se traduzem numa maior sensibilidade no receptor e um aumento da eficiência espectral. Para tal, é necessário realizar a detecção coerente sendo este método considerado, o mais avançado em detecção, onde o receptor

calcula a decisão do símbolo enviado baseando-se na informação contida na totalidade do campo eléctrico, incluindo a informação na amplitude e na fase.

2.3.1 Transmissão

Com os sistemas ópticos coerentes a informação pode ser codificada nas componentes da portadora [9]. Para tal, várias modulações podem ser usadas na codificação, tais como:

- a modulação ASK onde a informação é modulada na amplitude da portadora;
- a modulação PSK que usa a fase da portadora para codificar a informação;
- a modulação QPSK ou QAM que usa as duas componentes da portadora em simultâneo (amplitude e fase) para a transmissão dos dados.

Da constituição dum sistema óptico coerente, como no caso dos sistemas IM/DD, fazem parte o transmissor, a fibra óptica e o receptor. Ao nível do transmissor e do receptor a complexidade aumenta, todavia, esta complexidade é compensada por uma maior eficiência espectral e sensibilidade na recepção.

No transmissor é necessário mais um componente (como se pode ver pela Figura 2.2) para a realização das modulações referidas acima. O modulador normalmente usado é o *Mach-Zender*, pois só desta forma é possível a modulação em fase.

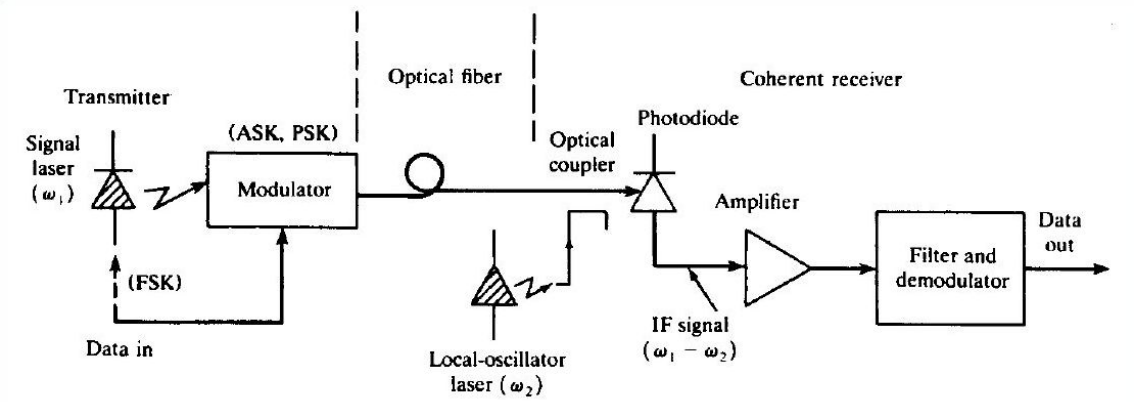


Figura 2.2: Elementos constituintes dum sistema óptico coerente [2].

2.3.2 Recepção

No receptor, além do fotodiodo, surgem mais componentes, como é o caso da PLL. Ao sinal óptico é adicionado um outro sinal proveniente do LO. A função da PLL é sincronizar a fase do LO com a fase da portadora óptica para posterior detecção e desmodulação. Dependendo se o oscilador opera à mesma frequência da portadora óptica ou a uma frequência diferente, tem-se dois tipos de receptores ópticos coerentes: homodinos ou heterodinos, respectivamente [2] assim classificados.

A detecção coerente exige ao receptor o conhecimento da fase da portadora e da frequência do LO que serve como referência da fase absoluta, como é observado na Figura 2.2.

Tradicionalmente, a sincronização da portadora é realizada por uma PLL sendo um sistema que usa o sinal da malha de realimentação para a sincronização.

Em sistemas ópticos coerentes podem ser usadas, ou uma PLL Óptica (OPLL) que sincroniza a frequência e a fase do laser do LO com o laser do transmissor, ou uma PLL eléctrica onde, no receptor é usado um laser como LO operando em modo *free-running* seguido de um segundo nível de desmodulação para o sinal eléctrico analógico ou digital através de um VCO, cuja frequência e fase estão sincronizadas [9].

2.3.2.1 Detecção Heterodina

Na Figura 2.3 que se segue é apresentado um receptor coerente heterodino.

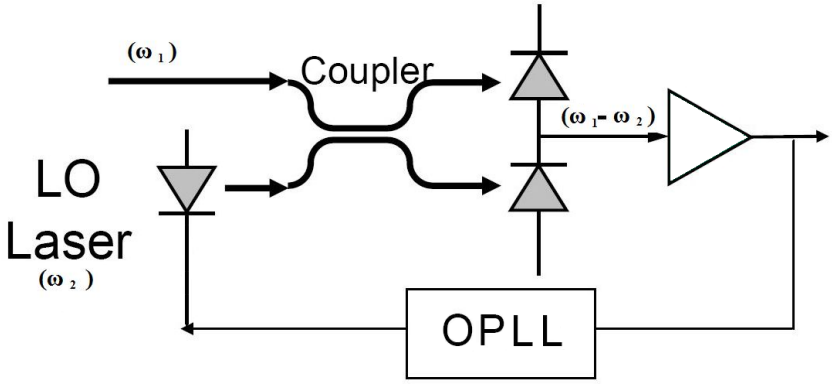


Figura 2.3: Receptor coerente heterodino [3].

Neste receptor, o sinal óptico recebido com frequência ω_1 é combinado com o sinal proveniente do laser LO com frequência ω_2 diferente da anterior e é, posteriormente, aplicado a um fotodetector.

Este fotodetector desempenha o papel dum misturador, convertendo o batimento dos dois sinais ópticos num sinal eléctrico, com uma frequência central (frequência intermédia) igual à diferença entre a frequência dos dois sinais ópticos ($\omega_1 - \omega_1$).

2.3.2.2 Detecção Homodina

A detecção homodina é um caso particular da detecção heterodina, pois neste caso o receptor tem o laser do LO a funcionar à frequência da portadora recebida. A Figura 2.4 ilustra isso mesmo.

Quer a detecção heterodina quer a detecção homodina apresentam grandes melhorias relativamente à detecção directa, principalmente na sensibilidade no receptor, na ordem dos 20 dB como se pode ler na Figura 2.5, quando comparados com detecção directa. Portanto, perante a Figura 2.5 pode-se observar que a sensibilidade depende da modulação praticada num sistema e dos componentes usados.

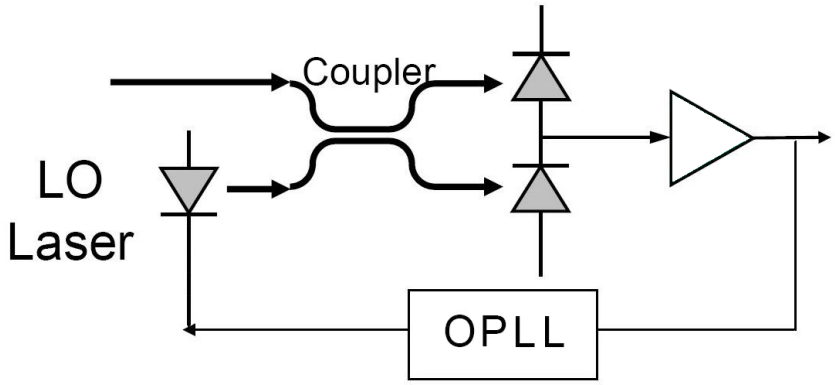


Figura 2.4: Receptor coerente homodino [3].

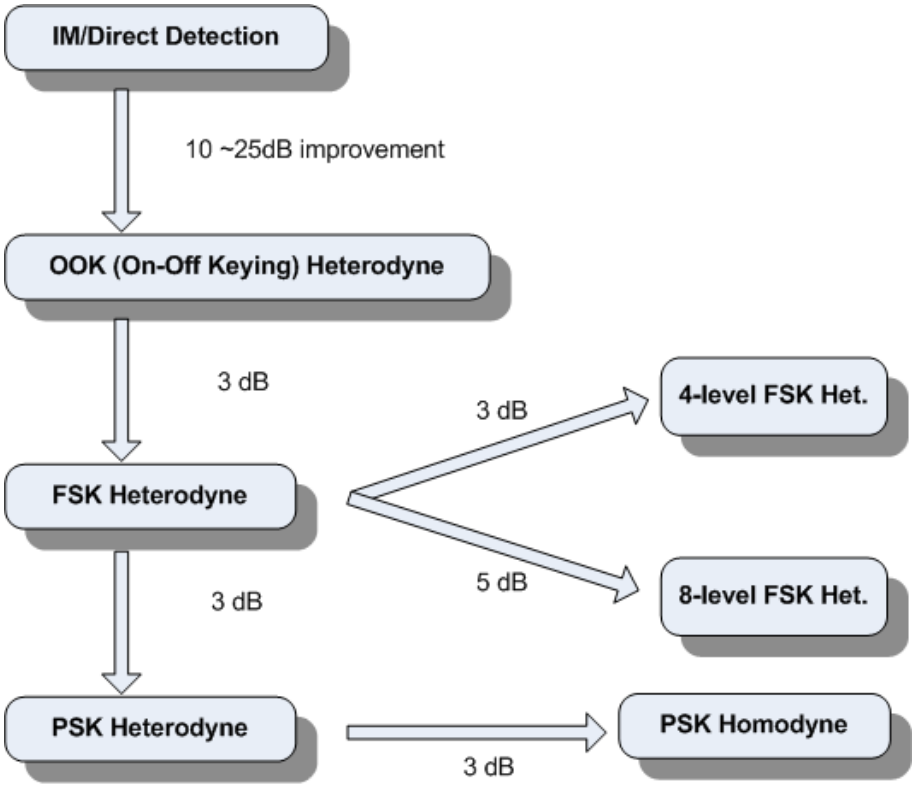


Figura 2.5: Melhorias na sensibilidade do receptor [4].

A utilização de uma PLL eléctrica torna o sistema mais acessível em termos de custo do que usando uma OPLL. Contudo, o uso de PLLs traz problemas para o sistema, pois estas são muito sensíveis ao atraso que se propaga na malha de realimentação, fazendo com que não haja perfeita sintonia entre o sinal do LO e o da portadora.

Como o uso de PLLs (tanto eléctrica, como óptica) acarreta complexidade ao sistema de recepção, facto esse que se traduz em custos e dificuldades de implementação prática, houve a necessidade de desenvolver outros sistemas de detecção coerente óptica, totalmente independentes do seu uso.

Surgem, então, novos métodos para detecção coerente usando Processamento Digital de Sinal (DSP).

2.4 Detecção Coerente Usando DSP

Estas novas técnicas, ultimamente estudadas, visam aproveitar uma tecnologia já conhecida, sistemas coerentes, e realizar a detecção coerente sem a utilização de PLLs [11]. Com a ajuda de FPGAs, combinadas com conversores A/D de elevado desempenho (>40 GSample/s), é possível implementar estas técnicas DSP em tempo real.

Estes métodos podem fazer a sincronização da portadora por *Feedforward* (FF) que é uma forma de ultrapassar o uso de PLLs e o problema do atraso que estas colocam. Além disso, como o sincronizador FF usa tanto os símbolos anteriores como os seguintes para estimar a fase da portadora, atinge-se melhor desempenho do que com uma PLL, uma vez que a PLL é um sistema de *feedback* que usa somente os símbolos anteriores. Recentemente, o processamento digital de sinal permitiu o alinhamento da polarização e sincronização da portadora [9], isto realizado em software.

A Figura 2.6 apresenta o esquema de um sistema óptico coerente que usa processamento digital de sinal.

Como é observado na Figura 2.6, o sistema é constituído pelo transmissor, pelo canal (fibra óptica) e pelo receptor.

Neste trabalho, o transmissor e o canal são simulados por um modelo teórico que se aproxima bastante dos valores na prática. Segue-se depois o receptor.

É no receptor que vai incidir o trabalho de implementação em FPGAs de algoritmos para estimação do sinal transmitido, realizando a detecção, a conversão A/D e depois o tratamento da informação por parte da FPGA.

Receptor

O receptor (Figura 2.6), é constituído por um oscilador local (LO) e um *Optical Hybrid*. Segue-se um par de fotodetectores balanceados que encaminham a informação para os conversores A/D. Com isto vai ser possível colocar os dados no domínio digital para, assim, serem processados pela FPGA.

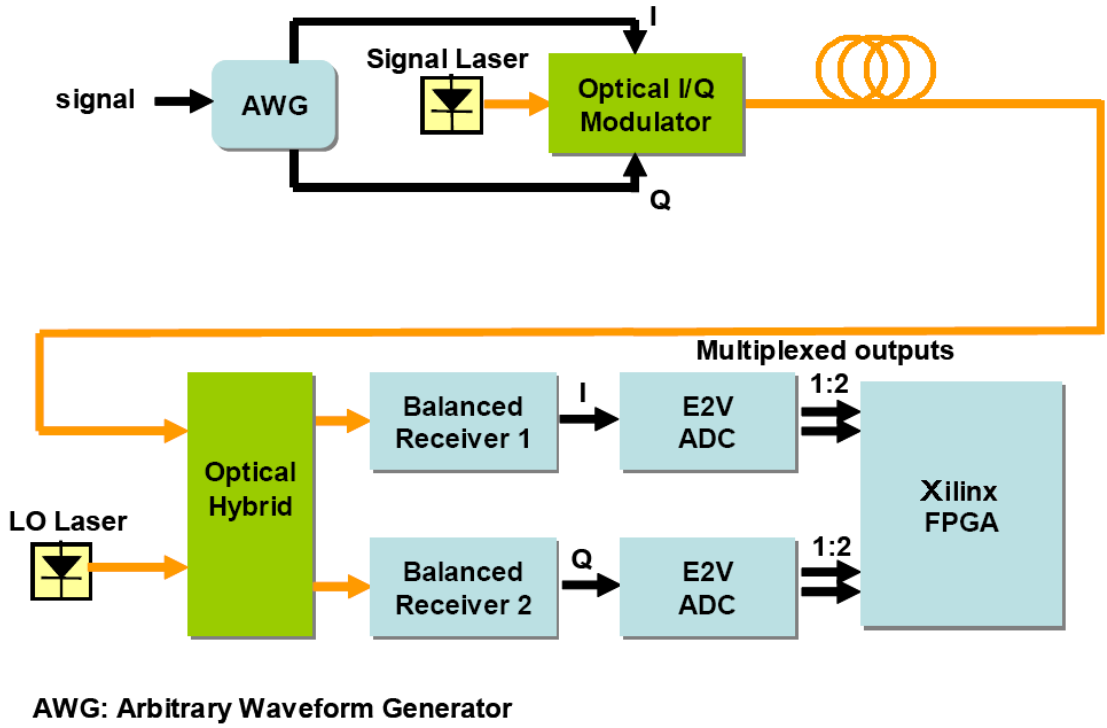


Figura 2.6: Esquema de um sistema óptico coerente usando DSP [5].

Oscilador local

O LO, nestes sistemas, vai estar em funcionamento livre com uma frequência próxima da frequência do sinal coerente. Através do uso de técnicas de estimação de fase é possível realizar a sincronização de fase do oscilador local com a da portador óptica no domínio digital, evitando desta forma a complexidade associada ao uso de uma OPLL [10].

O sinal do LO vai ser entregue ao híbrido óptico, assim como o sinal proveniente da fibra óptica.

Híbrido óptico

O híbrido óptico vai usar o sinal e LO para realizar a separação do sinal óptico nas suas componentes em fase e quadratura, entregando cada uma destas componentes a um receptor balanceado.

Nos sistemas coerentes com multiplexagem de polarização em que dois sinais são multiplexados em polarizações cruzadas, o componente híbrido óptico 90° é importante para a compensação da dispersão do modo de polarização. Isto é possível com o esquema da Figura 2.7, onde se usam dois destes componentes.

Este dispositivo vai realizar a mistura do sinal recebido com o sinal de referência (do LO) e fazer a separação, no domínio complexo para quatro portas de saída (no anterior era só para duas portas de saída), das componentes I e Q de cada duma das duas polarizações cruzadas (Figura 2.7).

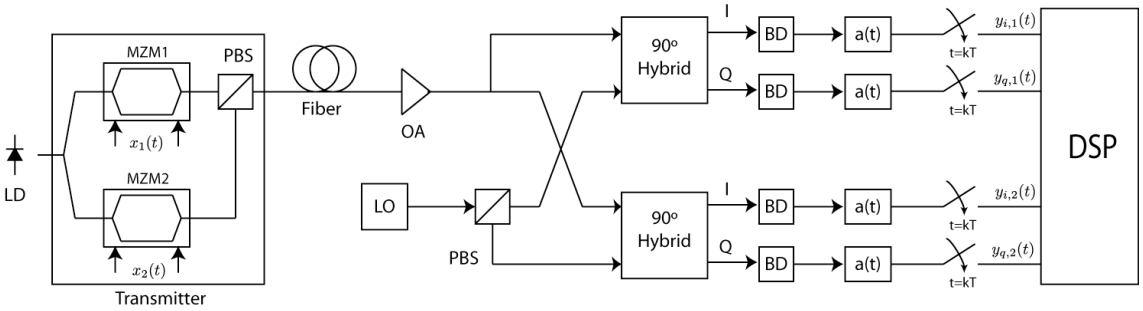


Figura 2.7: Esquema de um sistema que permite a compensação da dispersão do modo de polarização em cada componente [6].

O híbrido óptico entrega os quatro sinais luminosos de forma equilibrada aos dois pares de detectores balanceados, onde cada par representa uma polarização diferente. Desta forma, pode-se realizar a detecção e compensar cada uma das polarizações ortogonais individualmente, através dos algoritmos que serão apresentados na secção seguinte.

Para tal, a detecção usada é a detecção homodina, porque o receptor não usa frequência intermédia.

Conversores A/D

Os sinais detectados pelos fotodetectores (e consequente passagem para o domínio eléctrico) são entregues aos conversores A/D. A conversão A/D é um aspecto muito importante nestes sistemas. Porém, é uma área onde ainda há bastante que explorar, pois para débitos de 10 Gbit/s será preciso um conversor A/D com uma taxa de amostragem de 20 GSample/s para originar duas amostras por símbolo que é a taxa de *Nyquist* [10].

A função dos conversores, para além de originar duas amostras por símbolo, é transformar o sinal eléctrico num digital por forma a que possa ser entregue à FPGA para que este possa ser estimado pelos algoritmos.

FPGA

No trabalho em questão, a FPGA *(Field Programmable Gate Array* e que no capítulo 3 será mais aprofundada) recebe os dados provenientes de modelos teóricos e não de sistemas físicos, o que permite trabalhar *offline*. Na prática o conversor A/D e o processamento da informação terá que ser feito em tempo real. No entanto, este modelo simplificado permite avaliar o desempenho e aperfeiçoar a técnica de implementação dos algoritmos.

Contudo, há já estudos e testes com implementações em tempo real, como demonstrados em [12] e em [11], onde se evidenciam experiências com excelentes taxas de transmissão de 4.4 Gbits/s e 10 Gbits/s, respectivamente.

A FPGA é o dispositivo DSP do receptor onde vão ser implementados os algoritmos, isto é, é esse dispositivo que irá realizar a equalização do sinal. A equalização pode ser classificada em equalização adaptativa e não adaptativa.

2.5 Equalização adaptativa

Equalizador adaptativo é um filtro que se adapta automaticamente através de algoritmos, compensando distorções que o canal de transmissão impõe nos símbolos provenientes do transmissor. Quanto ao tipo de algoritmos existem algoritmos que se auto-adaptam, como é o caso do algoritmo CMA *(Constant Modulus Algorithm)*, e existem outros que são supervisionados necessitando de uma sequência de treino, como é o caso do algoritmo LMS *(Least Mean Square)*.

2.5.1 Algoritmo CMA *(Constant Modulus Algorithm)*

O algoritmo CMA realiza uma equalização cega e foi estudado por Sato em 1975 e mais tarde por Godard em 1980. O modo de funcionamento deste algoritmo é atingir a convergência perante aplicações com evolvente constante, de onde provém o seu nome [13].

É apresentada a estimação dos dados à saída do equalizador, neste algoritmo que usa métodos estocásticos de gradiente descendente para minimizar a função custo [14]. A velocidade de convergência é dada pelo passo de adaptação μ, usando ainda na actualização dos novos coeficientes uma constante R_2 que depende das características do sinal que vai estimar.

O CMA é o mais utilizado em equalização adaptativa devido à baixa complexidade computacional que impõe num sistema. Este procede à sua auto-adaptação, convergindo para um módulo constante independentemente da fase da portadora do sinal. Na Figura 2.8 é visível a sua forma de funcionamento onde se pode observar que não depende de qualquer sequência de treino, como acontece no algoritmo LMS explicado mais à frente.

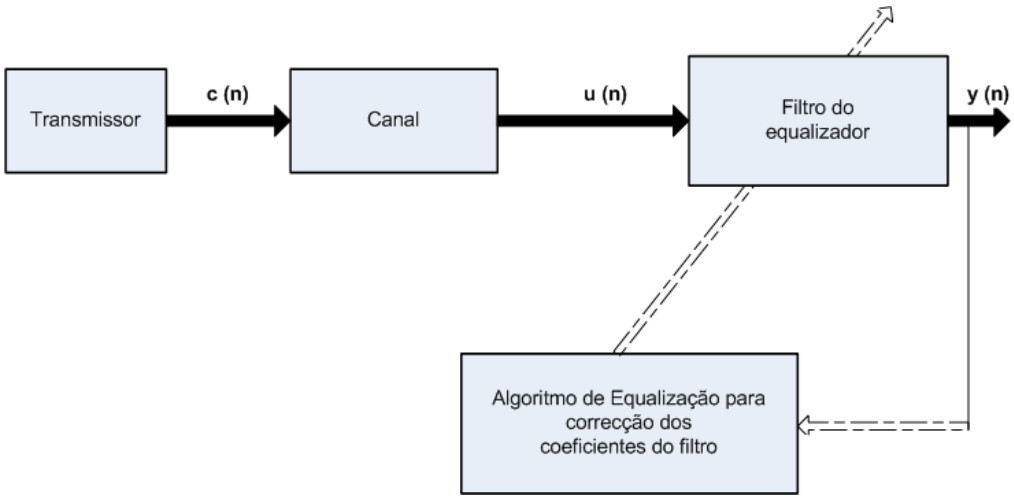


Figura 2.8: Esquema básico do processo de equalização do CMA.

As etapas que definem este algoritmo são apresentadas nas três seguintes equações.

A primeira etapa apresenta o símbolo calculado $y(n)$ após a convolução dos dados de entrada com os coeficientes, segundo a equação:

$$y(n) = \mathbf{w}^H(n) \cdot \mathbf{u}(n) \tag{2.1}$$

onde $\mathbf{w}^H(n)$ representa o vector transposto e conjugado dos coeficientes e $\mathbf{u}(n)$ o vector de dados com o mesmo comprimento do vector de coeficientes.

A segunda etapa é o cálculo do erro $e(n)$:

$$e(n) = y(n) \cdot (R_2 - |y(n)|^2) \tag{2.2}$$

onde R_2 é uma constante que depende da constelação seleccionada.

O cálculo dos novos coeficientes $\mathbf{w}(n+1)$ é a terceira etapa. A actualização é realizada da seguinte forma:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \cdot \mathbf{u}(n)e^*(n) \tag{2.3}$$

onde μ é o passo de adaptação do algoritmo.

Estas etapas são realizadas por esta ordem e cria-se aqui um ciclo de maneira a que cada ciclo representa um símbolo calculado.

Uma característica importante deste algoritmo é a inicialização dos coeficientes, que dela depende o sucesso da convergência do CMA. A inicialização usada neste trabalho, e que apresenta bons resultados de adaptação, é a denominada de *center spike* e consiste em colocar o coeficiente central igual à unidade, enquanto que todos os outros coeficientes são colocados a zero.

As amostras dos símbolos que vão ser tratadas pelo algoritmo são números complexos, constituídos pelas componentes real e imaginária, pelo que é necessário realizar operações com esses números complexos. Como a FPGA não realiza tais operações complexas há a necessidade de reescrever as equações (2.1), (2.2) e (2.3) em equações mais simples.

De seguida são apresentadas as equações com a separação em parte real e imaginaria para cada etapa.

Cálculo de $y(n)$:

$$\Re e\{y(n)\} = \Re e\{w(n)\} \times \Re e\{u(n)\} + \Im m\{w(n)\} \times \Im m\{u(n)\} \tag{2.4}$$

$$\Im m\{y(n)\} = \Re e\{w(n)\} \times \Im m\{u(n)\} - \Im m\{w(n)\} \times \Re e\{u(n)\} \tag{2.5}$$

As equações podem ser reescritas de outra forma:

$$U = A + jB \tag{2.6}$$

$$W^* = (C + jD)^* = C - jD \tag{2.7}$$

$$Y = W^* \times U = (C - jD) \times (A + jB) = (CA + DB) + j(CB - DA) \tag{2.8}$$

Separadas as partes real e imaginária, vem:

$$Re\ Y = CA + DB \tag{2.9}$$
$$Im\ Y = CB - DA \tag{2.10}$$

Cálculo de $e(n)$:

$$\Re e\{e(n)\} = \Re e\{y(n)\} \times (R_2 - |y(n)|^2) \tag{2.11}$$

$$\Im m\{e(n)\} = \Im m\{y(n)\} \times (R_2 - |y(n)|^2) \tag{2.12}$$

onde,

$$|y(n)|^2 = \left(\sqrt{\Re e\{y(n)\}^2 + \Im m\{y(n)\}^2}\right)^2 = \Re e\{y(n)\}^2 + \Im m\{y(n)\}^2 \tag{2.13}$$

E, por último, a actualização de $w(n+1)$:

$$\Re e\{w(n+1)\} = \Re e\{w(n)\} + \mu \times [\Re e\{u(n)\} \times \Re e\{e(n)\} + \Im m\{u(n)\} \times \Im m\{e(n)\}] \tag{2.14}$$

$$\Im m\{w(n+1)\} = \Im m\{w(n)\} + \mu \times [\Re e\{u(n)\} \times \Im m\{e(n)\} - \Im m\{u(n)\} \times \Re e\{e(n)\}] \tag{2.15}$$

O facto deste algoritmo não considerar a fase do sinal faz com que apresente uma constelação rodada.

Surge, então, a necessidade de partir para um outro algoritmo: o LMS.

2.5.2 Algoritmo LMS *(Least Mean Square)*

O algoritmo LMS é a mais popular de todas as estimativas sugerida por Widrow e Hoff em 1959 e consiste simplesmente em substituir os valores médios das variáveis pelos seus valores instantâneos [13].

É um método estocástico de gradiente descendente em que os coeficientes do filtro adaptativo são obtidos por forma a minimizar o erro quadrático médio da diferença entre o valor decidido e o valor estimado do sinal.

Este algoritmo é em tudo semelhante ao CMA excepto no cálculo do erro. Para tal é necessário um módulo extra que realiza a decisão do símbolo à saída do equalizador, como mostra a Figura 2.9. Nesta configuração o LMS funciona em modo Dedicado à Decisão (DD), existe também a configuração em moto treino que será abordada mais à frente.

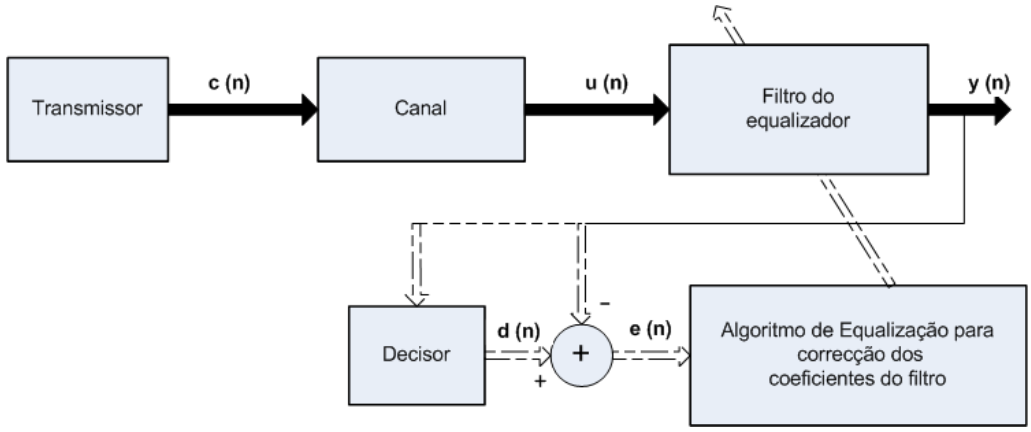


Figura 2.9: Esquema básico do processo de equalização do LMS em modo DD.

Após a decisão, o cálculo do erro é possível através da seguinte equação:

$$e(n) = d(n) - y(n) \tag{2.16}$$

onde $d(n)$ é o símbolo detectado pelo decisor.

As equações separadas em parte real e imaginária são, agora, as seguintes:

$$\Re e\{e(n)\} = \Re e\{d(n)\} - \Re e\{y(n)\} \tag{2.17}$$

$$\Im m\{e(n)\} = \Im m\{d(n)\} - \Im m\{y(n)\} \tag{2.18}$$

Uma outra característica − que representa uma dificuldade − deste algoritmo é a necessidade de inicialização dos coeficientes. Uma forma de iniciar é ter uma sequência conhecida de treino até atingir a convergência, como indica a Figura 2.10.

Depois de atingir a convergência o LMS pode ser comutado para o modo DD (Figura 2.9), onde fica a funcionar sem qualquer apoio de treino.

Contudo, dada a dificuldade de garantir uma sequência de treino, há uma maneira mais eficaz de obter a inicialização dos coeficientes do LMS: usar o CMA numa fase inicial até se atingir a convergência e depois utilizar esses coeficientes para inicializar e executar o algoritmo LMS. Neste trabalho é esta a configuração usada.

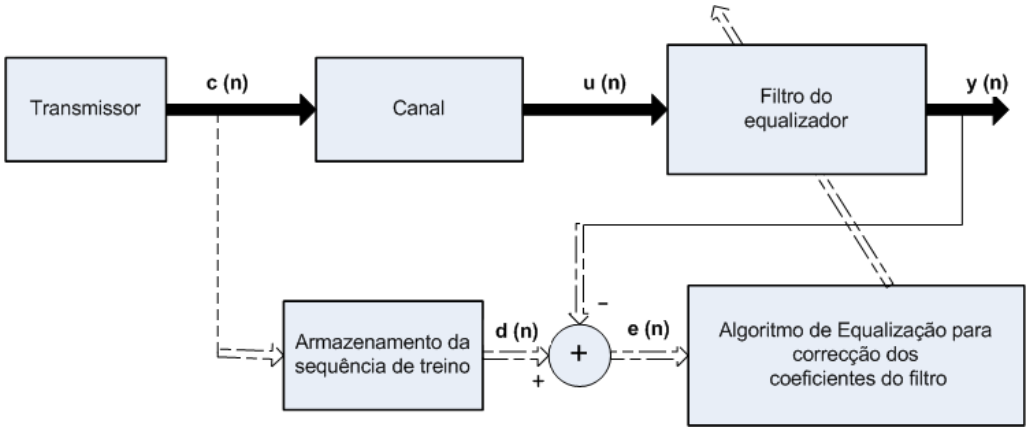


Figura 2.10: Esquema básico do processo de equalização do LMS em modo de treino.

2.6 Compensação da dispersão

A compensação da dispersão pode ser realizada no domínio óptico ou no domínio eléctrico. No domínio óptico é possível realizar a compensação através de fibras com dispersão cromática contrária à do sistema de transmissão. No domínio eléctrico é possível compensar a dispersão cromática com a ajuda de filtros de resposta impulsional. Este método veio tirar a complexidade que existia ao nível óptico.

Em DSP, este módulo pode ser implementado em conjunto com os equalizadores discutidos anteriormente. Assim, colocando um módulo antes dos algoritmos de equalização, LMS e CMA, é possível compensar a dispersão e ter um óptimo desempenho no funcionamento do equalizador.

A implementação do módulo de compensação é realizada em FPGA através de um filtro de coeficientes fixos, cujos valores são calculados de acordo com a transformada inversa de *Fourier* da função de transferência da fibra representada pela equação:

$$G(z,\omega) = exp\left(-j\frac{D\lambda^2 z}{4\pi c}\omega^2\right) \tag{2.19}$$

onde D representa o coeficiente de dispersão da fibra, λ o comprimento da luz, z a distância de transmissão, c a velocidade da luz e ω a frequência angular.

O filtro de compensação da dispersão, logicamente, é dado por um filtro passa tudo com a característica $1/G(z,\omega)$ e pode ser construído usando a recursividade ou a não-recursividade dos filtros digitais [15].

2.7 Resumo

Neste capítulo, foi apresentado o estado da arte em sistemas ópticos coerentes, onde são descritos os elementos de um sistema óptico IM/DD que usa detecção directa no receptor. Foi apresentado o conceito de sistemas ópticos coerentes e os seus elementos constituintes. Diferenciou-se

as detecções heterodina e homodina.

Na secção 2.4, foram apresentadas técnicas recentes de detecção coerente usando processamento digital de sinal numa FPGA e foi, ainda, detalhado o sistema de transmissão.

Após a detecção coerente usando DSP, foi realizada uma abordagem aos algoritmos CMA e LMS, explicando o funcionamento de cada um.

Para concluir, referiu-se a compensação da dispersão no domínio eléctrico, que é conseguida através de filtros usando a função de transferência da fibra para obter os coeficientes fixos.

Para implementação dos algoritmos em FPGA é necessário o conhecimento das FPGAs e das ferramentas usadas para as programar. No capítulo 3, que se segue, é apresentada a arquitectura de FPGAs e é feita uma descrição das funcionalidades da ferramenta *System Generator*, usada neste trabalho, para implementação dos algoritmos propostos.

Capítulo 3

Arquitectura de FPGAs e ferramentas de desenvolvimento

3.1 Introdução

Um dispositivo FPGA é uma plataforma de implementação constituída por um conjunto de células lógicas. É um circuito integrado de aplicação específica programado pelo criador ao invés de ser pelo fabricante do dispositivo, como é numa aplicação específica de circuitos integrados *(ASIC − Application Specific Integrated Circuit)*, que pode desempenhar uma função semelhante a um sistema electrónico, mas uma FPGA pode ser reprogramada um número de vezes sem conta [7]. Essa possibilidade de programação, associada a potentes ferramentas de desenvolvimento e modelagem, possibilita ao utilizador o acesso a projectos de circuitos integrados complexos evitando, deste modo, os custos de engenharia associados aos ASICs.

Relativamente à programação da FPGA, há um conjunto de ferramentas de software associado a um fluxo de projecto que proporciona um alto nível de abstracção ao criador, permitindo que este se foque no algoritmo que deseja implementar, em vez de se preocupar com os circuitos que serão implementados. Desta forma, a programação do dispositivo pode ser feita através de uma linguagem de descrição de hardware (VHDL ou Verilog) ou recorrendo à ferramenta de modelação de sistemas − *System Generator*.

Este capítulo começa por apresentar a arquitectura de FPGAs na secção 3.2. Segue-se uma descrição das funcionalidades da ferramenta *System Generator* − ferramenta esta que foi usada para o desenvolvimento deste trabalho. Aspectos a considerar em implementação usando *System Generator* são apresentados na secção 3.4, terminando ente capítulo com a co-simulação em hardware que é uma das funcionalidades do *System Generator*.

3.2 Arquitectura de FPGA

Como já referido, FPGA é um circuito integrado que contém um elevadíssimo número de células lógicas. Estas, juntamente com as suas interligações, formam a estrutura básica da FPGA que o utilizador tem ao seu dispor para desenvolver sistemas complexos.

Na família Virtex da Xilinx, a menor unidade lógica configurável é denominada de *slice* e é apresentada na Figura 3.1.

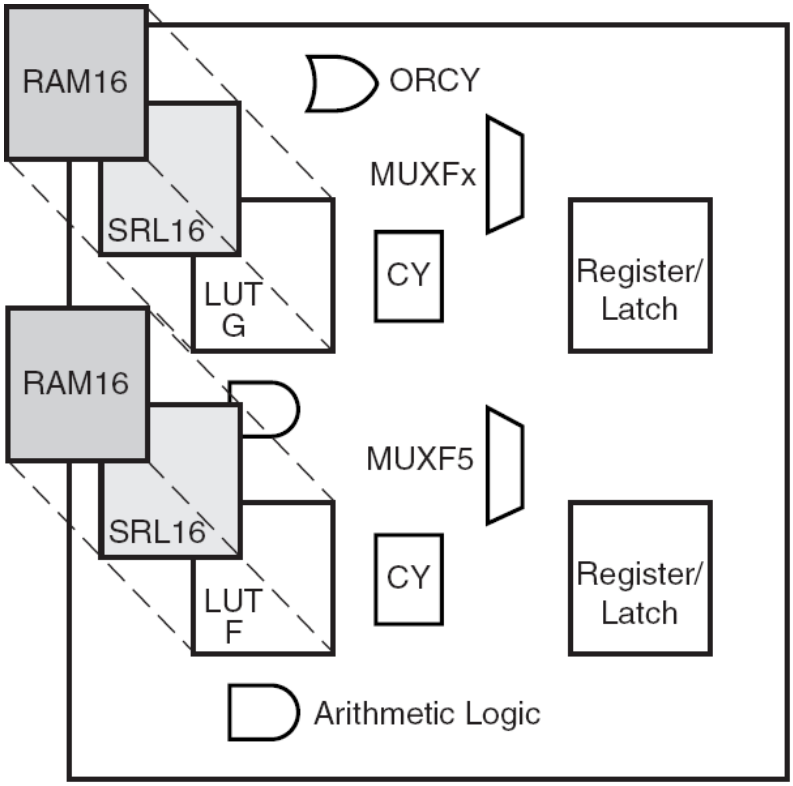


Figura 3.1: *Slice*−unidade lógica básica da família Virtex [7].

A *slice* é bastante flexível e pode ser configurada para funcionar como *LookUp Tables* (LUT) com quatro entradas e uma saída. A Figura 3.1 mostra também que a *slice* pode ser ainda configurável em memórias RAM distribuídas de 16 bits e *shift registers* também de 16 bits.

É constituída, ainda, por recursos adicionais tais como: *flip flops* tipo D, multiplexadores, circuito dedicado para sinais de transporte *(carry)* e portas lógicas. Estes recursos podem ser utilizados em conjunto com as LUTs para implementar funções booleanas, multiplicadores e somadores com palavras de comprimento bastante flexível. Importa referir que a rapidez dos somadores depende do atraso que o *carry* impõe nas operações, isto é, depende do tempo de propagação desse *carry*. Mas na arquitectura da FPGA há um caminho com lógica dedicada para o transporte o que torna possível ter somadores com um grau de rapidez bastante elevado, cuja utilização neste trabalho resulta num elevado benefício.

Na operação como SRL *(Shift Register LUT)*, estes recursos adicionais podem ser utilizados para implementar contadores, conversores série-paralelo e paralelo-série, entre outras funcionalidades. Os recursos adicionais mencionados podem ainda ser utilizados na interligação de unidades lógicas para implementar, por exemplo, multiplicadores, contadores, somadores e memórias com praticamente qualquer comprimento. O limite para este comprimento é determinado pela quantidade de *slices* disponíveis na FPGA, que é proporcional à área do circuito integrado.

Recursos especiais

Além das *slices* uma FPGA pode disponibilizar mais recursos bastante sofisticados, como é o caso dos multiplicadores dedicados, usados em implementação neste trabalho, que gozam da vantagem de serem muito mais rápidos que multiplicadores construídos em memória distribuída. Um outro recurso especial é a memória bloco RAM (BRAMs), que é mais eficiente que uma RAM configurada em memória distribuída. Além dos já mencionados, existem também outros recursos especiais como sejam os dispositivos MAC (multiplicador seguido de um acumulador, também chamados de *slice XtremeDSP)*, DCM (*Digital Clock Manager*, utilizado para multiplicar ou dividir a frequência do sinal de relógio) e micro-controladores *(IBM PowerPC)*.

A utilização destes recursos adicionais possibilita a optimização do consumo de *slices* e desenvolver projectos mais eficientes.

Para alcançar essa eficácia dos algoritmos pode-se usar o *System Generator*, que é descrito na secção seguinte, pois ele optimiza automaticamente o uso de todos esses recursos.

3.3 *System Generator*

O *System Generator* é uma ideia pioneira da Xilinx para programar FPGAs. É uma ferramenta de projecto integrado que utiliza o *Simulink* − ferramenta de modelação, simulação e análise de sistemas dinâmicos do MATLAB − como suporte de desenvolvimento.

Além do *Simulink*, o *System Generator* utiliza um conjunto de ferramentas para especificar os detalhes de implementação de hardware em dispositivos da Xilinx (FPGA Virtex 4, por exemplo), como se observa na Figura 3.2. Para tal, o *System Generator* utiliza a *Xilinx DSP Blockset* no *Simulink* e para gerar a *Netlist* optimizada dos módulos DSP invoca automaticamente o *Xilinx Core Generator*. Opcionalmente, pode-se gerar um *testbench* para usar no *ModelSim* ou no *Xilinx ISE Simulator* para aprofundar o nível de detalhe do projecto a implementar.

É no *Simulink* que o *System Generator* é apresentado sob a forma de uma biblioteca (*Xilinx Blockset* como referencia a Figura 3.3) adicionada a todas as outras bibliotecas do *Simulink*, quando este é instalado. A Figura 3.3 ilustra os diversos blocos que a biblioteca contém e a conexão desses blocos permite desenvolver modelos funcionais dum sistema.

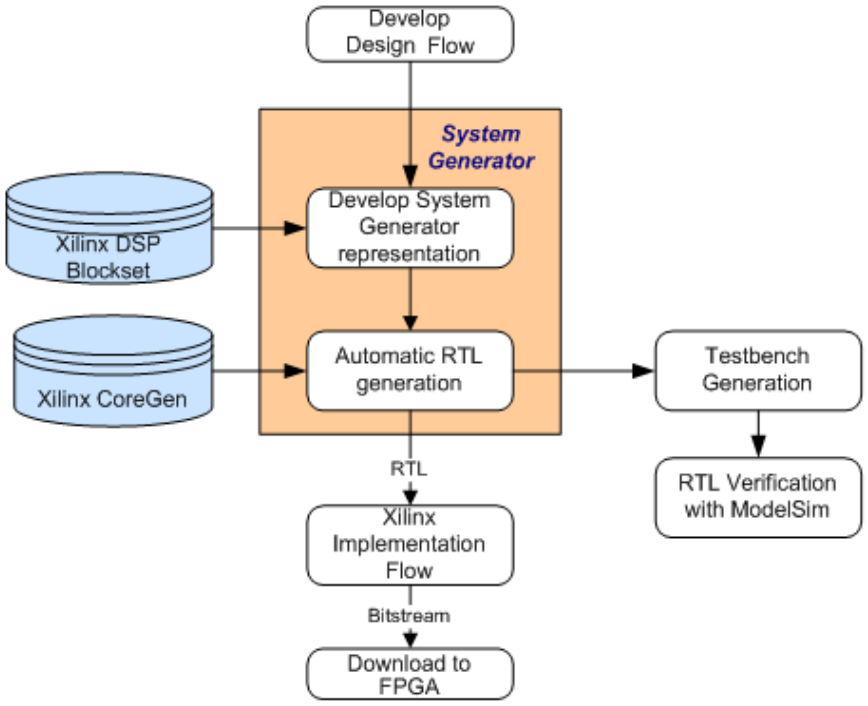


Figura 3.2: Ferramentas utilizadas pelo *System Generator* [8].

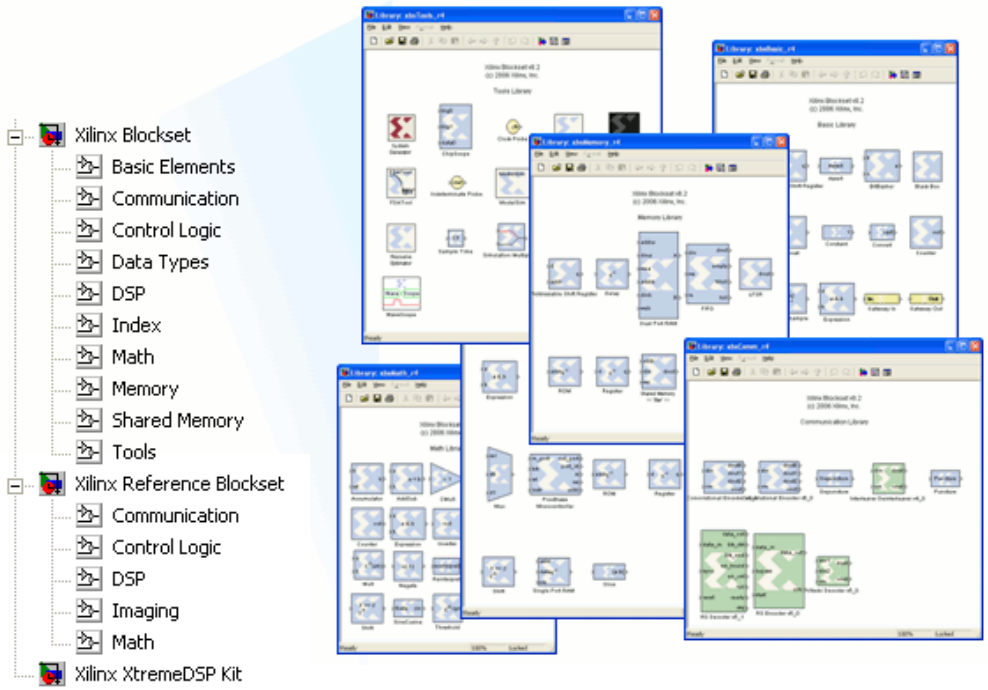


Figura 3.3: *Xilinx Blockset* – Biblioteca com o conjunto de blocos do *System Generator* [7].

Através dos blocos apresentados na Figura 3.3, o *System Generator* possibilita ao utilizador desenvolver algoritmos sofisticados e sistemas de processamento de sinal, abstraindo-se de funções complexas de matemática, lógica, memória ou DSP. A biblioteca da Xilinx no *Simulink* possui também blocos que proporcionam interfaces com outras ferramentas (por exemplo, *FDATool*, *ModelSim*) e outros que geram automaticamente o código VHDL ou Verilog [7].

Estes blocos estão organizados por categorias. As Tabelas 3.1 e 3.2 apresentam uma breve descrição de cada categoria associada às bibliotecas *Xilinx Blockset* e *Xilinx Reference Blockset*, respectivamente.

Categorias	**Descrição**
Elementos Básicos	Padrão de blocos standard para construção de lógica digital
Comunicação	Blocos correctores de erros (FEC−*Forward Error Correction*) e moduladores usados em sistemas de comunicação digital
Lógica de Controlo	Blocos para controlo de circuitos e máquinas de estado
Tipos de Dados	Blocos que convertem tipos de dados (inclui *Gateways*)
DSP	Blocos de processamento digital de sinal
Matemática	Blocos que implementam funções matemáticas
Memória	Blocos que implementam memórias e o seu acesso
Memória Distribuída	Blocos que implementam e acedem à memória distribuída Xilinx
Ferramentas	Contém blocos muito úteis, tais como: *System Generator*, estimação de recursos, co-simulação, etc.

Tabela 3.1: Descrição de cada categoria da biblioteca *Xilinx Blockset* [7].

Categorias	**Descrição**
Comunicação	Blocos frequentemente usados em sistemas de comunicação digital
Lógica de Controlo	Blocos lógicos usados para controlar circuitos e máquinas de estado
DSP	Blocos de processamento digital de sinal
Imagem	Blocos para realizar processamento de imagem
Matemática	Blocos que implementam funções matemáticas

Tabela 3.2: Descrição de cada categoria da biblioteca *Xilinx Reference Blockset* [7].

No *Simulink* podem ser usados os blocos das bibliotecas *Simulink* em conjunto com os do *System Generator*. No entanto, há que ter em linha de conta alguns aspectos a considerar quando se usam estes blocos em implementação.

3.4 Aspectos a considerar na implementação

Há dois aspectos a considerar, desde logo, quando se inicia uma implementação: um é o uso do bloco *System Generator*, pois sem ele o *Simulink* não consegue realizar a simulação do projecto a executar; o outro, fundamental, é a passagem dos dados para o campo do *System Generator* – isso é conseguido através dos blocos *Gateway In* e *Gateway Out*. Estes blocos definem a fronteira da FPGA no ambiente de simulação, onde o *Gateway In* representa a entrada de dados na FPGA e o *Gateway Out* é a saída de dados da FPGA, como se observa na Figura 3.4.

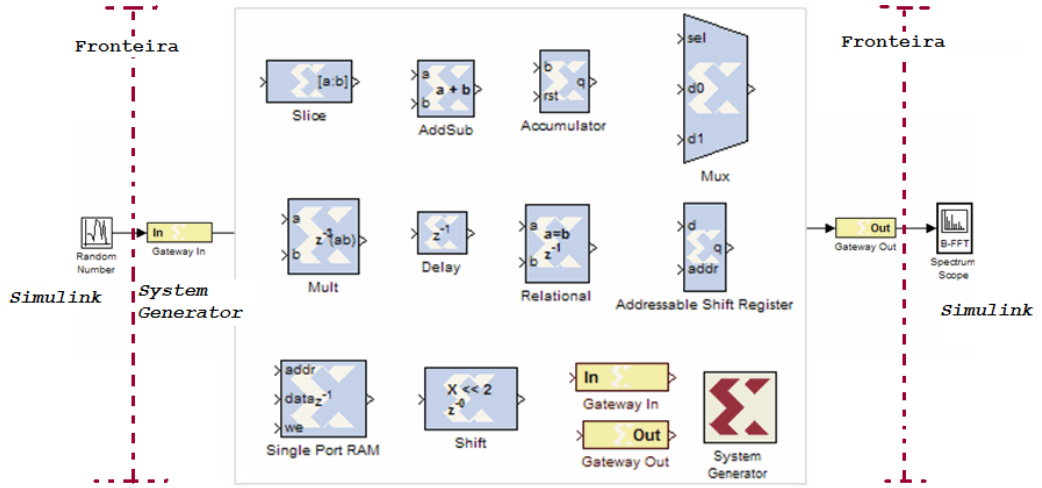


Figura 3.4: Entradas e saídas de dados da FPGA – Fronteiras [8].

Estes blocos realizam a interpretação dos dados, pois só assim é possível a representação dos mesmos na FPGA.

3.4.1 Representação de dados em hardware

O bloco *Gateway In* converte os dados provenientes do MATLAB (onde a precisão é alta – *double*) num sinal binário, no qual a precisão vai depender do número de bits disponíveis para o efeito. Mais precisamente, este bloco converte um número representado em vírgula flutuante num número que será representado em vírgula fixa. Essa conversão é demonstrada na Figura 3.5, onde se pode ver que o *System Generator* não usa todos os bits do número representado em *Simulink*.

Neste trabalho o número de bits usados para a representação dos dados é 18 bits, sendo que 1 bit é reservado para sinal, 5 bits para a parte inteira e os restantes 12 bits para a parte fraccionaria. O que pesou para a escolha do número 18 foi o facto de haver FPGAs, como já referido na secção 3.2, que disponibilizam multiplicadores 18×18 dedicados que realizam a multiplicação

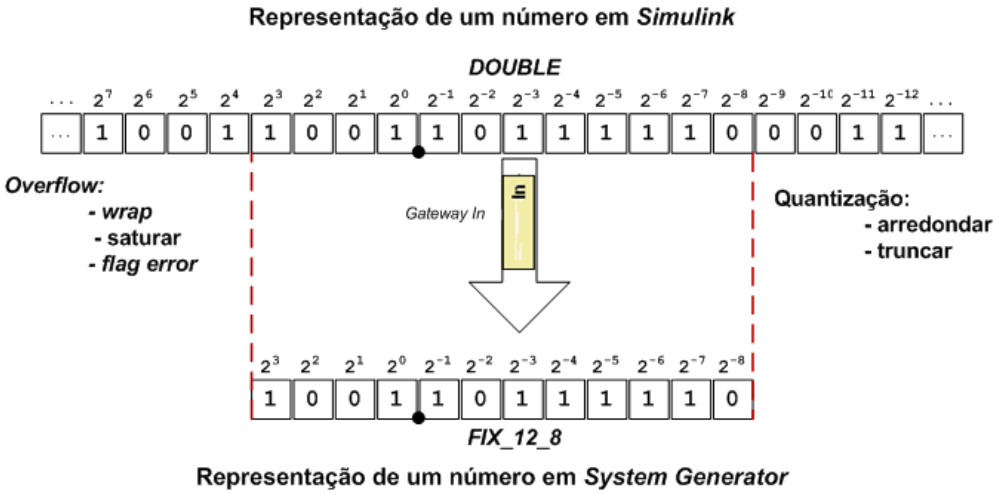


Figura 3.5: Conversão de um número representado em vírgula flutuante para vírgula fixa.

de um número de 18 bits por outro de 18 bits. Utilizar mais bits que os 18 terá um custo superior em termos de utilização destes recursos e como são em número bastante limitado, importa evitar o desperdício despropositado deles. Para um valor superior a solução passa por utilizar multiplicadores em memória distribuída.

Depois de converter o sinal para vírgula fixa, este pode já ser interpretado e tratado pelos outros blocos da biblioteca da Xilinx, que é o mesmo que dizer que o sinal já pode ser tratado pela FPGA. O bloco *Gateway Out* realiza o processo inverso.

3.4.2 Precisão

A precisão de um sinal representado em *System Generator* é tanto maior quanto maior for o número de bits, quer à direita quer à esquerda, da vírgula binária que separa os bits reservados para a parte inteira dos reservados para a parte decimal. A maioria dos blocos permite ao utilizador escolher a precisão que melhor se ajusta ao projecto. Os blocos da biblioteca Xilinx são capazes de determinar o tipo apropriado de saída baseado nos seus tipos de entrada. A maioria dos blocos trabalha com dois tipos de precisão: a precisão máxima ou a precisão definida pelos bits que o utilizador definir.

O ideal era ter um número infinito de bits para a precisão ser a máxima possível, no entanto, quanto mais bits se usar para definir essa precisão mais recursos estarão a ser utilizados. O limite é 4096 bits [16].

Este limite leva a que seja necessário descartar informação provocando erros, ou por *overflow* (nos bits mais significativos − MSB), ou por quantização (nos bits menos significativos − LSB).

Overflow e quantização

Overflow, como já referido, afecta os bits MSB e ocorre quando os valores estão fora da gama representável. Erros de quantização surgem quando o número de bits da parte fraccionaria é

insuficiente para representar a fracção decimal de um valor (LSB).

A opção vírgula fixa da Xilinx suporta várias opções para definir a precisão. Para *overflow* as opções são: saturar, *wrap*, ou lançar um erro no *Simulink* durante a simulação *(flag error)*.

Para a quantização essas opções são: arredondar, ou truncar [16].

Perante este cenário, a melhor opção passará por saturar em *overflow* e arredondar para a quantização de modo a minimizar o erro de precisão.

3.4.3 Determinação automática do número de bits

Um outro aspecto relevante a ter em consideração é o aumento do número de bits que se propaga ao longo de um sistema: tanto maior é esse aumento, quanto maior o número de operações a realizar.

Por exemplo, o número de bits à saída de um bloco que realiza a multiplicação é o dobro do número de bits da entrada, como se pode ver na Figura 3.6. Portanto, o número de multiplicações que um algoritmo realiza traduz-se num crescente número de bits. Esta situação não é viável uma vez que, na maioria dos casos, nem se quer são necessários todos os recursos, o que pode ser resolvido usando um bloco para converter o sinal.

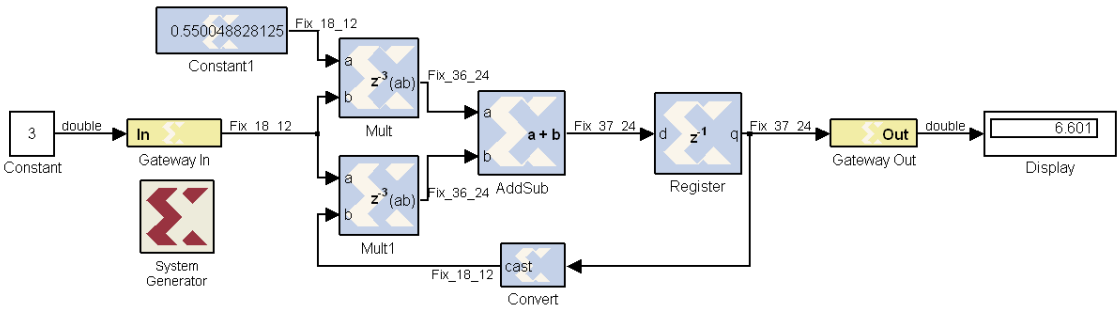


Figura 3.6: Contagem automática do número de bits.

Outro exemplo são as somas, que aumentam de um bit à esquerda para o caso de haver *carry*.

A implementação de realimentações num sistema implica um crescente número de bits, como é ilustrado na Figura 3.6. Isto é contornado usando o bloco *convert* evitando, desta forma, que o sistema use um elevado número de bits desnecessariamente.

Após todas as considerações tomadas na realização de um sistema ou algoritmo, é possível realizar uma co-simulação em hardware, onde o projecto desenvolvido é carregado na FPGA ficando disponível para testes em conjunto com o *Simulink*.

3.5 Co-simulação em hardware

Co-simulação em hardware consiste num género de compilação criando automaticamente *bitstreams* e blocos associados a esses *bitstreams*.

Para realizar a co-simulação é necessária a instalação da plataforma de hardware que se pretende utilizar. Depois da instalação gera-se o modelo do sistema que se deseja correr no hardware.

Esse modelo é gerado automaticamente pelo *System Generator* que produz um bloco de co-simulação em hardware, como o ilustrado na Figura 3.7.

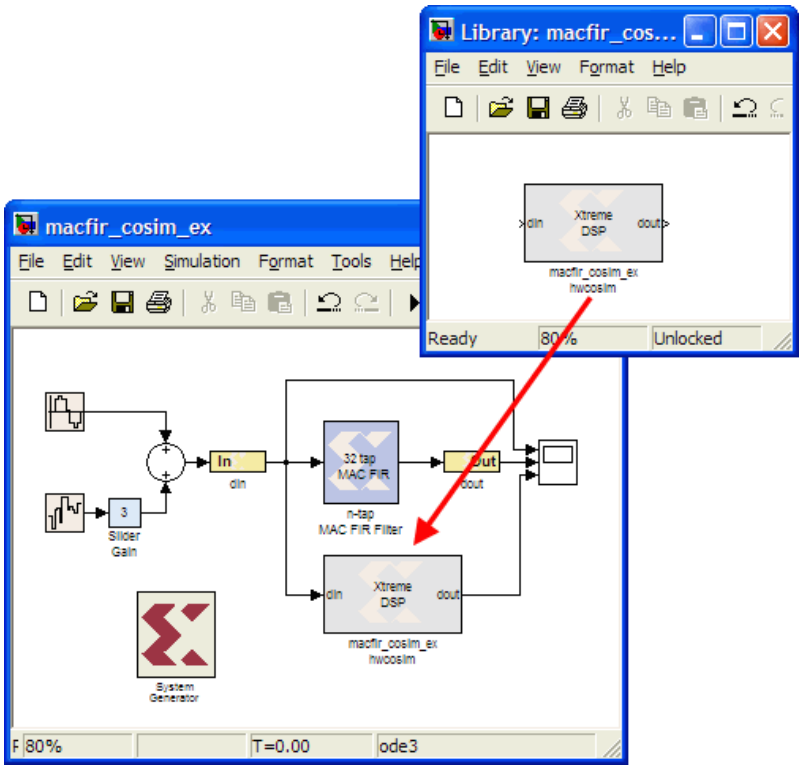


Figura 3.7: Exemplo de um bloco de co-simulação em hardware [7].

Este bloco torna passível interfaces entre a FPGA e o PC por uma ligação física (USB ou JTAG). É através dessa ligação que vai ser possível colocar valores na FPGA e lê-los através das fronteiras da FPGA exibidas na Figura 3.4 atrás apresentada. Em simulação, o bloco entrega os mesmos resultados ao *Simulink*, mas difere no facto de os dados serem processados em hardware. Portanto, a co-simulação é uma forma de validar em hardware o sistema implementado.

Esse processamento dos dados em hardware pode ser realizado mediante duas opções disponíveis com o bloco co-simulação: uma é simulação passo a passo, a outra é simulação em funcionamento livre.

A simulação passo a passo foi a opção usada neste trabalho, porque permite a sincronização dos dados com a simulação do *Simulink*, uma vez que são processados na FPGA e devolvidos os resultados ao ambiente *System Generator*. No caso dos dados processados na FPGA não terem retorno ao *Simulink* pode-se usar a opção funcionamento livre.

3.6 Resumo

Neste capítulo apresentou-se uma descrição das funcionalidades da ferramenta *System Generator* e das considerações a ter em linha de conta, importantes para uma óptima implementação dos algoritmos apresentados no capítulo 2.

Seguem-se, no capítulo 4, as etapas da implementação dos algoritmos adaptativos na ferramenta descrita neste capítulo.

Capítulo 4

Implementação dos algoritmos adaptativos com o *System Generator*

4.1 Introdução

A descrição das ferramentas de desenvolvimento para a implementação em FPGA realizada no capítulo 3 é importante para adquirir sensibilidade e tomar conhecimento com as plataformas e tecnologias de implementação. É na ferramenta de desenvolvimento *System Generator* que é realizada a implementação dos algoritmos discutidos no capítulo 2.

Como já foi referido em capítulos anteriores, o trabalho a desenvolver centra-se na implementação dos algoritmos CMA e LMS em FPGA para a estimação de sinal em sistemas ópticos coerentes. Os algoritmos em causa haviam sido já implementados em MATLAB, o que permite que haja uma comparação de resultados dos já implementados com a nova implementação em *System Generator*. Portanto, o propósito deste trabalho é, agora, implementá-los em FPGA.

Nas próximas secções, são apresentadas as etapas de implementação dos algoritmos CMA e LMS em *System Generator* em duas configurações diferentes. A secção 4.2 inicia com a configuração sequencial e termina com a configuração paralela.

Na secção 4.3 encontram-se as etapas da implementação do LMS, terminando este capítulo com a implementação do compensador da dispersão e o conceito de paralelismo do CMA/LMS.

4.2 Etapas da implementação do algoritmo CMA

Com o estudo dos *scripts* de MATLAB foi possível desenhar o diagrama de blocos da Figura 4.1 que se segue onde são identificadas todas as áreas do equalizador CMA a implementar. Estas áreas estão divididas por blocos, representando cada bloco uma etapa da implementação do algoritmo CMA.

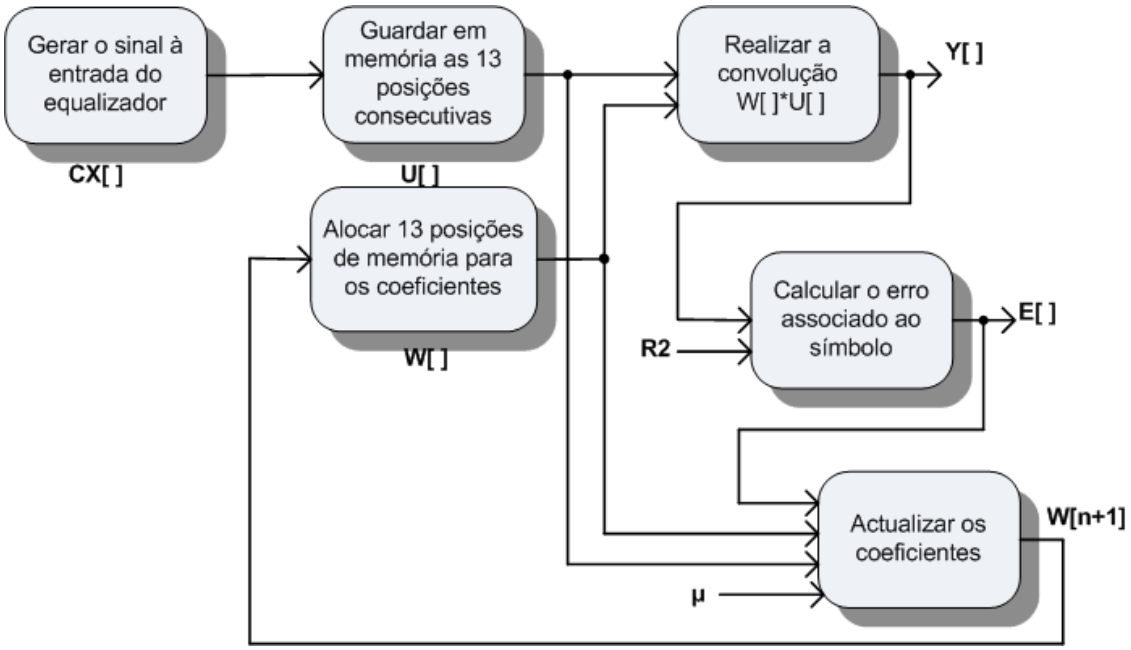


Figura 4.1: Diagrama de blocos do equalizador CMA.

Essas etapas são assim numeradas:

1. Gerar o sinal à entrada do equalizador.
2. Guardar em memória as 13 posições consecutivas.
3. Alocar 13 posições de memória para os coeficientes.
4. Realizar a convolução W*U (Filtro do equalizador).
5. Calcular o erro associado ao símbolo.
6. Actualizar os coeficientes.

Existem várias formas de realizar a implementação destas etapas, destacando-se duas dessas formas originando as configurações: sequencial e paralela.

A configuração sequencial tem a vantagem de usar um menor número de recursos da FPGA, uma vez que trata os dados em série.

A configuração paralela não beneficia deste facto, mas em contra partida tem a vantagem de ter menor latência de execução em comparação com a configuração anterior, uma vez que realiza o tratamento dos dados em paralelo, necessitando, para tal, de utilizar mais recursos.

Procede-se, na secção seguinte, à descrição das etapas da implementação na configuração sequencial.

4.2.1 Versão sequencial

1. Gerar o sinal à entrada do equalizador

A primeira etapa é gerar o sinal à entrada do equalizador. Os algoritmos já implementados em MATLAB permitem obter o vector de dados que o equalizador irá tratar. Tendo sido esse vector guardado no *Workspace* do MATLAB, uma forma de usar esses valores no *Simulink* é usar o bloco *Signal From Workspace*, como indica a Figura 4.2, com um período de amostragem de 13.

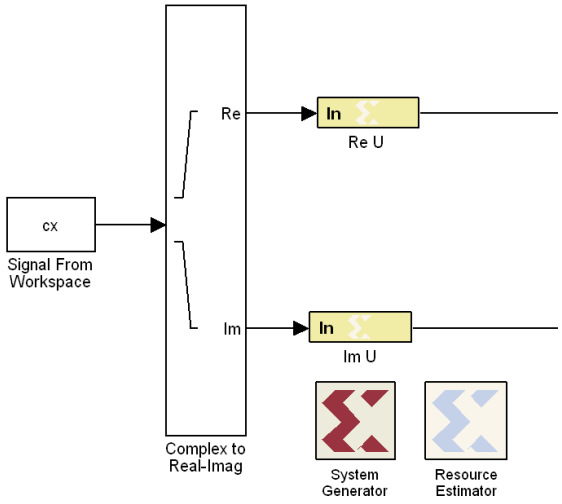


Figura 4.2: Entrada de dados.

Deste modo é facultada a passagem dos dados do MATLAB para o equalizador. Sendo os dados do vector 'CX' números complexos há a necessidade de separá-los em parte real e parte imaginária. Isso é conseguido através do bloco *Complex to Real-Imag* presente na Figura 4.2. Feito isto, os dados são representados em hardware pelos blocos *Gateway In* já discutido no capítulo 3. A figura mostra ainda os blocos *System Generator* e *Resource Estimador*, blocos esses também já descritos no capítulo anterior e que são de relativa importância na implementação dos algoritmos em *System Generator*.

As etapas 2 e 3, que se seguem evidenciadas na Figura 4.3, são referentes ao armazenamento de amostras de símbolos e de coeficientes, respectivamente.

2. Guardar em memória as 13 posições consecutivas

Para o armazenamento das 13 amostras consecutivas a solução implementada foi o *Addressable Shift Register* (ASR). A razão para se armazenar 13 amostras é porque este é o número de coeficientes usados. Este número é muito usado na literatura e através dos 13 coeficientes os algoritmos têm a capacidade de compensar a dispersão cromática da fibra até 200 km de distância.

Outras possíveis soluções para o armazenamento poderiam ter sido implementadas, mas o simples funcionamento deste bloco levou a que a decisão recaísse sobre ele.

Para o endereçamento da memória é usado um contador *(counter1* da Figura 4.3), que conta endereços de memória até 13 e serve para colocar na saída do ASR a amostra correspondente a esse endereço de memória a cada ciclo de relógio.

Portanto, o ASR permite ler os símbolos pela sua ordem de chegada e, na altura de descartar amostras, elimina sempre a mais antiga. Este processo é importante para realizar a convolução na etapa 4. Mas antes dessa etapa, procede-se à explicação da etapa 3.

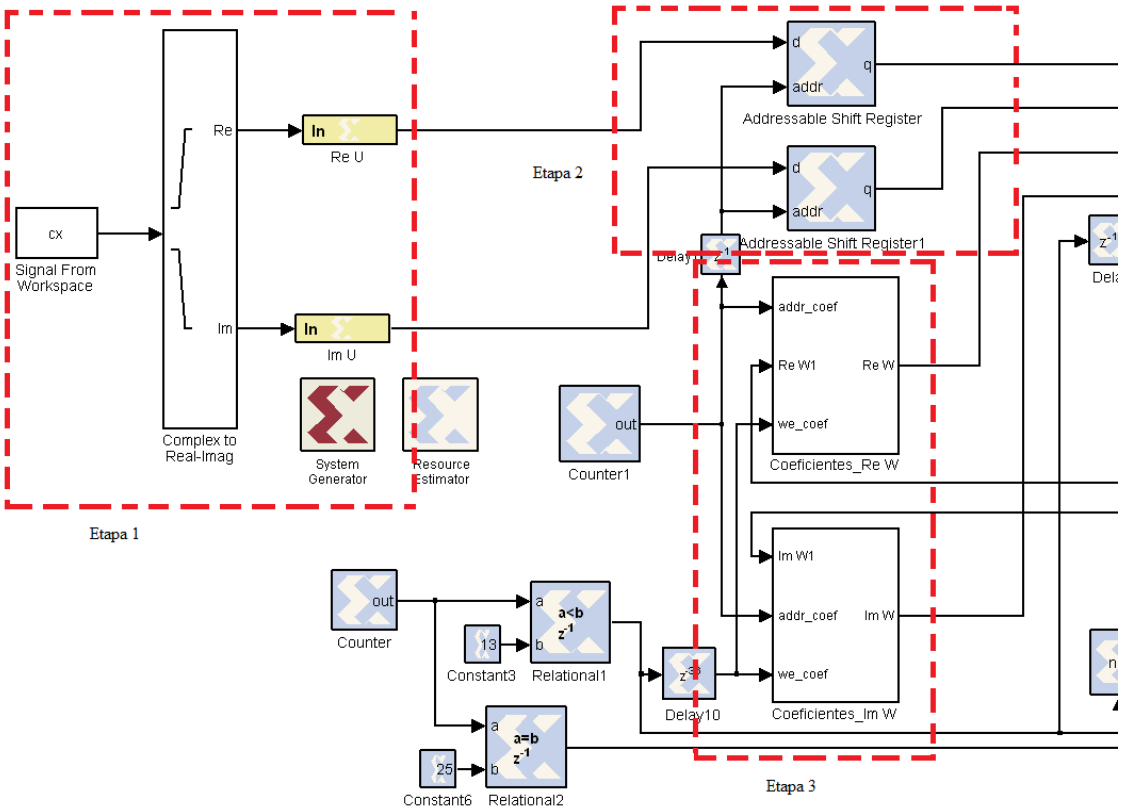


Figura 4.3: Armazenamento dos dados e dos coeficientes.

3. Alocar 13 posições de memória para os coeficientes

Na etapa 3 implementou-se um processo que armazena os coeficientes actualizados provenientes da etapa 6 e os disponibiliza ao filtro do equalizador representado pela etapa 4, etapas que serão explicadas mais à frente neste documento.

Pela análise à Figura 4.4 pode-se observar que a solução para esta etapa do algoritmo é a *Dual Pot RAM*. Optou-se por este recurso disponibilizado por uma grande parte das FPGAs, porque permite a escrita e leitura de coeficientes em simultâneo, desde que não seja no mesmo endereço de memória. Deste modo, a porta B fica dedicada exclusivamente à escrita dos novos coeficientes ("Re W1"), ficando a porta A dedicada para a leitura.

O contador, presente na Figura 4.4, incrementa os endereços de memória para a escrita dos coeficientes e só está activo quando o sinal de escrita estiver a '1', ou seja, só quando for necessário

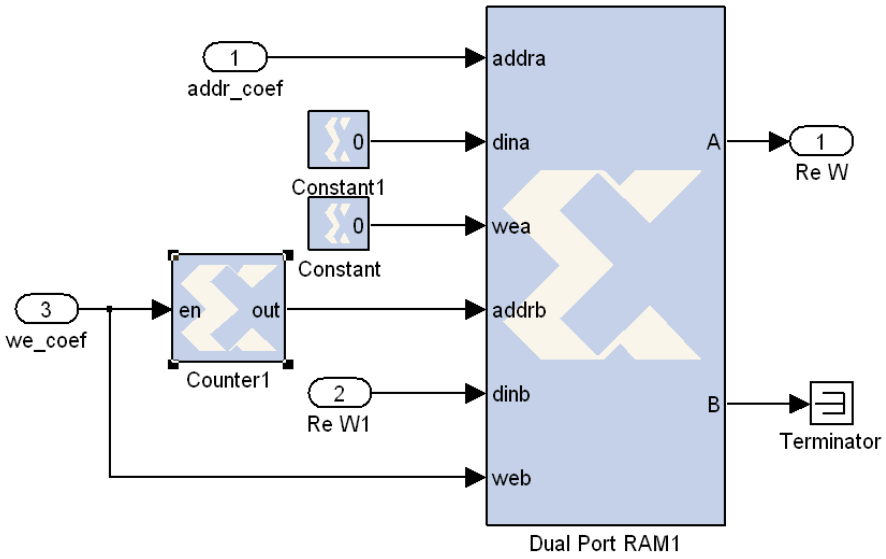


Figura 4.4: *Dual Port RAM* para armazenamento dos coeficientes.

escrever é que os endereços são incrementados. Com isto evita-se que valores não desejáveis sejam escritos nos coeficientes.

Quanto à porta A, esta é exclusiva para a leitura dos coeficientes como já referido, sendo usado o mesmo endereçamento de memória que para o do ASR, garantindo, assim, que os coeficientes estejam em sintonia com os símbolos à entrada do bloco do Filtro, assinalado na Figura 4.5. O filtro é a etapa 4 a qual se descreve a implementação a seguir.

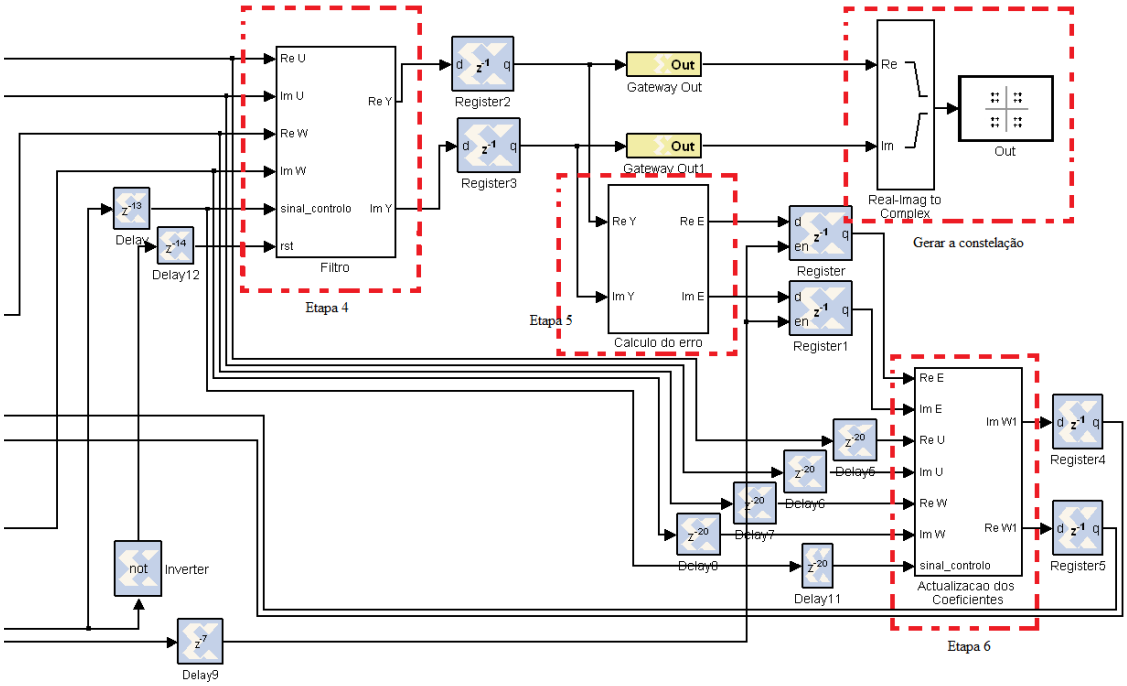


Figura 4.5: Esquema representativo das etapas 4, 5 e 6.

4. Realizar a convolução W*U

O bloco do Filtro (à esquerda na Figura 4.5) representa a etapa 4, e as etapas 5 e 6 são representadas pelos blocos cálculo do erro (ao centro) e actualização dos coeficientes (à direita), respectivamente. A Figura 4.5 apresenta ainda o conjunto de blocos que são utilizados para gerar a constelação para obtenção dos resultados apresentados no capítulo seguinte.

A etapa 4 desempenha o filtro do equalizador, onde é realizada a convolução do sinal de entrada com o sinal dos coeficientes. Para tal, este bloco implementa as equações (2.4) e (2.5) ou as mais simplificadas (2.9) e (2.10) descritas no capítulo 2. Os símbolos calculados serão entregues aos blocos *Gateway Out* para posterior criação da constelação.

A Figura 4.6 que se segue ilustra a implementação desta etapa.

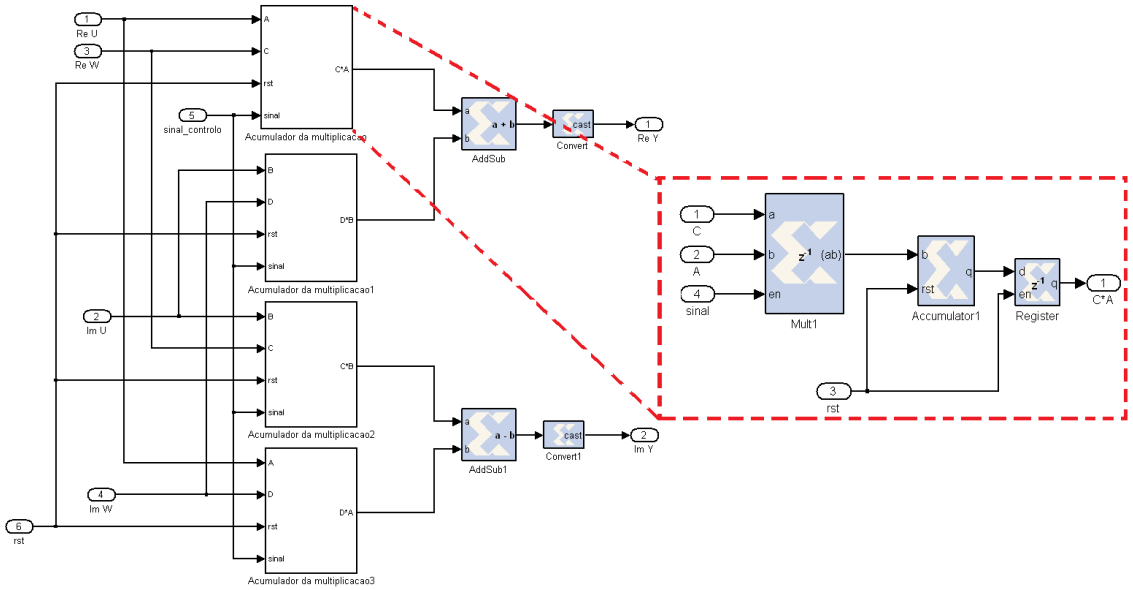


Figura 4.6: Implementação do filtro.

Para realização desta etapa 4 implementou-se um multiplicador seguido de um acumulador, para efectuar a soma das 13 multiplicações consecutivas necessárias para a execução da convolução.

O "sinal_controlo" da Figura 4.6 é um sinal que faz com que o filtro realize a convolução de 2 em 2 amostras, isto é, faz com que os multiplicadores realizem a multiplicação referente às 13 amostras consecutivas guardadas no ASR e durante as 13 seguintes ficam inactivos, permitindo ao algoritmo realizar a actualização dos coeficientes nesse período de tempo e coloca-los disponíveis para as novas multiplicações.

O sinal "rst" é importante para capturar o símbolo calculado e, simultaneamente, colocar a zero o acumulador para o símbolo seguinte. Desta forma, evita-se que na estimação do símbolo o acumulador não introduza ruído, isto é, que o acumulador não contribua para o valor do erro. O cálculo do erro é a etapa que que segue.

5. Calcular o erro associado ao símbolo

Na etapa 5 foi implementado o módulo do cálculo do erro, cuja funcionalidade é produzir as equações (2.11), (2.12) e (2.13), como apresenta a Figura 4.7.

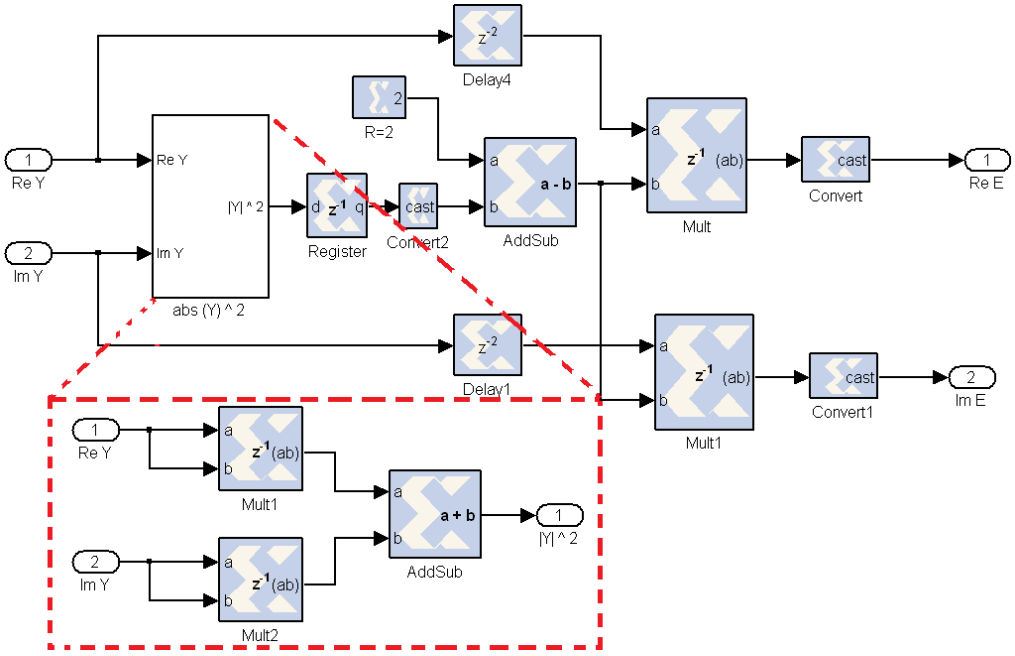


Figura 4.7: Implementação da etapa 5.

Importa também realçar a importância em usar o bloco *Convert*. Sem ele, o sistema teria um crescente aumento do número de bits havendo o interesse em não desperdiçar recursos da FGPA, como foi referenciado no capítulo 3.

Para concluir a implementação da configuração sequencial, apresenta-se a etapa 6 que efectua a actualização dos coeficientes.

6. Actualizar os coeficientes

Para a etapa 6 implementaram-se dois conjuntos de blocos como os apresentados na Figura 4.8.

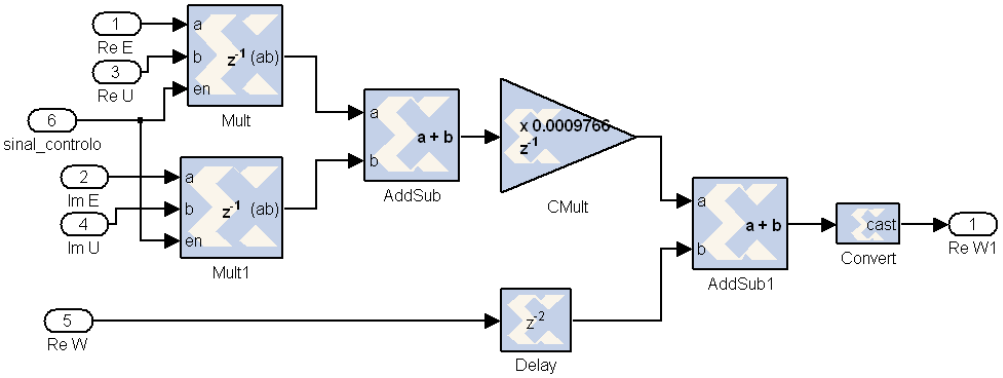


Figura 4.8: Implementação da actualização da parte real dos coeficientes.

É necessário um conjunto de blocos que implemente a actualização da parte real dos coeficientes (como é o caso da Figura 4.8) e um outro que realize o mesmo referente à parte imaginária dos coeficientes, executando as respectivas equações (2.14) e (2.15).

Os coeficientes actualizados nesta etapa serão entregues à etapa 3, cada parte dos coeficientes no módulo correspondente completando, desta forma, o ciclo do algoritmo.

(Fim da descrição das etapas de implementação)

Como referido anteriormente, esta implementação tem a vantagem de usar poucos recursos e ser de implementação simples. No entanto, tem uma latência bastante elevada, o que será um factor limitativo para um sistema de altos débitos. Na seguinte Tabela 4.1 consta a informação geral da implementação em termos de ciclos de relógio.

Etapas	**Número de ciclos**
1	Não tem implicação
2	1 para cada endereço
3	1+1 para cada endereço
4	15+1
5	3+1
6	2+1
	$Total = 13 \times 2 = 26$

Tabela 4.1: Latência da configuração sequencial − o ponto crítico é a etapa 4.

O ciclo adicional nas etapas 4, 5 e 6 refere-se ao registo colocado à saída de cada bloco implementado, destes só o da etapa 5 é indispensável.

Portanto, com esta configuração são necessários 26 ciclos de relógio para colocar um símbolo à saída, quando nesse período entram 2 amostras. Esta latência de 26 ciclos é imposta pelo número de coeficientes do filtro 13 e como o são 2 amostras por símbolo originam 26 ciclos de relógios.

Posto isto, implementou-se uma nova configuração do algoritmo CMA, afim de melhorar a latência de execução. Segue-se a configuração paralela para o mesmo algoritmo.

4.2.2 Versão paralela

Nesta configuração desenvolveram-se as mesmas etapas com o objectivo de melhorar o ponto crítico da implementação sequencial descrita anteriormente.

1. Gerar o sinal à entrada do equalizador

A etapa 1 mantém inalterada em relação à configuração anterior, tendo sido a sua descrição feita na página 31.

2. Guardar em memória as 13 posições consecutivas

A solução de implementação no caso anterior para guardar as 13 amostras consecutivas foi um ASR, que coloca os dados em série na saída. Na versão paralela surge a necessidade de criar um bloco que faça o deslocamento das amostras mais antigas para receber as mais recentes e que apresente essas 13 amostras em paralelo no mesmo instante. A solução implementada, apresentada na Figura 4.9, usa 13 registos para guardar os símbolos que serão disponibilizados à entrada do equalizador. Esta implementação gasta apenas 2 ciclos de relógio: um para escrita e o outro para leitura.

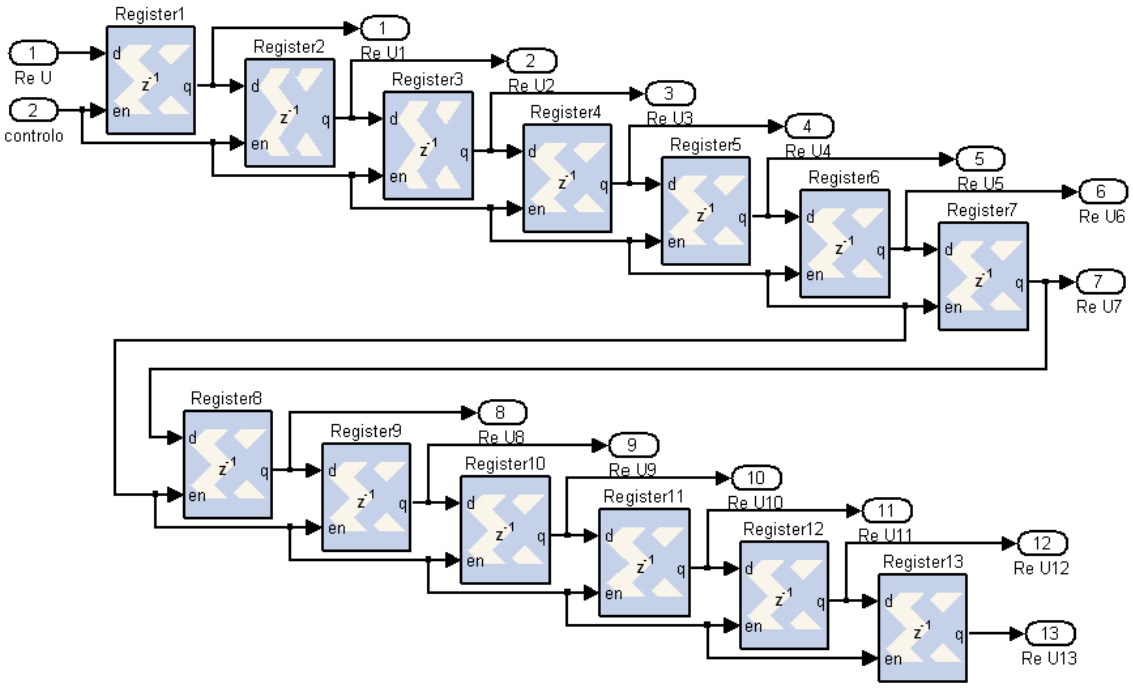


Figura 4.9: *Shift register* paralelo para guardar as amostras.

A Figura 4.9 representa o módulo implementado para a parte real, o que significa que para a parte imaginária é igual, pelo que só se ilustra um como tem vindo a acontecer ao longo deste capítulo da implementação.

3. Alocar 13 posições de memória para os coeficientes

Na etapa 3 foi abandonada a solução da *Dual Port RAM*, uma vez que esta à semelhança do ASR, também apresenta os dados em série. Então, criou-se um módulo com 13 registos e, assim, são armazenados os coeficientes ficando todos disponíveis em paralelo, como se observa na Figura 4.10.

Depois dos coeficientes estarem em sintonia com as amostras e estes serem apresentados em paralelo à etapa 4, surge a necessidade de criar um módulo que execute as operações como na implementação sequencial, mas desta feita, paralelamente.

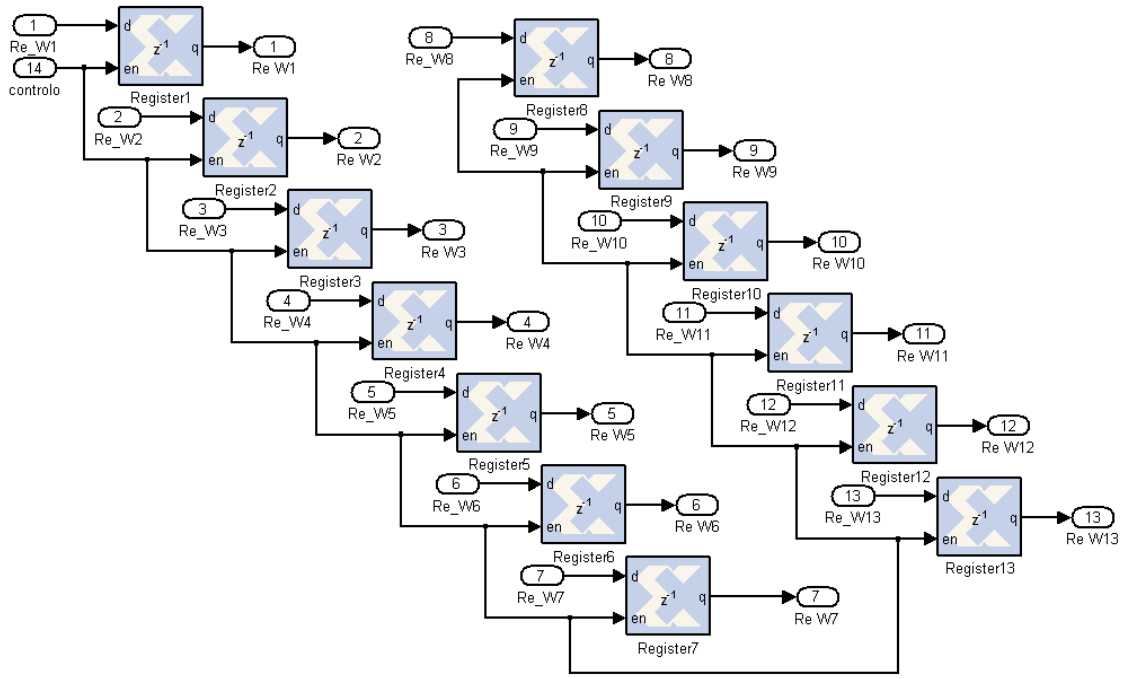


Figura 4.10: Armazenamento dos coeficientes.

4. Realizar a convolução W*U

Esta etapa terá que realizar o cálculo das equações (2.4) e (2.5) para cada um dos 13 valores em simultâneo e depois somar os resultados das multiplicações complexas em forma de árvore binária para, deste modo, se obter o resultado da convolução, quer para a parte real, quer para a parte imaginária. A Figura 4.11 ilustra parte dessa etapa e é bem visível a árvore binária de somadores.

A implementação do filtro em paralelo é custosa, como se pode concluir pela figura, mas é a solução mais praticável já que os dados estão todos disponíveis no mesmo instante. Desta forma, é contornado o ponto crítico da configuração anterior que demorava 16 ciclos e agora passa a executar em 4.

À saída deste bloco é apresentado o símbolo calculado. Posteriormente, é realizado o encaminhamento deste para o bloco *Gateway Out* (para depois ser gerada a constelação) e para o módulo referente à etapa 5.

5. Calcular o erro associado ao símbolo

O cálculo do erro realiza as equações (2.11) e (2.12), tal como na configuração anterior (Figura 4.7). Uma vez que é igual, foi aproveitado e incluído nesta versão sem qualquer alteração.

Para concluir o ciclo de execução do algoritmo, realiza-se a actualização dos coeficientes na etapa 6.

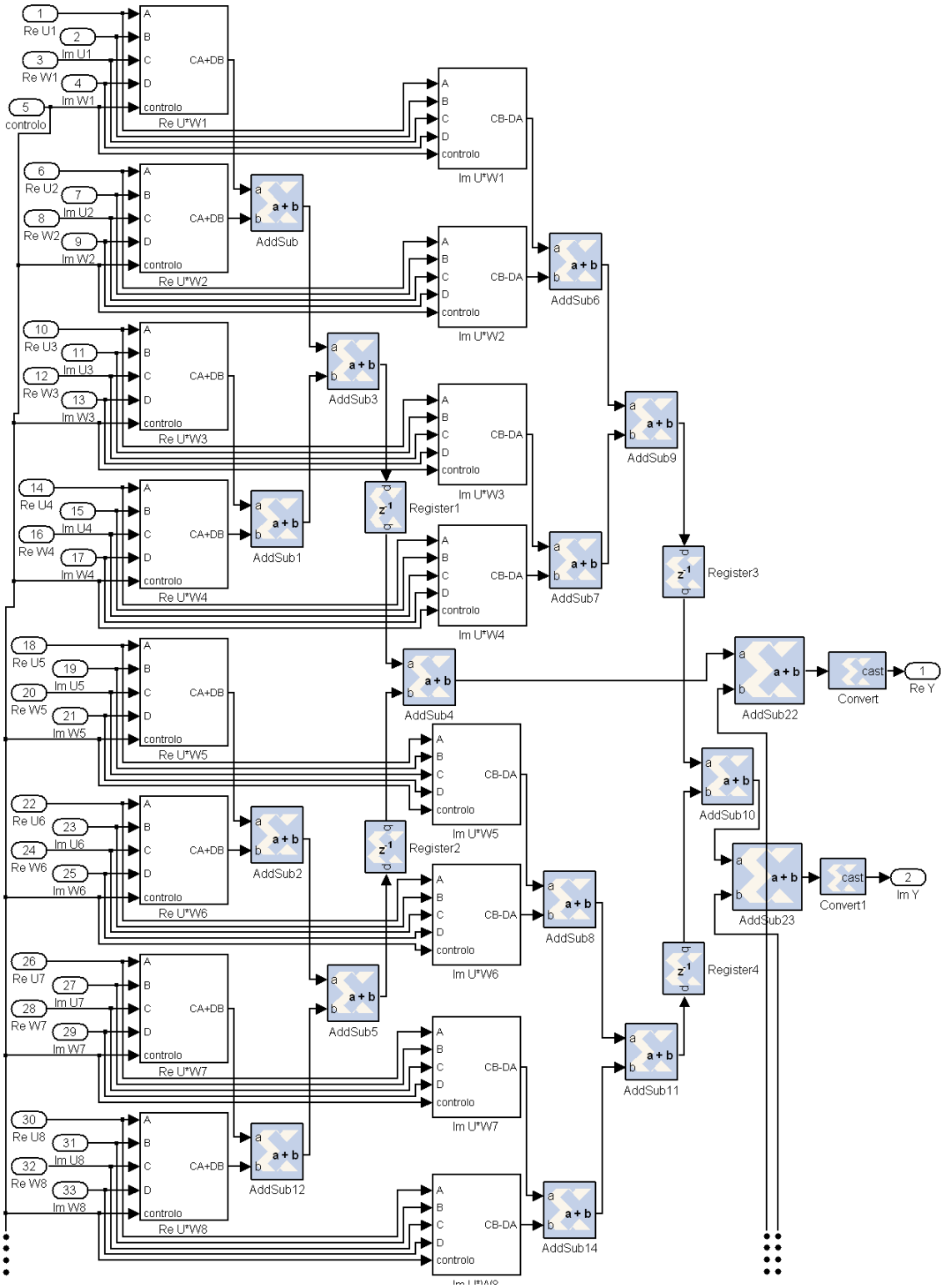


Figura 4.11: Implementação do filtro em paralelo.

6. Actualizar os coeficientes

O processo de actualização dos coeficientes é o mesmo que o apresentado na secção anterior (Figura 4.8), mas desta vez, o módulo é replicado 13× para realizar em paralelo a actualização de cada um dos 13 novos coeficientes. A Figura 4.12 ilustra os cálculos realizados para a actualização dos 3 primeiros coeficientes, neste caso referentes à parte real. Para a parte imaginária procedeu-se à implementação da mesma forma.

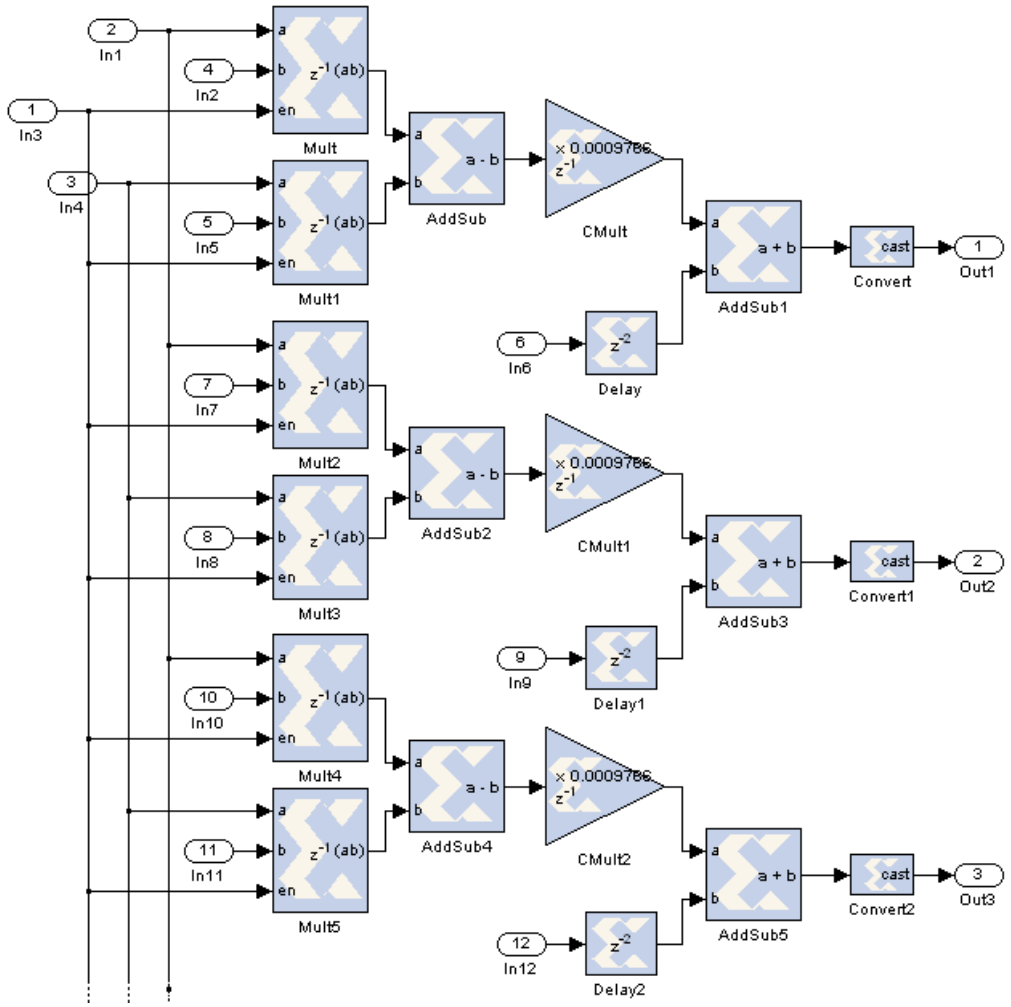


Figura 4.12: Actualização dos coeficientes paralelo.

Depois de actualizados os coeficientes, estes vão ser armazenados nos registos descritos na etapa 3 para ficarem disponíveis para o cálculo do novo símbolo.

(Fim da descrição das etapas de implementação)

Esta implementação, ao contrário da configuração sequencial, tem a desvantagem de usar muitos recursos, como se pode comprovar pela descrição da implementação apresentada. No

entanto, apresenta resultados bastante satisfatórios (Tabela 4.2 seguinte) no que diz respeito ao número total de ciclos de relógio necessários para a execução do algoritmo CMA.

Etapas	**Número de ciclos**
1	Não tem implicação
2	1+1
3	1+1
4	3+1
5	3
6	2
	$Total = 5+5 = 10$

Tabela 4.2: Latência da configuração paralela − o ponto crítico é a etapa 4.

Analisando o número total de ciclos de relógio apresentado na Tabela 4.2, verifica-se que esta implementação permite ter um símbolo na saída do equalizador de 10 em 10 ciclos, sendo apresentadas 2 amostras na entrada nesse mesmo período, isto é, uma amostra à entrada de 5 em 5 ciclos de relógio.

Esta configuração é de realização custosa, mas é excelente a melhoria que se obtém em termos de latência. Com esta implementação para o CMA, reduziu-se a latência de 26 para 10 ciclos, o que melhora em mais de 60% o desempenho do algoritmo em comparação com a primeira configuração implementada.

Estas versões de implementação apresentadas para o equalizador CMA foram desenvolvidas para um sistema de transmissão que usa modulação 4QAM. Porém, estas configurações desenvolvidas podem ser, facilmente, modificadas para a modulação 16QAM, bastando para isso proceder à alteração do valor da constante "R2" do bloco no cálculo do erro (Figura 4.7), de 2 para 1.4667 − valores obtidos com base nas características de cada constelação. Desta forma, o equalizador CMA fica preparado para receber dados provenientes de um sistema com modulação 16QAM.

Depois de explicadas as etapas de implementação do algoritmo CMA, seguem-se as etapas referentes ao algoritmo LMS.

4.3 Etapas da implementação do algoritmo LMS

O que difere o algoritmo LMS do CMA é o modo como calcula o erro associado ao símbolo. Na Figura 4.13 são apresentadas as etapas deste algoritmo onde se destacam uma nova etapa e alteração de outra. Essas etapas são:

1. Realizar a decisão do símbolo.

2. Calcular o erro associado ao símbolo.

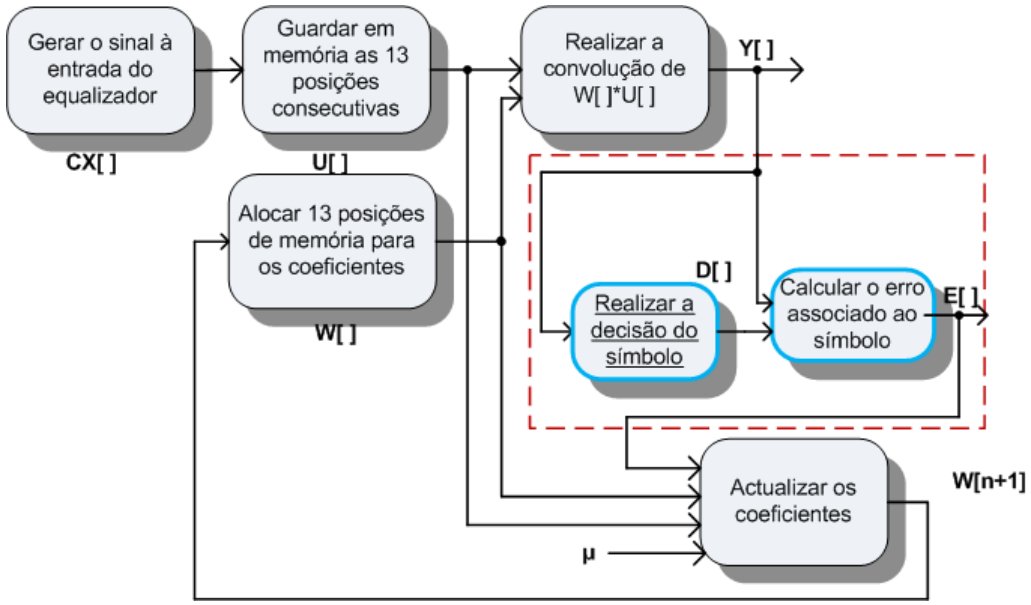


Figura 4.13: Diagrama de blocos do equalizador LMS.

As etapas referidas são as que estão assinaladas na Figura 4.13 na caixa a tracejado, sendo a nova etapa (etapa 1) apresentada a sublinhado e a outra a etapa alterada (etapa 2).

Procede-se agora à descrição de implementação destas duas etapas, uma vez que todas as outras etapas do algoritmo CMA mantêm-se inalteradas, quer para a configuração sequencial, quer para a paralela.

1. Realizar a decisão do símbolo

No desenvolvimento do algoritmo LMS, a dificuldade está na implementação do decisor. Para a decisão do símbolo foi usado inicialmente um bloco do *System Generator* que realiza a decisão com base num valor de referência, isto para o caso 4QAM como mostra a Figura 4.14.

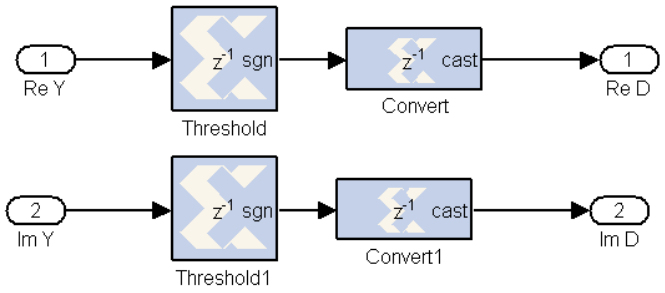


Figura 4.14: Decisor para modulação 4QAM.

O bloco *threshold* simplesmente compara o símbolo calculado pela etapa que realiza a convolução com o valor de referência que é zero. Caso o valor do símbolo seja menor que zero, ou seja, negativo, este bloco coloca na saída o valor '-1', caso contrário apresenta o valor '1'. Posto isto, tem-se um decisor 4QAM simples e prático.

Para a modulação 16QAM foi necessário implementar um módulo para realizar a decisão do símbolo. Esse módulo também funciona, em teoria, para 4QAM e é apresentado na Figura 4.15.

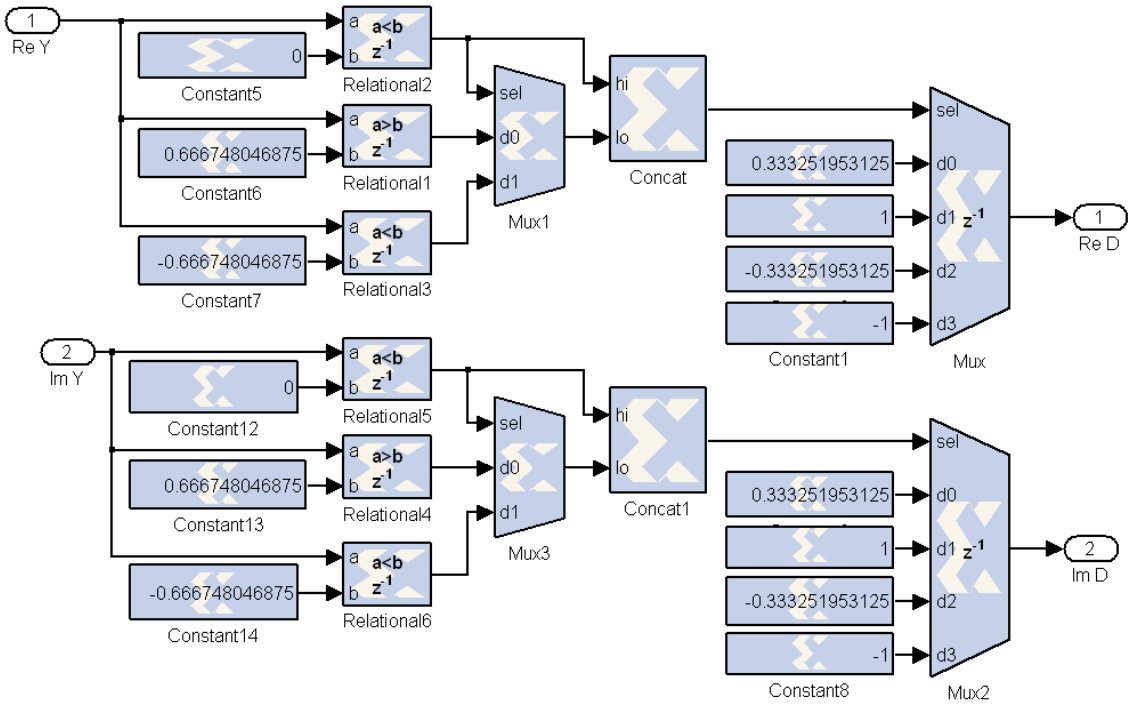


Figura 4.15: Decisor 16QAM.

O que está implícito neste processo é verificar a que zona de decisão da constelação 16QAM (Figura 4.16) o símbolo pertence.

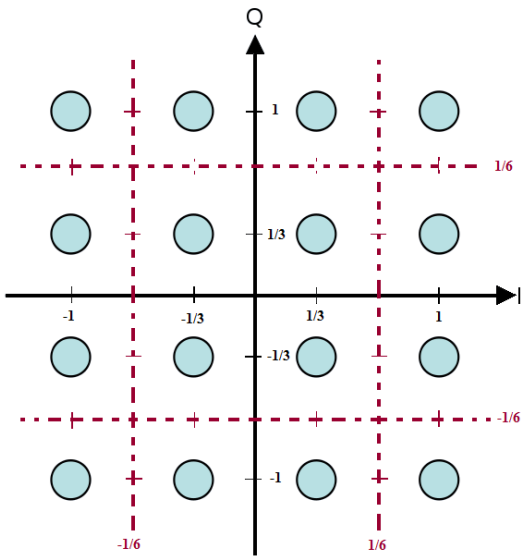


Figura 4.16: Zonas de decisão da constelação 16QAM.

Mediante as comparações realizadas o módulo decide por um valor da constelação, isto é, relaciona as comparações para seleccionar um dos 4 valores possíveis do "Mux" da Figura 4.15.

Estas soluções adaptadas não obtiveram desempenho satisfatório perante os resultados que apresentaram, isto devido à tomada de decisão da parte real ser independente da parte imaginária e vice-versa. Houve a necessidade de implementar a solução praticada nos algoritmos desenvolvidos em MATLAB, cuja implementação para o caso 16QAM é de maior complexidade e de maior gasto de recursos da FPGA.

Em MATLAB o decisor realiza as seguintes operações:

$$min(constparam - \Re e(Y \times conjconst)) \tag{4.1}$$

$$constparam = \frac{1}{2}(sigconst \times conjconst) \tag{4.2}$$

e $conjconst$ é o conjugado do sinal da constelação $(sigconst)$. Por exemplo, para 4QAM:

$$sigconst = \begin{bmatrix} -1+j & -1-j & +1+j & +1-j \end{bmatrix}$$

$$conjconst = \begin{bmatrix} -1-j & -1+j & +1-j & +1+j \end{bmatrix}$$

$$constparam = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Segue-se a implementação do novo decisor 4QAM desenvolvido, cuja Figura 4.17 apresenta essa imprementação que realiza as operações descritas em (4.1) e (4.2).

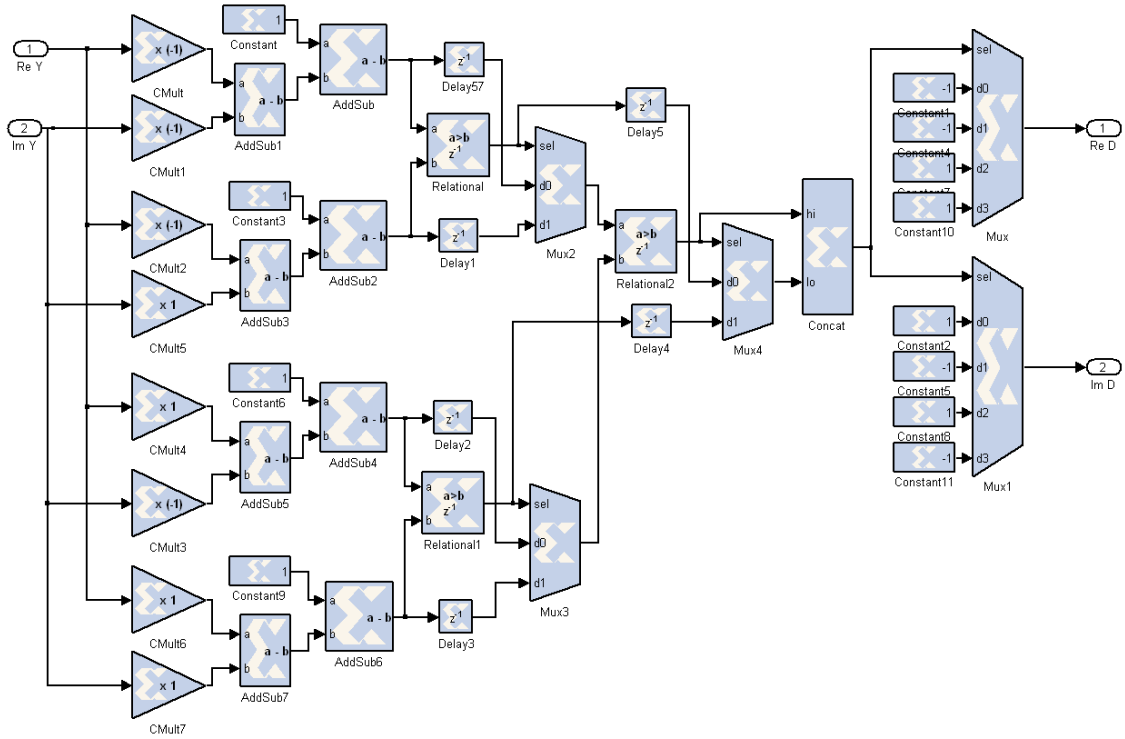


Figura 4.17: Novo decisor para a modulação 4QAM.

Para encontrar o mínimo em 4QAM é necessária toda esta complexidade, o que permite prever

que para 16QAM a complexidade vai ser maior. Portanto, o decisor 4QAM, decide pelo valor da constelação (*sigconst*) associado ao valor mínimo que resulta da operação (4.1).

Para 16QAM o decisor apresenta maior latência, pois necessita de realizar mais comparações para escolher o mínimo de 16 valores. Isto implica que se proceda a reajustes na contabilização dos ciclos de relógio do algoritmo.

A implementação deste decisor é bastante complexa exigindo que se realizem testes de validação para comprovar que o funcionamento é o pretendido. Os testes realizaram-se da seguinte forma: colocou-se uma constante com valor inferior a todas as outras e, com o bloco *Display* do *Simulink*, verifica-se se o valor seleccionado pelo "Mux" é ou não o correspondente à constante mínima. Perante isto, pode-se comprovar que o decisor funciona correctamente para todos os 16 valores que cada *multiplexer* coloca à saída deste bloco de decisão.

2. Calcular o erro associado ao símbolo

Após a decisão do símbolo, é realizado o cálculo do erro segundo as equações (2.17) e (2.18) anteriormente apresentadas. Esta etapa, simplesmente, realiza a subtracção do símbolo detectado ("Re D" e "Im D") com o símbolo calculado ("Re Y" e "Im Y"), como se pode observar na Figura 4.18.

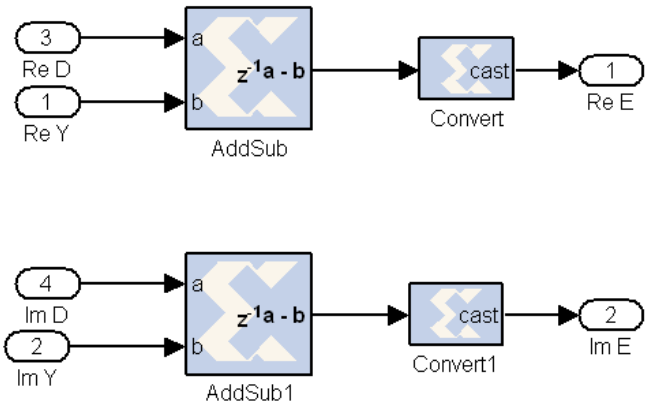


Figura 4.18: Cálculo do erro no algoritmo LMS.

Estas duas etapas descritas, anteriormente, para o LMS são incluídas num módulo único chamado "Calculo do erro LMS" representado na Figura 4.19. Este substitui o módulo cálculo do erro na implementação do CMA, quer na configuração sequencial, quer na paralela.

Para o algoritmo LMS funcionar é necessário proceder à inicialização dos coeficientes. Uma das formas de o fazer é executar o CMA até este atingir a convergência e depois usar o valor dos coeficientes nesse instante para a inicialização.

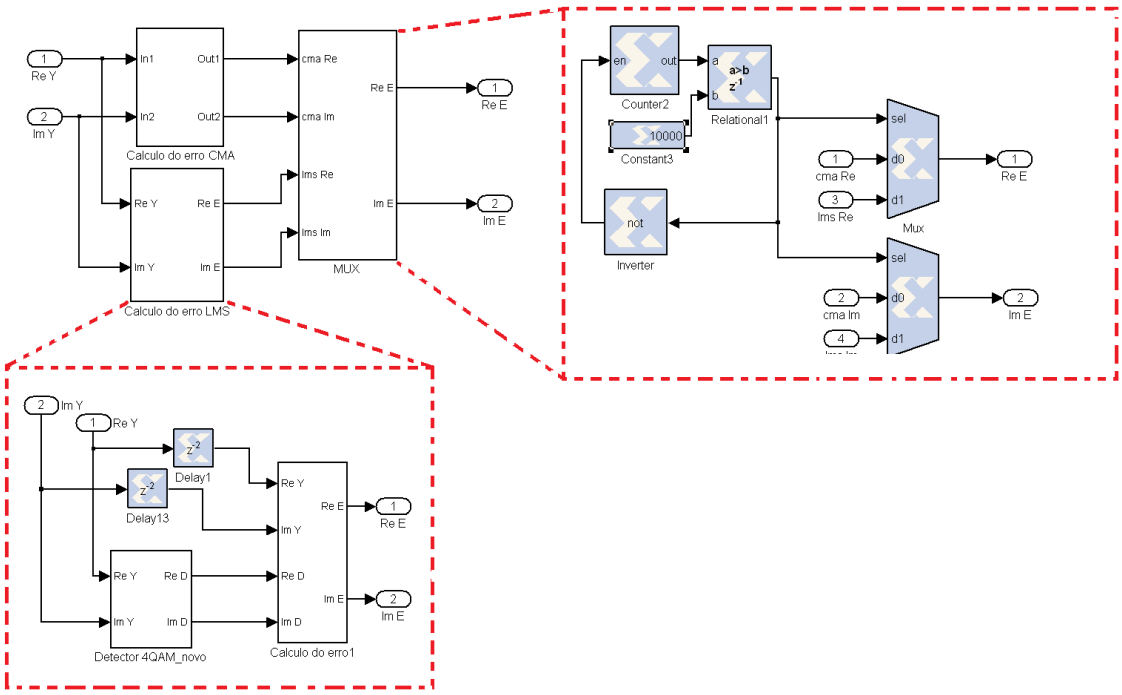


Figura 4.19: Cálculo do erro dos algoritmos LMS e CMA, à esquerda; implementação da inicialização dos coeficientes para o algoritmo LMS, à direita.

Uma maneira prática de resolver esta questão é a implementação sugerida pela Figura 4.19, que consiste em ter os dois algoritmos em funcionamento, ocorrendo uma comutação (após atingir a convergência) do algoritmo CMA para o LMS no instante indicado à direita na Figura.

Após implementação dos algoritmos e assegurado o seu bom funcionamento, partiu-se para a implementação do módulo para compensar a dispersão cromática da fibra óptica a usar no sistema de transmissão.

4.4 Módulo para compensar a dispersão

A compensação da dispersão é conseguida através de um filtro implementado exactamente antes dos algoritmos (Figura 4.20), cujas características são inversas às da função de transferência da fibra óptica, como já descrito na secção 2.6.

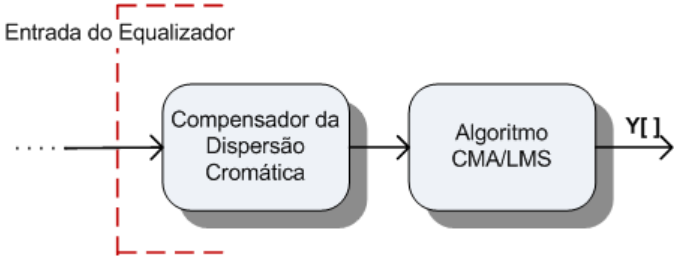
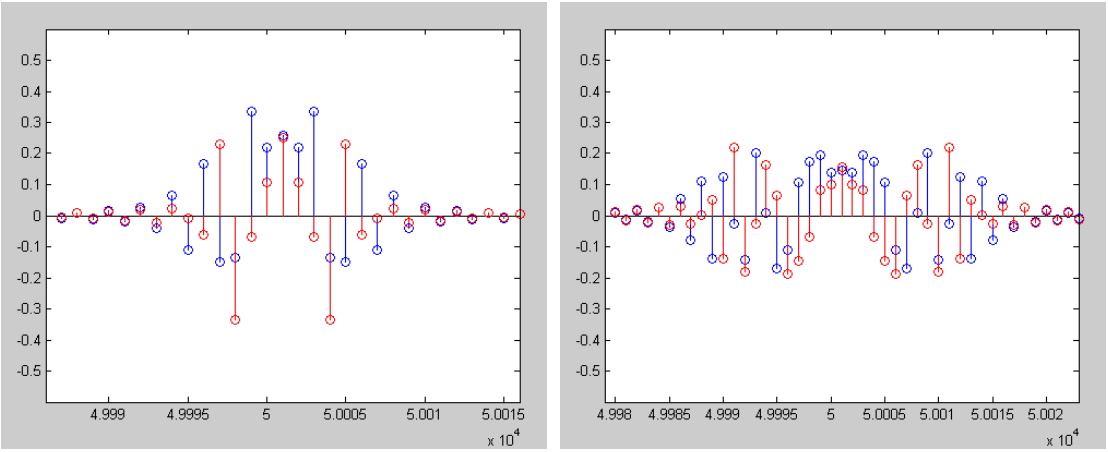


Figura 4.20: Localização do compensador da dispersão no sistema.

A implementação desse filtro é realizada através da convolução do sinal de entrada com os coeficientes fixos. Estes coeficientes fixos são obtidos, realizando a Transformada Inversa de *Fourier* (IFFT) da função de transferência inversa da fibra óptica. Após realizada a IFFT obtém-se as seguintes características do filtro apresentadas nas Figuras 4.21(a) e 4.21(b), onde cada valor representa um coeficiente: cor azul a parte real e a cor vermelha a parte imaginária.



(a) Referentes a 200 km (b) Referentes a 500 km

Figura 4.21: Coeficientes fixos para a implementação do compensador de dispersão.

São estes os coeficientes fixos usados para o desenvolvimento do módulo do compensador. Como o número de coeficientes é muito grande e a maior parte deles são nulos ou próximo de zero, faz-se aqui uma selecção dos valores com maior energia, sensivelmente 90%, a fim de diminuir a complexidade do módulo.

Procedeu-se à implementação de um módulo para compensar 200 km de fibra óptica e outro módulo para 500 km. Para 200 km usaram-se os 13 coeficientes do centro da Figura 4.21(a), enquanto que, para 500 km se usaram os 31 coeficientes centrais da Figura 4.21(b).

O módulo do compensador realiza todas as operações de multiplicação em paralelo, à semelhança do filtro dos algoritmos atrás descrito na configuração paralela (Figura 4.11), realizando a seguinte operação:

$$u_c(n) = \mathbf{f}^T(n) \cdot \mathbf{u}(n) \tag{4.3}$$

ou então:

$$Re\ U_c = CA - DB \tag{4.4}$$

$$Im\ U_c = DA + CB \tag{4.5}$$

Portanto, quanto maior for a distância da fibra a compensar, maior será o número de coeficientes a usar no filtro e maior será também a complexidade do módulo, nomeadamente na árvore binária de somadores.

Para concluir, é apresentado o conceito de paralelismo dos algoritmos a fim de tornar mais rápida a sua execução.

4.5 Paralelismo do CMA/LMS

Nesta implementação o conceito de paralelismo foi empregue num sistema em que o processamento dos símbolos é realizado em paralelo, isto é, utilizando vários módulos com implementações quer do CMA, quer do LMS, consegue-se reduzir a latência de cada algoritmo. Estes podem apresentar na saída um símbolo a cada dois ciclos de relógio, reduzindo grandemente a latência de execução. Cada módulo irá ter os seus próprios coeficientes que serão actualizados por cada algoritmo colocado em paralelo.

Esta implementação é muito dispendiosa em termos de recursos usados, pelo que é necessária uma FPGA com muitos recursos e de preferência com abundantes multiplicadores dedicados.

4.6 Resumo

Neste capítulo foram apresentadas as etapas da implementação do algoritmo CMA, quer para a configuração sequencial, quer para a configuração paralela. Foram apresentadas as descrições de implementação de cada uma das etapas. O mesmo foi discutido para o algoritmo LMS.

Depois apresentou-se a implementação do módulo para compensar a dispersão e o conceito de paralelismo foi aplicado para os dois algoritmos CMA/LMS a fim de reduzir o tempo de execução.

No capítulo 5 que se segue, são apresentados os resultados obtidos das simulações em *System Generator* e em hardware. É realizada a comparação dos resultados para avaliar o desempenho dos algoritmos implementados e feito o estudo dos limites do sistema com estes equalizadores.

Por fim, é apresentada uma possível plataforma FPGA para a realização do sistema.

Capítulo 5

Resultados obtidos e implementação em FPGA

5.1 Introdução

Neste capítulo são apresentados os resultados obtidos das simulações em *System Generator* e em hardware para cada um dos algoritmos implementados. As simulações em hardware são possíveis através da co-simulação, funcionalidade disponibilizada pelo *System Generator* que permite validar em hardware os algoritmos CMA e LMS.

Os resultados são apresentados na forma de constelação para cada formato de modulação: para 4QAM − onde os símbolos são apresentados em volta de 4 pontos numa grelha quadrada − e para 16QAM − onde a grelha quadrada contém 16 pontos, formando a constelação 16QAM.

É realizada a comparação dos resultados para avaliar o desempenho do algoritmos implementados. Essa comparação é feita com os resultados obtidos em *System Generator*, em MATLAB e em hardware.

O estudo dos limites do sistema com os equalizadores adaptativos implementados é feito na secção 5.4.

Por fim, é apresentada uma possível plataforma FPGA para a realização de um sistema prático.

5.2 Resultados da implementação

Após a implementação, simularam-se os algoritmos CMA e LMS, a fim de realizar a medição em termos de ocupação de recursos, resultando a Tabela 5.1.

Salta logo à vista que a configuração em paralelo ocupa cerca de 6 vezes mais recursos que a configuração sequencial e no que se refere a multiplicadores dedicados a sua utilização é muito superior.

Recursos	**Sequencial**	**Paralela**
Slices	1070	6944
FFs	939	5285
BRAMs	2	0
LUTs	1758	9933
IOBs	108	108
Emb. Mults.	14	60
Características		
Máx. Frequência	57,917 MHz	97,144 MHz
Min. Período	17,266 ns	10,294 ns
Tempos mínimos		
Latência	26 períodos	10 períodos
Tempo de símbolo	448,916 ns	102,94 ns
Taxa de transmissão	2,2 MSímbolos/s	9,7 MSímbolos/s

Tabela 5.1: Ocupação e características das configurações sequencial e paralela, para a Virtex 4 XC4VLX60-10ff672.

De facto, esses recursos (multiplicadores dedicados) não são muito abundantes nas FPGAs, o que poderá ser um factor limitativo do sistema, sobrando a alternativa em realizar as multiplicações em memória distribuída, o que demora mais tempo. Outra diferença entre as configurações é o uso de BRAMs por parte da configuração sequencial, para o armazenamento dos coeficientes, não causando grande problema uma vez que existem em número razoável.

Apresentados os valores para a Virtex 4 (xc4vfx60-10ff672) conclui-se que o sistema em paralelo é mais rápido na execução, permitindo uma taxa mais elevada: 9,7 MSímbolos/s contra 2,2 MSímbolos/s da versão sequencial.

Apresentam-se de seguida os resultados que avaliam o desempenho dos algoritmos, para cada um dos formatos de modulação. São apresentadas as constelações 4QAM e 16QAM, realizando-se uma análise qualitativa do sistema.

5.3 Resultados das simulações

Após os resultados do desenvolvimento dos algoritmos nas duas configurações, apresentam-se agora os resultados das simulações do sistema de transmissão.

Para os resultados apresentados em seguida, foi usada a codificação de impulsos NRZ, obtidos por meio de um *train* rectangular de impulsos ideais através de um filtro de *Bessel* passa baixo de 5ª ordem com 3dB de largura de banda, a 80% da taxa de transmissão dos símbolos. É usado também um filtro *anti-alias*, constituido por um filtro de *Bessel* passa baixo de 3ª ordem.

Apresentam-se os resultados obtidos para a modulação 4QAM e posteriormente 16QAM, onde se evidencia o desempenho de cada algoritmo.

5.3.1 Desempenho dos algoritmos

Modulação 4QAM em *System Generator*

Simulando os algoritmos implementados em *System Generator* obtém-se o conjunto de constelações da Figura 5.1 que se segue para 4QAM. Esse conjunto de constelações apresenta o desempenho dos algoritmos CMA e LMS para o sistema de transmissão na ausência da dispersão conseguindo-se, deste modo, uma avaliação isolada do desempenho dos algoritmos.

Na Figura 5.1(a) é apresentada a constelação 4QAM do sinal à entrada do equalizador, sendo esta constelação obtida sem qualquer intervenção por parte dos algoritmos. Já as Figuras 5.1(b) e 5.1(c) apresentam as constelações dos algoritmos CMA e LMS respectivamente, após o funcionamento dos mesmos.

Pela análise das Figuras 5.1(b) e 5.1(c) verifica-se que os algoritmos convergem e apresentam bons resultados, uma vez que na constelação da Figura 5.1(a) existe ruído, que é anulado pelo bom funcionamento dos respectivos algoritmos como evidenciam as Figuras 5.1(b) e 5.1(c).

Verifica-se que as constelações geradas após o funcionamento dos algoritmos apresentam a forma esperada, isto é, são exibidas com a nuvem de símbolos concentrada em volta dos pontos de decisão, pelo que é óptimo este resultado, pois reflecte que ocorrem poucos erros no sistema de transmissão.

Esta é uma análise qualitativa e é possível realizar uma análise quantitativa medindo o desvio padrão dos símbolos à volta de cada valor da constelação. Através desse desvio padrão é possível quantificar o desempenho dos algoritmos, no entanto, uma análise qualitativa é suficiente para avaliar o bom funcionamento dos algoritmos.

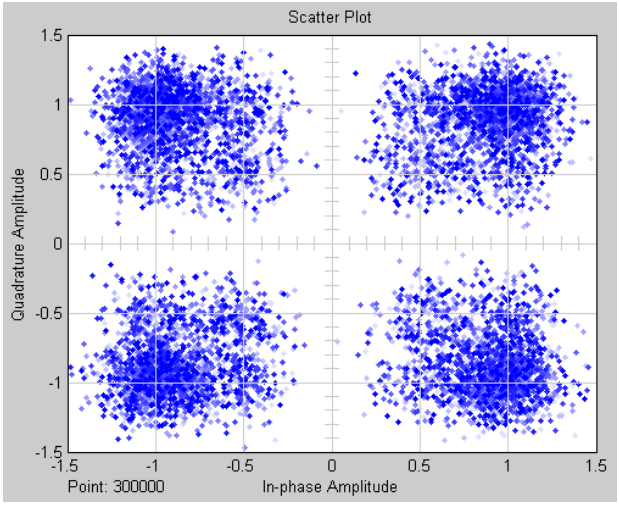
Modulação 16QAM em *System Generator*

Para a modulação 16QAM procedeu-se da mesma forma e os resultados obtidos são apresentados na Figura 5.2.

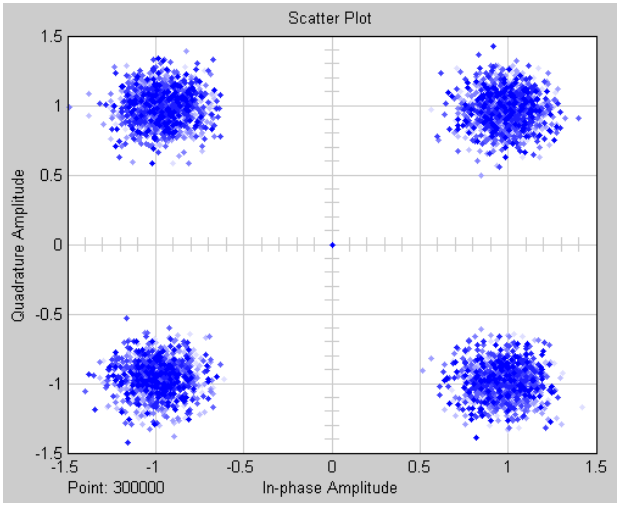
Tal como sucedido no caso da modulação 4QAM, também nos resultados da modulação 16QAM verifica-se o bom desempenho apresentado pelos os algoritmos, uma vez que conseguem retirar o ruído que os símbolos contêm à entrada do equalizador (Figura 5.2(a)), apresentando o algoritmo CMA uma constelação com os símbolos concentrados em torno dos 16 pontos possíveis da constelação 16QAM, como mostra a Figura 5.2(b).

O mesmo acontece com o algoritmo LMS que apresenta a constelação da Figura 5.2(c) como bom resultado do seu desempenho. Perante estes resultados há a necessidade de os confirmar com os resultados obtidos em MATLAB.

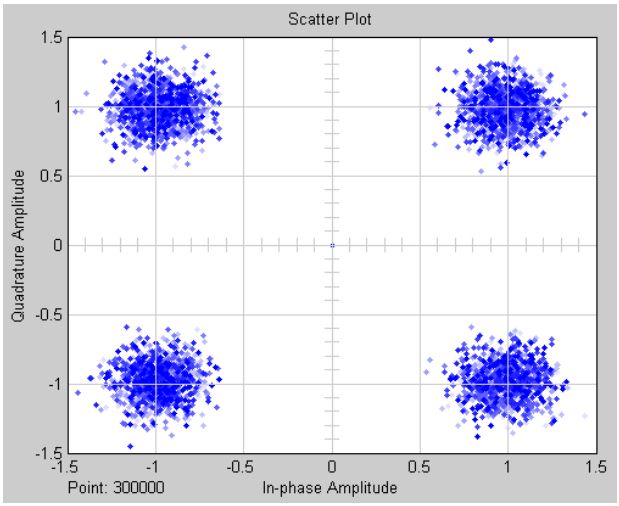
Portanto, seguem-se os resultados obtidos em MATLAB para realizar uma comparação e demonstração do desempenho dos algoritmos.



(a) Entrada do equalizador

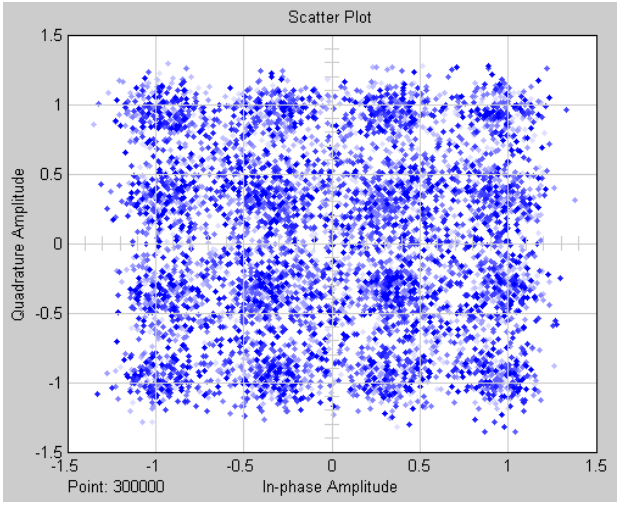


(b) Resultado após execução do CMA

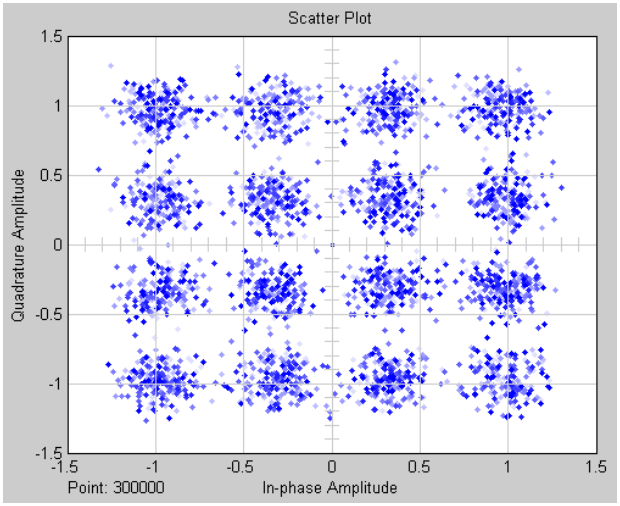


(c) Resultado após execução do LMS

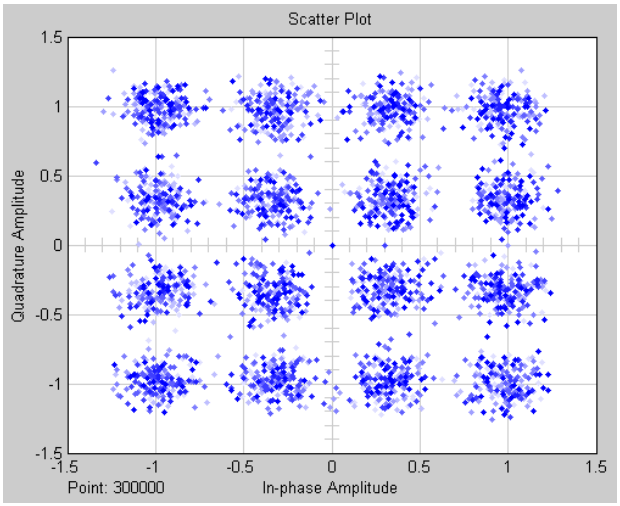
Figura 5.1: Desempenho dos algoritmos para a modulação 4QAM.



(a) Entrada do equalizador



(b) Resultado após execução do CMA

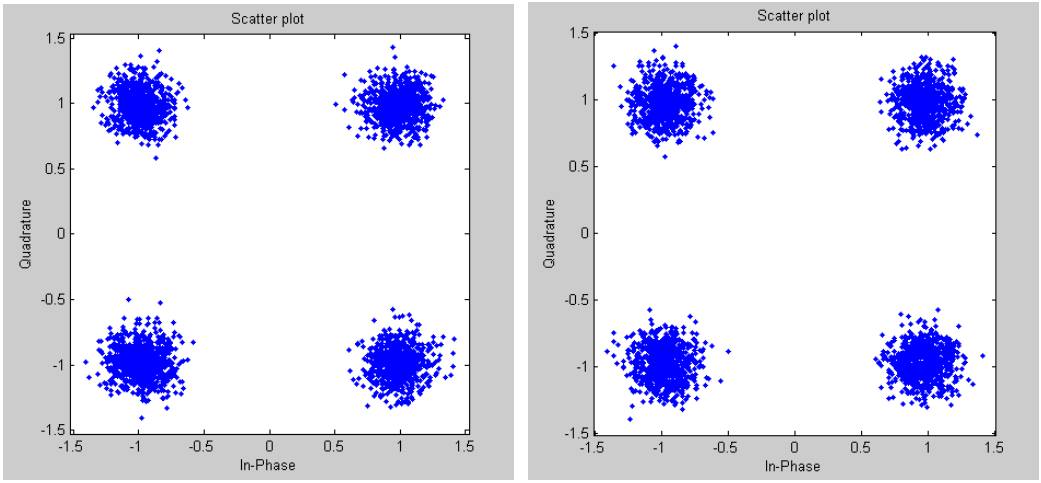


(c) Resultado após execução do LMS

Figura 5.2: Desempenho dos algoritmos para a modulação 16QAM.

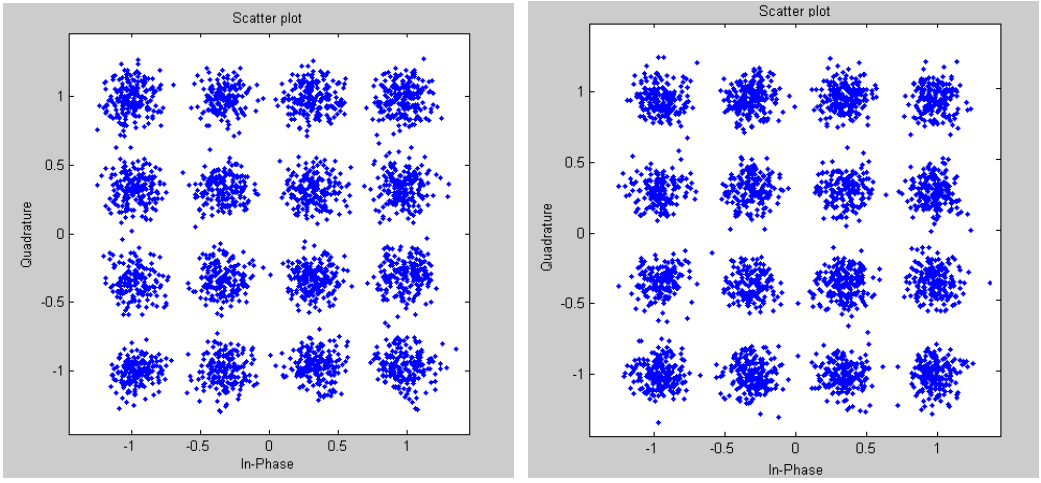
Resultados em MATLAB para modulação 4QAM e 16QAM

Estes resultados (Figuras 5.3 e 5.4) foram obtidos, através dos algoritmos já desenvolvidos em MATLAB, com o objectivo de ter um modelo de comparação e validação dos resultados para os algoritmos implementados. Esta forma de validação é vantajosa e permite ao utilizador ter a percepção se o caminho que está a tomar na implementação é o correcto ou não.



(a) Resultado após execução do CMA (b) Resultado após execução do LMS

Figura 5.3: Resultados MATLAB dos algoritmos para a modulação 4QAM.



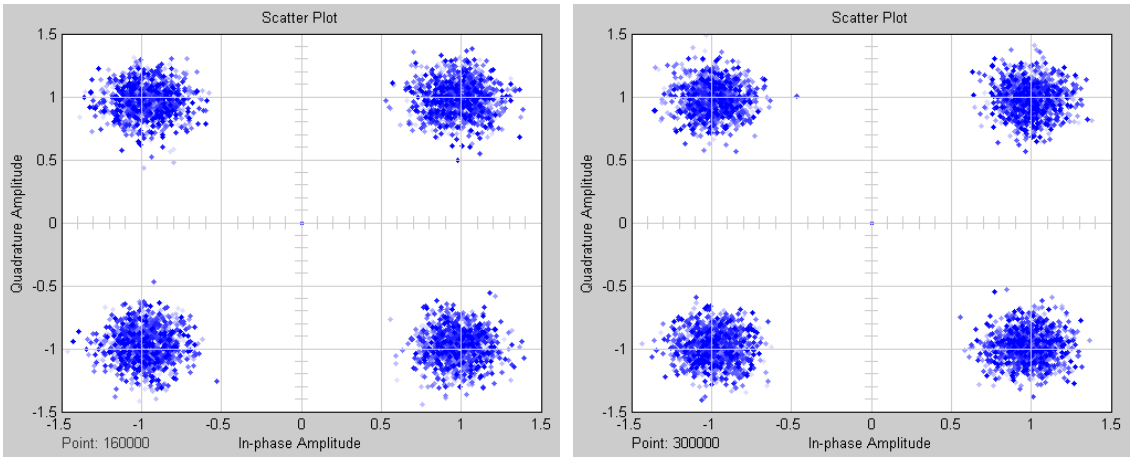
(a) Resultado após execução do CMA (b) Resultado após execução do LMS

Figura 5.4: Resultados MATLAB dos algoritmos para a modulação 16QAM.

Posto isto, comparando os resultados obtidos em *System Generator* (Figuras 5.1 e 5.2) com os resultados obtidos em MATLAB (Figuras 5.3 e 5.4) pode-se comprovar, quer para o algoritmo CMA quer para o LMS, que estes algoritmos apresentam o desempenho desejado para ambas as modulações 4QAM e 16QAM, como era esperado.

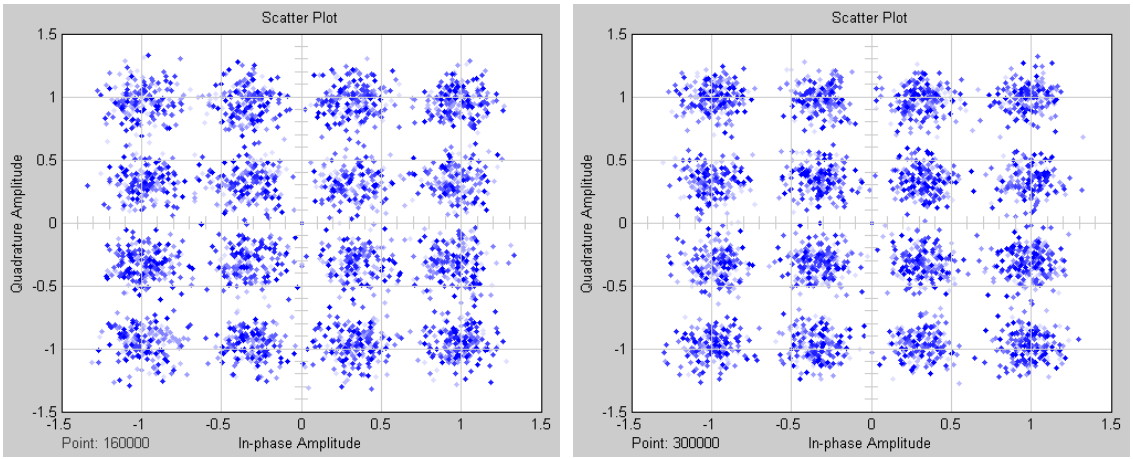
Resultados em hardware para modulação 4QAM e 16QAM

As constelações das Figuras 5.5 e 5.6 resultaram da co-simulação realizada na FPGA Virtex 4. Com este procedimento é possível ter uma validação do funcionamento dos algoritmos em hardware. Foi necessário realizar um esforço em termos de multiplicadores para que a implementação fosse realizável em co-simulação, uma vez que esta FPGA contém um número inferior ao necessário. O procedimento tomado for distribuir os multiplicadores em excesso por LUTs.



(a) Resultado após execução do CMA (b) Resultado após execução do LMS

Figura 5.5: Resultados co-simulação dos algoritmos para a modulação 4QAM.



(a) Resultado após execução do CMA (b) Resultado após execução do LMS

Figura 5.6: Resultados co-simulação dos algoritmos para a modulação 16QAM.

Realizando a comparação destes resultados obtidos em co-simulação (Figuras 5.5 e 5.6) com os resultados obtidos em *System Generator* (Figuras 5.1 e 5.2) verifica-se o bom funcionamento apresentado pelos algoritmos. Verificada esta situação é validado o sistema em hardware na plataforma FPGA Virtex 4 como se previa.

5.3.2 Desempenho extra dos algoritmos

Modulação 4QAM com 200 km de fibra óptica

Realizaram-se, após o correcto funcionamento dos algoritmos adaptativos, simulações com os símbolos a passarem no canal dispersivo que é a fibra óptica. O tipo de fibra em estudo foi a fibra standard mono modo (SSMF), cuja dispersão (D_0) é 17 ps/nm/km.

Os algoritmos foram testados para a distância de 200 km de fibra óptica e os resultados são apresentados no conjunto de constelações da Figura 5.7.

Na Figura 5.7(a), os símbolos ao passarem pelo canal ficam desorganizados e dispersos pela constelação originando elevadíssimos erros, o que torna impraticável qualquer sistema de comunicação.

Mas, observando as Figuras 5.7(b) e 5.7(c), verifica-se que estes algoritmos são capazes de compensar dispersão cromática até 200 km apresentando um desempenho satisfatório perante esta distância de fibra óptica.

O desempenho do LMS é melhor que o do CMA na medida em que os símbolos aparecem em torno dos pontos da constelação. No equalizador que usa o CMA a constelação aparece rodada, dado que este apresenta um desinteresse pela fase convergindo para um módulo constante ignorando qualquer alteração de fase (efeito provocado pela dispersão cromática). A consequência desse desinteresse pela fase é apresentar uma constelação rodada, como a observada na Figura 5.7(b).

Já a característica do LMS é convergir para um valor mínimo utilizando um decisor. Desta forma apresenta bons resultados e um óptimo desempenho, como é ilustrado na Figura 5.7(c).

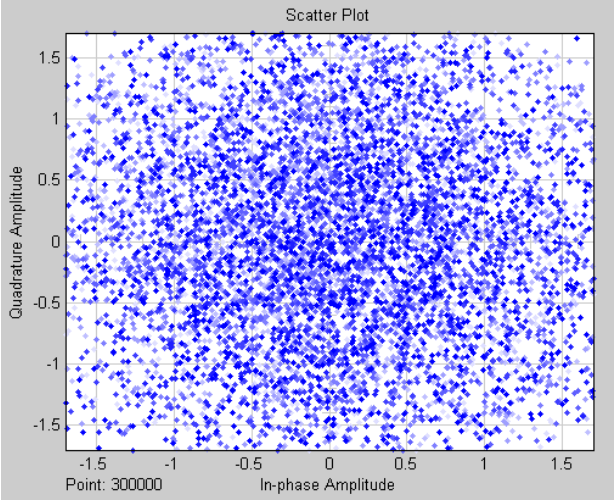
Modulação 16QAM com 200 km de fibra óptica

Para a distância de 200 km, o sistema com modulação 16QAM apresenta algumas falhas, como se observa pela Figura 5.8.

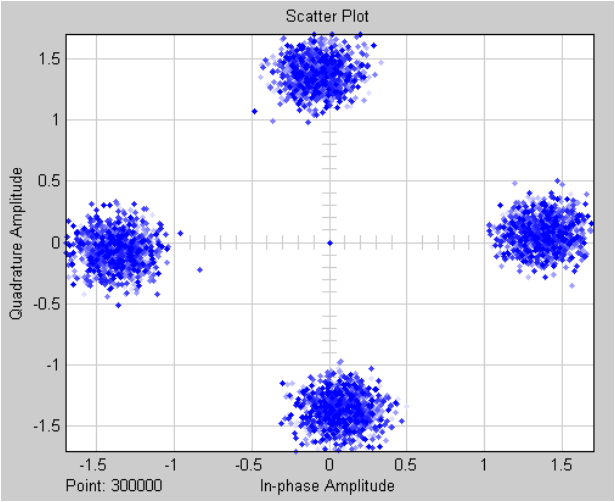
Verifica-se que o CMA (Figura 5.8(b)) apresenta a constelação rodada devido ao desvio de fase provocado pela dispersão. A dispersão provoca ainda uma demora na convergência deste algoritmo, o que pode tornar-se num problema, uma vez que se perdem dados enquanto não se atingir essa convergência. Já o funcionamento do LMS é prejudicado pela alteração da fase verificando-se pelo resultado da Figura 5.8(c) que não consegue apresentar a constelação 16QAM na forma de grelha quadrada de 16 pontos definindo um quadrado.

Daqui surge a necessidade de implementar o compensador da dispersão, para que a convergência do CMA seja mais rápida e o LMS consiga responder mais eficazmente.

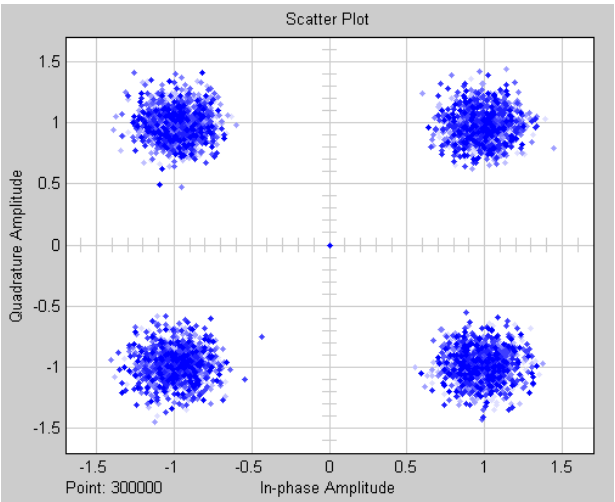
Os dados que se seguem foram obtidos usando o compensador desenvolvido, primeiro para 200 km e depois para 500 km.



(a) Entrada do equalizador

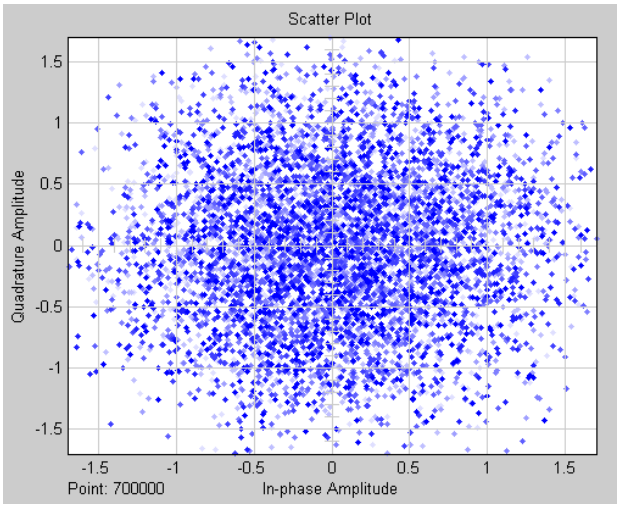


(b) Resultado após execução do CMA

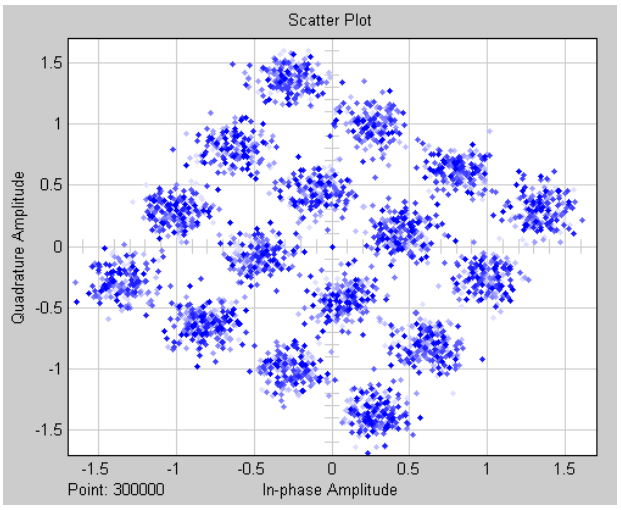


(c) Resultado após execução do LMS

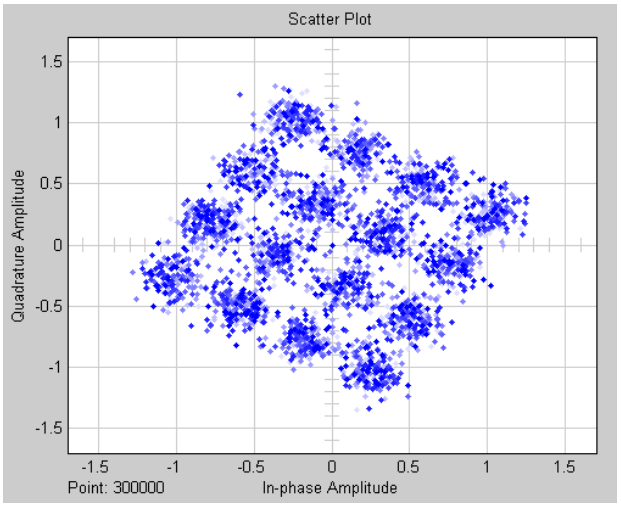
Figura 5.7: Desempenho dos algoritmos para a modulação 4QAM para 200 km.



(a) Entrada do equalizador



(b) Resultado após execução do CMA



(c) Resultado após execução do LMS

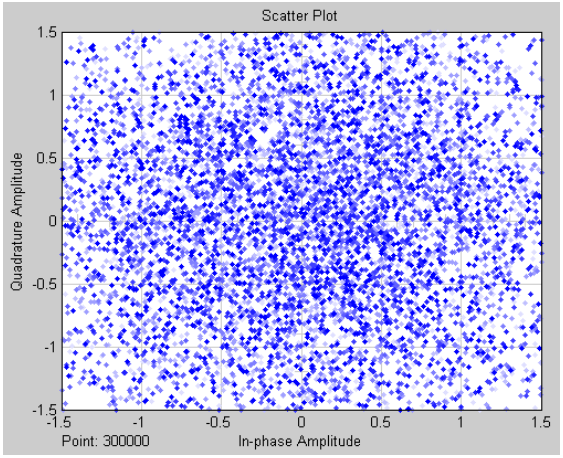
Figura 5.8: Desempenho dos algoritmos para a modulação 16QAM para 200 km.

5.3.3 Resultados obtidos com compensador

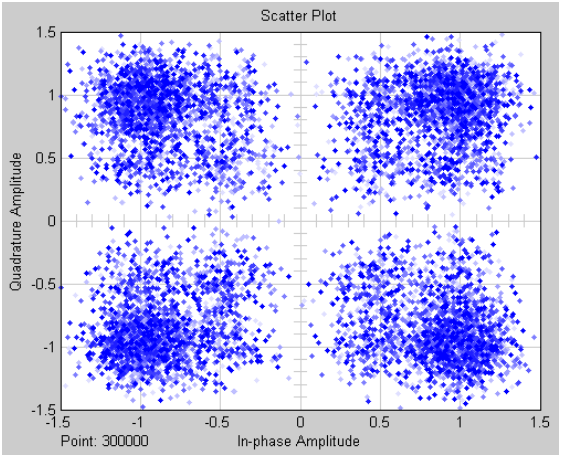
Compensador de 200 km para modulação 4QAM

Estes algoritmos adaptativos conseguem, por si só, compensar a dispersão cromática da fibra óptica até 200 km, como já foi verificado por exemplo para 4QAM (Figura 5.7). Isto deve-se ao facto de o número de coeficientes usados serem suficientes para tal vantagem extra. No entanto, para a modulação 16QAM os resultados não foram satisfatórios, nomeadamente para o LMS (Figura 5.8), pelo que se desenvolveu um módulo para compensar a dispersão cromática que a fibra impõem ao sistema de transmissão.

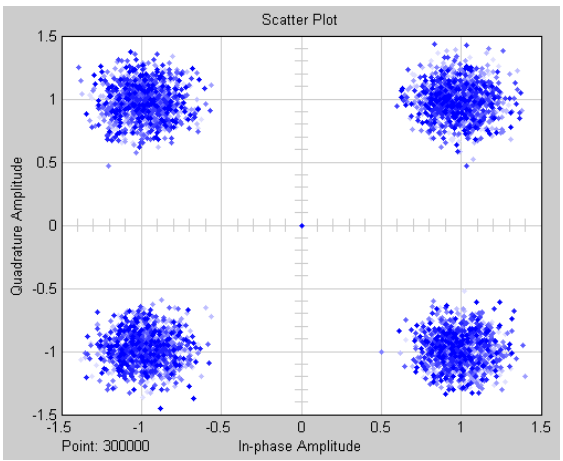
Portanto, para compensar 200 km de distância o compensador necessita de 13 coeficientes fixos, sendo os resultados apresentados na Figura 5.9.



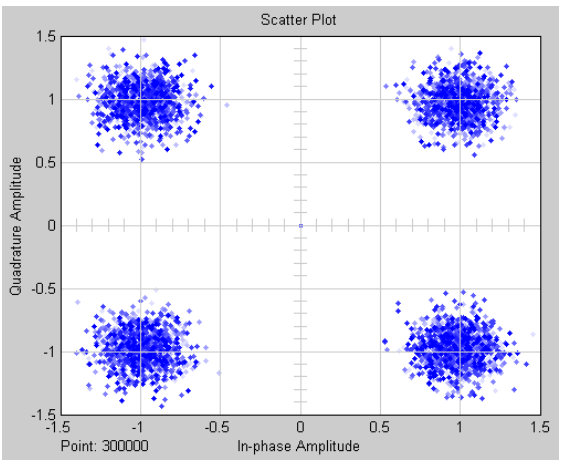
(a) Entrada do equalizador



(b) Saída do compensador



(c) Resultado após execução do CMA



(d) Resultado após execução do LMS

Figura 5.9: Desempenho dos algoritmos para a modulação 4QAM com compensador 200 km.

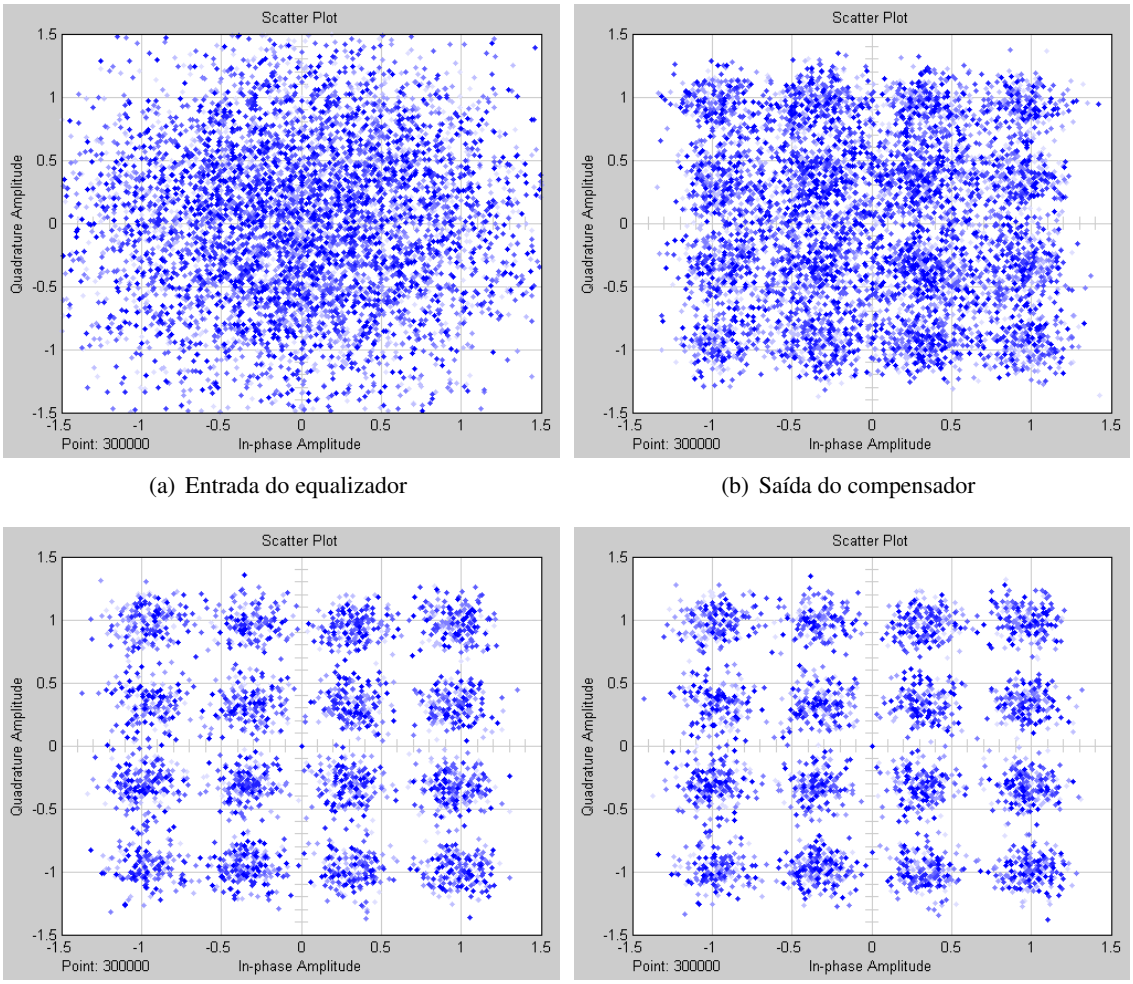
A Figura 5.9(b) indica que o compensador está a funcionar correctamente, uma vez que a constelação 4QAM apresentada é muito semelhante à da Figura 5.1(a) exibida nos resultados anteriores perante a ausência de fibra óptica (dispersão). É notável a melhoria significativa que o

compensador produz, como ilustra a passagem da Figura 5.9(a) para a Figura 5.9(b). Esta melhoria faz com que o algoritmo CMA tenha um melhor desempenho em comparação com o caso em que não possuía o compensador em causa (Figura 5.7).

O algoritmo LMS continua com o bom desempenho, como acontecia anteriormente verificado na Figura 5.7(c), apresentando uma boa convergência.

Compensador de 200 km para modulação 16QAM

Foi utilizado o mesmo compensador, mas agora para a modulação 16QAM e a Figura 5.10 apresenta os resultados dessa simulação.



(a) Entrada do equalizador

(b) Saída do compensador

(c) Resultado após execução do CMA

(d) Resultado após execução do LMS

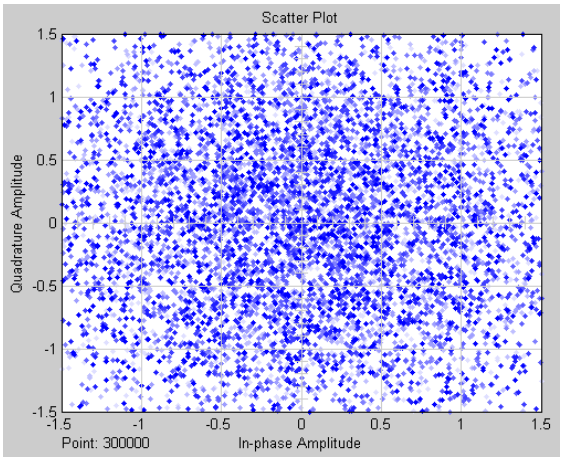
Figura 5.10: Desempenho dos algoritmos para a modulação 16QAM com compensador 200 km.

Verifica-se que há melhorias significativas com este compensador, como descreve a constelação da Figura 5.10(b), onde aos símbolos dispersos da Figura 5.10(a) é retirada quase a totalidade da dispersão ficando estes dispostos em 16 nuvens mais concentradas em torno dos 16 pontos da constelação da Figura 5.10(b).

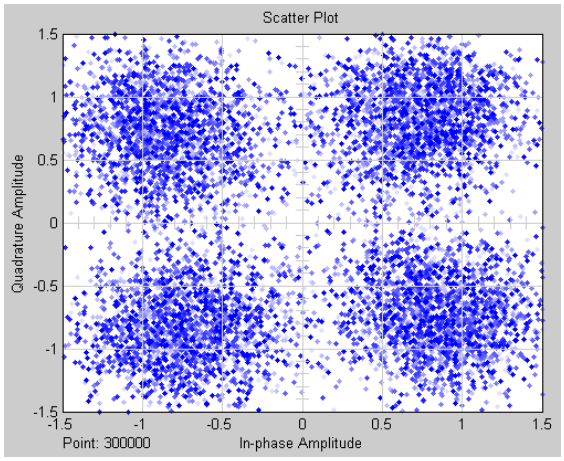
Após o compensador, os algoritmos melhoram o seu desempenho e apresentam as constelações esperadas, onde os símbolos são distribuídos pelos 16 pontos da constelação 16QAM, como se observa nas Figuras 5.10(c) e 5.10(d). O desempenho do CMA melhorou ao nível da convergência necessitando de um menor número de símbolos para convergir. O LMS apresenta um aumento significativo no desempenho em relação ao funcionamento apresentado sem o compensador (Figura 5.8(b)), como se esperava.

Compensador de 500 km para modulação 4QAM

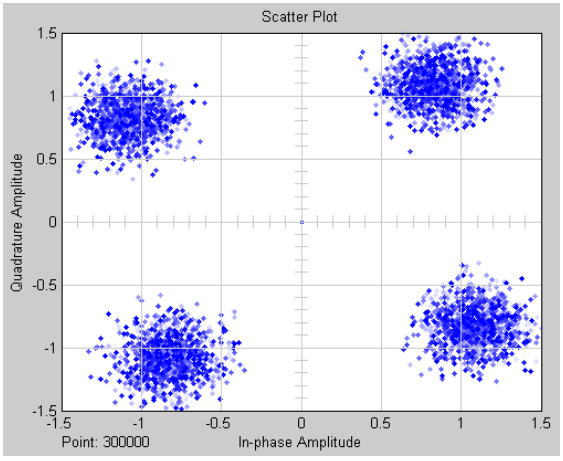
Desenvolveu-se, ainda, um compensador para 500 km de dispersão cromática com 31 coeficientes fixos, cujos resultados se apresentam na Figura 5.11.



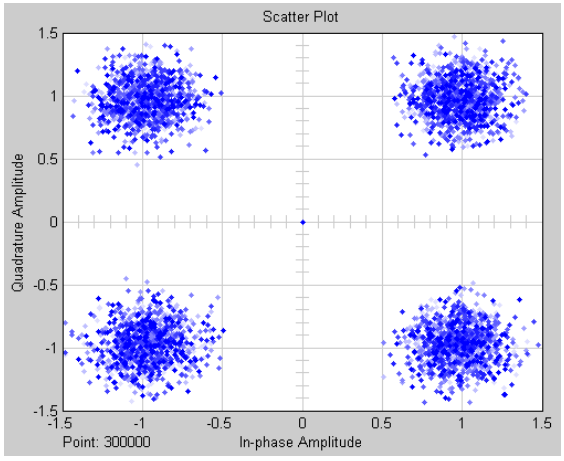
(a) Entrada do equalizador



(b) Entrada do equalizador



(c) Resultado após execução do CMA



(d) Resultado após execução do LMS

Figura 5.11: Desempenho dos algoritmos para a modulação 4QAM com compensador 500 km.

Como se observa na Figura 5.11(b), o compensador compensa a dispersão, mas não correctamente como para o caso anterior (Figura 5.9(b)). Conclui-se, desde já, que os 31 coeficientes não

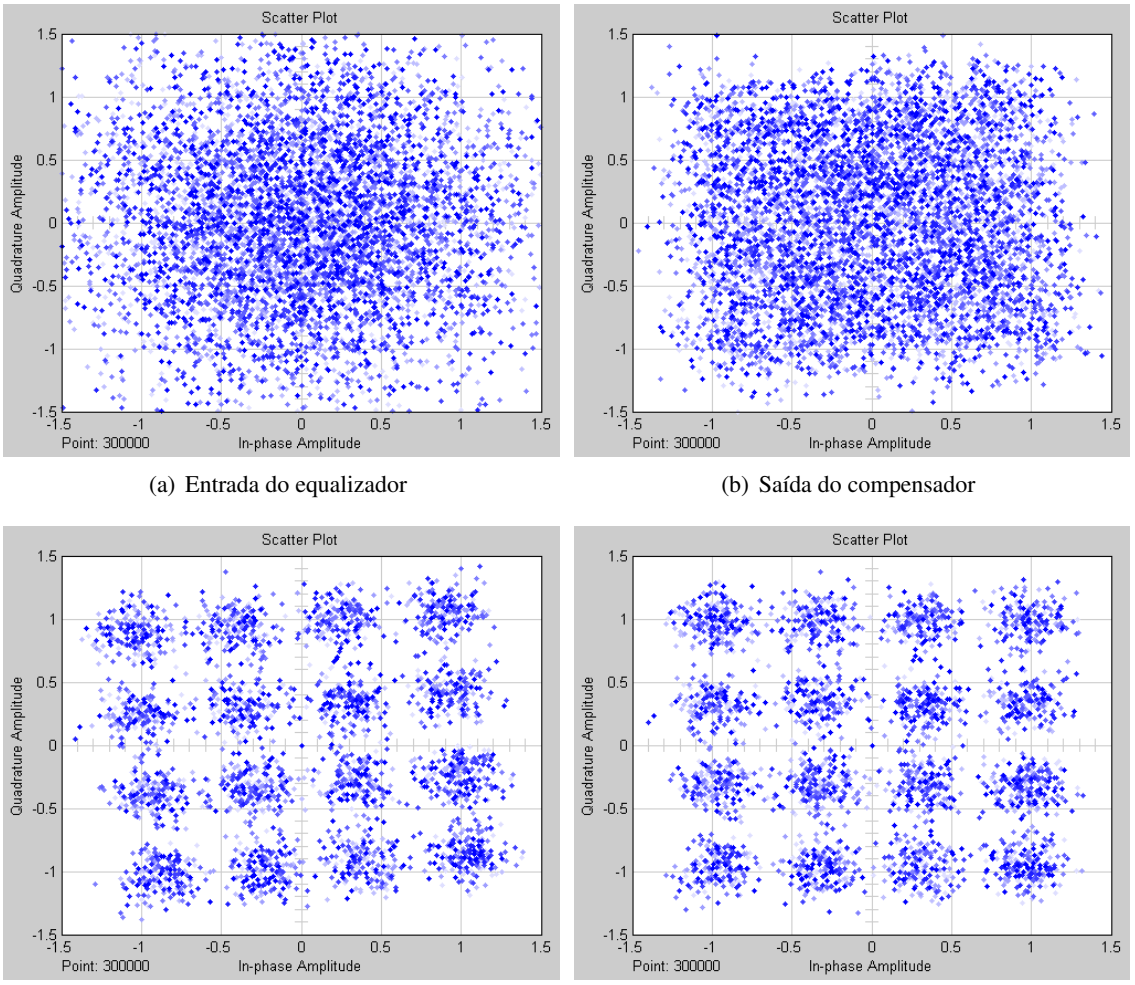
são totalmente suficientes. No entanto, para 4QAM os dois algoritmos apresentam bom desempenho.

O facto do compensador não ter o número de coeficientes suficiente leva a que a constelação resultante do algoritmo CMA esteja ligeiramente rodada para a esquerda (Figura 5.11(c)). Já a constelação referente ao algoritmo LMS é apresentada sem sofrer qualquer desvio na fase, fruto da característica de funcionamento do aproprio algoritmo.

Segue-se o desempenho deste compensador para a modulação 16QAM.

Compensador de 500 km para modulação 16QAM

Para a recepção do sistema 16QAM, o presente compensador não compensa eficazmente a dispersão cromática, como é observável pela Figura 5.12(b), o que permite confirmar que o valor 31 para os coeficientes não é totalmente suficiente para compensar 500 km de distância.



(a) Entrada do equalizador

(b) Saída do compensador

(c) Resultado após execução do CMA

(d) Resultado após execução do LMS

Figura 5.12: Desempenho dos algoritmos para a modulação 16QAM com compensador 500 km.

Apesar disso, realça-se o bom desempenho dos algoritmos, apresentando o CMA um pequeno desvio de fase na constelação da Figura 5.12(c). A Figura 5.12(d) apresenta o óptimo desempenho do LMS, conseguindo este, compensar o pequeno desvio de fase da constelação que o CMA apresenta.

Portanto, visto que o CMA apresenta esse pequeno desvio de fase, verifica-se que o compensador terá de ter mais coeficientes para, deste modo, realizar a convolução com maior percentagem de energia da função de transferência da fibra diminuindo o erro produzido pela truncagem realizada.

5.4 Limites do sistema

Os limites do sistema em termos de taxas de transmissão variam consoante o tipo de plataforma FPGA usada. A Tabela 5.2 mostra as simulações realizadas para várias plataformas FPGAs, realizadas em *System Generator*.

Plataforma	**Recursos**		**CMA**	
FPGA	*Slices*	*Emb. Mults*	Versão Sequencial	Versão Paralela
Spartan 3A XC3S700A	5.888	20	T: 20 ns F: 50 MHz	Excede a capacidade
Virtex 4 XC4VLX60	26.624	64	T: 17,266 ns F: 57,917 MHz	T: 10,294 ns F: 97,144 MHz
Virtex 4 XC4VSX55	24.576	512	T: 14,711 ns F: 67,976 MHz	T: 13,344 ns F: 74,940 MHz
Virtex 5 XC5VLX330T	51.840	192	T: 8,955 ns F: 111,669 MHz	T: 9,212 ns F: 108,530 MHz
Virtex 5 XC5VSX95T	14.720	640	T: 9,167 ns F: 109,087 MHz	T: 9,892 ns F: 101,092 MHz

Tabela 5.2: Características e funcionamento de cada plataforma FPGA. "T" representa o tempo mínimo e "F" a frequência máxima.

Esta ferramenta de desenvolvimento permite, com facilidade, simular os algoritmos implementados para vários tipos de plataformas sem que sejam necessárias alterações de maior.

São apresentadas na Tabela 5.2 algumas características e tempos mínimos que um ciclo de relógio consome, definindo o limite de funcionamento do algoritmo CMA. Apresenta-se aqui o CMA, porque foi com este algoritmo que se realizaram as simulações; para o LMS os valores são semelhantes.

Na Tabela 5.3 são apresentadas as taxas de transmissão conseguidas para várias FPGAs nas duas configurações de implementação.

Plataforma FPGA	**Taxas de transmissão**	
	Versão Sequencial	Versão Paralela
Spartan 3A XC3S700A	1,92 MSímbolos/s	—
Virtex 4 XC4VLX60	2,23 MSímbolos/s	9,71 MSímbolos/s
Virtex 4 XC4VSX55	2,61 MSímbolos/s	7,49 MSímbolos/s
Virtex 5 XC5VLX330T	4,29 MSímbolos/s	10,86 MSímbolos/s
Virtex 5 XC5VSX95T	4,20 MSímbolos/s	10,11 MSímbolos/s

Tabela 5.3: Taxas de transmissão para várias plataformas de FPGAs.

Pala análise à Tabela 5.3 pode-se concluir que os algoritmos implementados em paralelo permitem taxas de transmissão melhores que as da configuração sequencial, isto é devido à latência da configuração paralela ser de 10 ciclos de relógio.

Portanto, o limite deste sistema em termos de taxas de transmissão é de 10,86 MSímbolos/s, sendo esta taxa obtida com a plataforma FPGA Virtex 5 XC5VLX330T. Supostamente, a Virtex 5 XC5VSX95T deveria atingir mais velocidade, pois esta é direccionada para DSP, no entanto esta obteve taxas ligeiramente inferiores.

Este pequeno decréscimo pode dever-se ao facto da Virtex 5 XC5VSX95T ter uma ocupação inferior (14.720 *slices*) em comparação com a Virtex 5 XC5VLX330T (51.840 *slices*). Perante este cenário, o *System Generator* ao realizar o mapeamento na FPGA Virtex 5 XC5VSX95T, tem menor margem de manobra para realizar uma implementação optimizada e eficiente.

Com o sistema em paralelismo pode-se diminuir o tempo entre cada símbolo à saída do equalizador e, no caso de se manterem as mesmas velocidades de processamento das FPGAs, obtêm-se excelentes melhorias: por exemplo, para a versão em paralelo a taxa de transmissão aumenta até 5 vezes mais passando de 10 MSímbolos/s para 50 MSímbolos/s, considerando que a velocidade máxima da FPGA se mantém constante em volta dos 100 MHz.

5.5 Plataforma FPGA adequada para a tarefa proposta

Uma plataforma adequada ao sistema será uma FPGA com grande capacidade de recursos *(slices)* e em particular, possuir muitos multiplicadores dedicados de alta velocidade permitindo ao sistema implementado um óptimo desempenho, uma vez que este realiza muitas multiplicações.

Uma possível plataforma FPGA adequada ao sistema será a Virtex 5 XC5VSX240T, com 37.440 *slices* e 1.056 multiplicadores dedicados. Esta FPGA é direccionada para DSP, pelo que se poderá obter bons resultados.

5.6 Resumo

Neste capítulo foram apresentados os resultados de implementação e verificou-se que a configuração paralela gasta muitos mais recursos que a configuração sequencial (em cerca de 6 vezes mais).

Os resultados das simulações mostram o bom funcionamento dos algoritmos para 4QAM e 16QAM, sendo este bom funcionamento validado com a comparação dos resultados obtidos com os do MATLAB. Realizou-se também uma validação em hardware dos algoritmos através da co-simulação. Comprovou-se, ainda, que os algoritmos apresentam um desempenho extra, isto é, na presença de dispersão cromática são capazes de se adaptarem e compensarem essa dispersão até 200 km.

Posteriormente, foram apresentados os resultados com um compensador de dispersão desenvolvido para 200 km e outro para 500 km. Os resultados foram bastante razoáveis e satisfatórios.

Na secção 5.4, fez-se um estudo da implementação de várias plataformas FPGA a fim de avaliar os limites do sistema. Foram obtidas as taxas de transmissão com valores na ordem dos 11 MSímbolos/s para a configuração paralela e dos 4 MSímbolos/s para a configuração sequencial.

Por último, foi apresentada uma possível plataforma para a implementação em hardware dos algoritmos desenvolvidos.

Seguem-se, no capítulo 6, as conclusões finais e trabalho futuro.

Capítulo 6

Conclusões finais e trabalho futuro

6.1 Satisfação dos objectivos

Este trabalho envolveu o estudo e a implementação de algoritmos adaptativos para equalização de sinal em sistemas ópticos coerentes.

Para a implementação desses algoritmos em plataformas FPGAs foi necessária a aprendizagem das ferramentas de desenvolvimento, de síntese e de implementação em hardware.

O objectivo principal desta tese de dissertação é a implementação dos algoritmos adaptativos em plataformas FPGA. Este objectivo foi cumprido com o desenvolvimento dos algoritmos CMA e LMS em duas configurações distintas: uma sequencial e a outra em paralelo.

No decorrer da implementação em *System Generator* houve a necessidade de recorrer a algumas técnicas para contornar certos aspectos de implementação, quando a aprendizagem da ferramenta de desenvolvimento estava no início.

Neste trabalho houve primeiro um estudo de *scripts* MATLAB onde os algoritmos já tinham sido desenvolvidos. O facto de existir essa implementação dos algoritmos em MATLAB permite comparar os resultados obtidos no *System Generator* (FPGA) com os do MATLAB, e assim, realizar uma correcta validação da implementação pelos resultados obtidos.

Esta metodologia é extremamente recente e é possível, graças ao poder de abstracção da ferramenta de desenvolvimento *System Generator*, obter resultados em simulação e/ou transportá-los para o ambiente MATLAB a fim de serem tratados para boa interpretação dos mesmos.

Como resultados importantes destacam-se o funcionamento dos algoritmos nas duas configurações diferentes, apresentando taxas de transmissão diferentes para cada uma delas. Estes limites do sistema variam consoante a plataforma FPGA usada, por exemplo, para a Virtex 5 consegue-se taxas de 4 MSímbolos/s e 11 MSímbolos/s, para a configuração sequencial e paralela, respectivamente.

Estes limites poderão ser aumentados caso se consiga uma implementação de paralelismo dos algoritmos aqui desenvolvidos conseguindo-se, desta forma, diminuir a latência dos símbolos à saída do sistema, aumentando as taxas de transmissão.

Foram também desenvolvidos compensadores de dispersão cromática da fibra, conseguindo-se assim melhor desempenho dos algoritmos, perante esta penalidade do sistema.

No capítulo 5 foram apresentados e discutidos os resultados obtidos para sistema com modulação 4QAM e 16QAM.

Este trabalho abre novos horizontes neste tipo de tecnologia que é recente e permite, daqui em diante, ter uma base de partida para futuras implementações e adaptações.

6.2 Trabalho futuro

Para trabalho futuro seria interessante optimizar a implementação dos algoritmos no módulo do filtro, uma vez que este é o ponto crítico dos equalizadores implementados. Isto seria possível realizando uma implementação com blocos DSP48, já que são capazes de executar multiplicações e somas com relativa facilidade e permitem aumentar a velocidade de processamento para, desta forma, se diminuir a latência entre símbolos.

Uma outra funcionalidade interessante seria a implementação de algoritmos *FeedForward*. Este algoritmo poderá desempenhar um papel fundamental na sincronização da portadora óptica, a quando da presença de ruído de fase que o transmissor provoca. Isto seria importante para melhorar o funcionamento do LMS, uma vez que na presença de ruído de fase este não obtém convergência.

Anexo A

Poster

O presente poster foi apresentado no Simpósio Nacional: *Symposium on Enabling Optical Networks*, nas instalações da Nokia Siemens Networks, Amadora, Portugal, a 26 Junho de 2009.

Universidade do Porto
FEUP Faculdade de Engenharia

INESCPORTO
INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES DO PORTO
LABORATORIO ASSOCIADO

FPGA Implementation of Signal Processing Algorithms in Coherent Optical Systems

Nuno M. Pinto (INESC Porto/FEUP)
Luís M. Passos (INESC Porto/FEUP)
João C. Ferreira (INESC Porto/FEUP)
Henrique M. Salgado (INESC Porto/FEUP)

Abstract

This work describes the FPGA hardware implementation of well known algorithms for signal estimation in coherent optical systems. The results address both serial and parallel implementations, in terms of complexity and performance. Proof of principle results were obtained, which could be easily scaled to currently used optical systems data rates.

Coherent Systems advances

These systems have gained renewed interest due to the availability of high speed digital signal processing, which allows for complex operations to be carried out in the digital domain, enabling high potential for a reconfigurable software defined optical receiver, while enabling "quasy-exact" compensation of linear transmission impairments (CD and PMD) by a linear filter, and higher spectral efficiency.

Adaptive equalization

- Constant Modulus Algorithm – CMA – is the most used algorithm for adaptive equalizers, essentially because of its robustness and ability to converge prior to phase recovery.
- This algorithm provides good performance, but ignores the signal phase causing the constellation to twist.
- Least Mean Squares – LMS – is similar to the CMA while being able to track small signal phase phase changes.

FPGA Implementation

The implementation of CMA and LMS algorithms is achieved using System Generator. Figure 1 describes the serial version of the CMA equalizer. Each symbol is presented after the data and the coefficients are loaded in the filter block. Then the algorithm error is calculated to allow the coefficients to be updated. This process is repeated for each set of two samples.

CMA equalizer implemented in System Generator

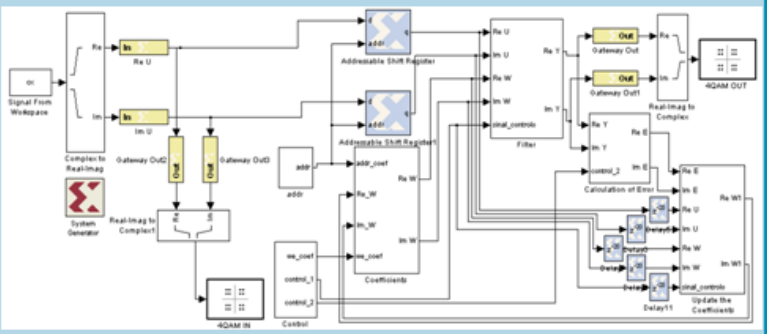
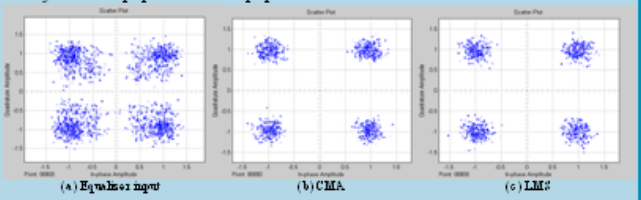


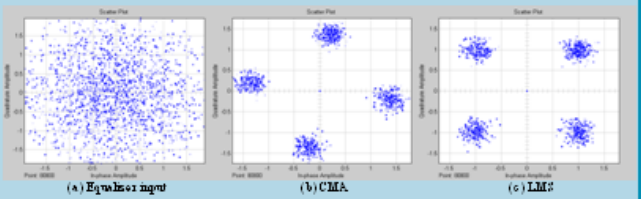
Fig 1 - The FPGA is delimited by blocks In - Input path - and Out - Output path.

In this work 18 bits were used and the number of coefficients is 13. Each algorithm originated two different versions, one serial and one parallel. While the parallel version uses 6 times more resources than the serial, it provides an over performance of 5 times. We present the results of a 4-QAM constellation, with and without CD for both algorithms (CMA and LMS). Figure 2 plots the CMA and LMS performance without dispersion. Figure 3 shows that the presented algorithm successfully compensates CD of 200 km of standard single mode fiber (SSMF), since the 13 taps are capable of synthesizing the inverse transfer function of the fiber CD.



(a) Equalizer input (b) CMA (c) LMS

Fig. 2. 4-QAM constellation without dispersion.



(a) Equalizer input (b) CMA (c) LMS

Fig. 3. 4-QAM constellation with 200 km fiber.

Conclusion

We conclude these algorithms might be implemented with good performance given that a sufficient number of bits is used. We have successfully initialized the LMS algorithm with the coefficients resulting from CMA convergence. For high fiber distances an equalization module with fixed coefficients would be required.

Acknowledgements

This work was supported in part by "Fundação para a Ciência e a Tecnologia" (FCT) under the programme "Programa Operacional Ciência Tecnologia e Inovação" – POCTI/FEDER with grant REEQ/1272/EEI/2005-Fibre Optic Supported Broadband Communication Networks.

Figura A.1: Poster Seon 2009.

Referências

[1] H. Ghafouri-Shiraz. *The Principles of Semiconductor Laser Diodes and Amplifiers - The Principles of Semiconductor Laser Diodes and Amplifiers*. Imperial College Press, 2004.

[2] Gerd Keiser. *Optical Fiber Communications*. McGraw-Hill Series in Electrical Engineering, 2ª edição, 1991.

[3] Keang-Po Ricky Ho. Coherent Optical Communications. Class notes, February 2002. Consultado em http://cc.ee.ntu.edu.tw/~kpho/u1310/intro.pdf.

[4] T. Okoshi. Recent Advances in Coherent Optical Fiber Communication Systems. *Journal of Lightwave Technology*, LT-5(1):44–52, January 1987.

[5] T. Pfau, S. Hoffmann, R. Peveling, S. Bhandare, S. K. Ibrahim, O. Adamczyk, M. Porrmann, R. Noé, e Y. Achiam. First Real-Time Data Recovery for Synchronous QPSK Transmission With Standard DFB Lasers. *IEEE Photonics Technology Letters*, 18(18):1907–1909, September 2006.

[6] L. M. Pessoa, H. M. Salgado, e I. Darwazeh. Algorithms for DSP implementation in coherent optical systems. Em *Cranfield Multi-Strand Conference*, Cranfield, United Kingdom, 5-6 May 2008.

[7] Xilinx. *System Generator for DSP - User Guide*. Release 10.1, March 2008.

[8] Xilinx. *System Generator for DSP - Getting Started Guide*. Release 10.1, March 2008.

[9] E. Ip, A. Lau, D. Barros, e J. M. Kahn. Coherent Detection in Optical Fiber Systems. *Optics Express*, 16(2):753–791, January 2008.

[10] L. M. Pessoa, H. M. Salgado, e I. Darwazeh. Joint Mitigation of Optical Impairments and Phase Estimation in Coherent Optical Systems. Em *IEEE LEOS Summer Topical Meetings 2008*, páginas 169–170, Mexico, July 2008. Paper TuE4.3.

[11] A. Leven, N. Kaneda, e Y.-Kai Chen. A Real-Time CMA-Based 10Gb/s Polarization Demultiplexing Coherent Receiver Implemented in an FPGA. Em *OFC/NFOEC 2008*, San Diego, California, 24 February 2008. Paper OTuO2.

[12] A. Leven, N. Kaneda, A. Klein, U.-V. Koc, e Y.-K. Chen. Real-Time Implementation of 4.4 Gbit/s QPSK Intradyne Receiver Using Field Programmable Gate Array. *Electronics Letters*, 42(24):1421–1422, 23 November 2006.

[13] Sílvio A. Abrantes. *Processamento Adaptativo de Sinais*. Fundação Calouste Gulbenkian, Lisboa, 2000.

[14] Xi-Lin Li e Xian-Da Zhang. A Family of Generalized Constant Modulus Algorithms for Blind Equalization. *IEEE Transactions on Communications*, 54(11):1913–1917, NOVEMBER 2006.

[15] S. Savory. Digital filters for coherent optical receivers. *Optics Express*, 16(2):804–817, 9 January 2008.

[16] Xilinx. *System Generator for DSP - Reference Guide*. Release 10.1, March 2008.

[17] G. P. Agrawal. *Fiber-Optic Communications Systems*. John Wiley & Sons, Inc, 3ª edição, 2002.

[18] Vítor H. Nascimento e Magno T. M. Silva. Stochastic Stability Analysis for the Constant-Modulus Algorithm (CMA). *IEEE Transactions on Signal Processing*, 56(10):4984–4989, 2008.

[19] M. Taylor. Coherent Detection Method Using DSP for Demodulation of Signal and Subsequent Equalization of Propagation Impairments. *IEEE Photonics Technology Letters*, 16(2):674–676, February 2004.

[20] Xilinx. Virtex-5 Family Overview, February 2009.

[21] Xilinx. Spartan-3A FPGA Family: Introduction and Ordering Information, March 2009.

[22] Xilinx. Virtex-4 Family Overview, September 2007.