Plataforma Matlab/FPGA com Otimização de Desempenho para Simulação de Sistemas MIMO

Matias A. Bortoluzzi e Renato Machado

***Resumo*— Neste artigo, apresenta-se uma plataforma Matlab/FPGA com otimização de desempenho para simulação de sistemas MIMO (*Multiple-Input Multiple-Output*). Este simulador combina a flexibilidade da linguagem de alto nível Matlab com o elevado poder de processamento da FPGA (*Field-Programmable Gate Array*). As variáveis envolvidas na simulação são geradas em Matlab e enviadas para memórias RAMs da FPGA onde realiza-se, rapidamente, a análise de BER (*Bit Error Rate*) $\times$ SNR. Essas grandezas manipuladas pelo Matlab em ponto flutuante são quantizadas para a representação em ponto fixo, adequadas para o processamento em FPGA. Uma análise da perda de desempenho na quantização do ponto flutuante é realizada com intuito de obter o tamanho otimizado das memórias RAMs em função da requisição de BER. Por fim, utiliza-se a plataforma proposta para um estudo de caso em um sistema SISO (*Single-Input Single-Output*) que serve de referência para as técnicas MIMO.**

***Palavras-Chave*— FPGA; Matlab; MIMO; quantização; simulador.**

***Abstract*— In this paper, we present a Matlab/FPGA platform with performance optimization for multiple-input multiple-output (MIMO) systems simulation. This simulator combines the flexibility of high-level Matlab language with the Field-Programmable Gate Array (FPGA) power processing. The variables involved in the simulation are generated in Matlab and sent to the FPGA RAM memories where they are used to quick analyze the bit error rate (BER) $\times$ SNR performance. These floating point quantities handled by Matlab are quantized to fixed point representation, suitable for processing on FPGA. An analysis of the performance loss in floating point quantization is performed in order to obtain the optimal size of the RAM memories depending on the BER requirements. Finally, we use the proposed plataform to a case study on a single-input single-output (SISO) which is a reference of MIMO techniques.**

***Keywords*— FPGA; Matlab; MIMO; quantization; simulator.**

I. Introdução

A completa caracterização de um sistema de comunicação requer a medição de propriedades estatísticas, sendo uma das mais importantes a taxa de erro de bit de transmissão [1]. Dependendo da complexidade do sistema, essas métricas não podem ser derivadas analiticamente e precisam ser extraídas de simulações do tipo Monte Carlo, que requerem elevada taxa de processamento para uma rápida análise de desempenho. Nesse contexto, a prototipagem em FPGA com verificação em *hardware* surge como uma boa solução para redução do tempo de análise de desempenho em comparação com simulações em linguagens de alto nível como C, C++, ou Matlab.

Matias Américo Bortoluzzi e Renato Machado estão com o Grupo de Pesquisa em Comunicações e Processamento de Sinais, Departamento de Eletrônica e Computação, Universidade Federal de Santa Maria, Santa Maria, RS, 97105-900, Brasil. Email: matiasbortoluzzi@gmail.com, renatomachado@ufsm.br

Em função das vantagens supracitadas a implementação em FPGA tem despertado a atenção da indústria e da academia nos últimos anos [2]. Em [3] é realizada a implementação em FPGA de um receptor MIMO utilizando o código de Alamouti para as modulações BPSK, QPSK e 16QAM. Em [4] é apresentado e implementado um codificador baseado no esquema de Alamouti. Já em [5], um codificador convolucional, um decodificador utilizando o algoritmo de Viterbi e um pseudo gerador de ruído são programados em FPGA.

Entretanto, a implementação em *hardware* possui limitações que não são verificadas em linguagens de alto nível. No caso de algoritmos embarcados em FPGA um dos grandes desafios é o processamento de grandezas reais/complexas e a necessidade de conversão da representação em ponto flutuante para ponto fixo [6]. Usualmente as aplicações que envolvem processamento digital de sinais são implementadas em sistemas embarcados com a aritmética de ponto fixo para minimizar custos, área e consumo [7]. No entanto, a conversão para ponto fixo causa perda de desempenho. Assim, a máxima perda de desempenho precisa ser definida para que seja garantido um mínimo de precisão [8].

Neste artigo, uma plataforma Matlab/FPGA com otimização de desempenho para simulação de sistemas de comunicação MIMO é proposta. Uma varredura da razão sinal-ruído (*signal-to-noise ratio* (SNR)) é realizada para se obter as respectivas taxas de erro de bit (*bit error rate* (BER)) e o traço das curvas de desempenho. As informações necessárias à simulação são geradas em Matlab e importadas pela FPGA, onde são armazenadas em memórias RAMs. Através do Matlab o simulador determina o comprimento ótimo da palavra binária, que representa menor ocupação de memória RAM e maior velocidade de execução na FPGA, além da exatidão mínima necessária para uma faixa de SNR, ou de BER, considerando o erro de quantização do processo de conversão de ponto flutuante para ponto fixo.

O restante deste artigo está organizado da seguinte maneira: a Seção II propõe o modelo do sistema; a Seção III apresenta a plataforma de simulação proposta; na Seção IV, considera-se um estudo de caso; a Seção V apresenta alguns resultados deste estudo; e, finalmente, a Seção VI encerra o artigo com algumas conclusões e comentários finais.

II. Modelo do Sistema

Um fluxo de projeto completo de um protótipo embarcado, incluindo as etapas em *software* e *hardware*, é apresentado em [1]. O fluxo de projeto apresentado em [6] é adaptado para este trabalho conforme ilustra a Fig. 1.

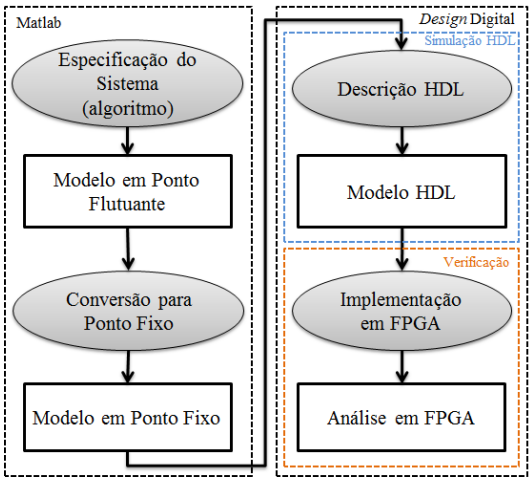

Fig. 1. Fluxo de projeto para o simulador embarcado.

A. Modelo em Ponto Flutuante

O fluxo de projeto mostrado na Fig. 1 inicia com a **especificação da arquitetura do sistema** e a avaliação do algoritmo. Esse algoritmo é tipicamente descrito em linguagem de alto nível, como Matlab, C, ou C++ e utiliza operações em ponto flutuante, constituindo um **modelo em ponto flutuante**.

B. Conversão para Ponto Fixo

O modelo em ponto flutuante descrito precisa ser convertido - através do procedimento de quantização - para a representação em ponto fixo, constituindo um **modelo em ponto fixo**. Em [8] é apresentado em detalhes os passos para o refinamento do processo de quantização. O maior esforço dessa etapa é a escolha dos comprimentos de palavra adequados para as operações aritméticas nos caminhos de dados do processamento digital [6].

Inerente ao procedimento de quantização há a perda de desempenho em relação ao modelo em ponto flutuante. Essa perda de desempenho é causada pela redução da exatidão na representação dos dados. Para minimizar os efeitos da quantização deve ser determinada a máxima perda de desempenho permitida. No entanto, deve-se atentar para o comprimento desnecessário da palavra binária sob pena de redução do desempenho em *hardware*.

Dessa maneira, uma estratégia de refinamento é necessária para conversão de ponto flutuante para ponto fixo. Tradicionalmente, esse refinamento é manual e consume bastante tempo. Existem dois tipos de abordagens para a busca desse refinamento no processo de quantização [9]:

- Abordagem baseada em simulação: compara o desempenho do sistema descrito (algoritmo) com um modelo de referência. Os tamanhos das palavras são escolhidos heuristicamente observando algum critério de erro. O procedimento é repetido até que o resultado convirja;
- Abordagem analítica: os tamanhos das palavras são derivados analiticamente. Esse método é rápido, porém conduz à sobre-estimação do tamanho das palavras.

Este trabalho considera a abordagem baseada em simulação para determinar qual o tamanho ótimo de bits (para uma dada faixa de BER) da palavra binária que irá armazenar cada um dos N diferentes canais emulados. Nesse contexto, objetiva-se obter o menor número possível de bits capaz de representar cada um dos sinais nos diferentes caminhos de processamento do sistema [10]. Para isso, curvas são levantadas considerando a perda de desempenho devido a redução de exatidão. Isso permite minimizar a ocupação de memória RAM da FPGA, afinal de contas, esse recurso é finito.

C. Descrição HDL

O algoritmo descrito em linguagem de alto nível é, então, convertido para linguagem de descrição de *hardware* (Verilog, ou VHDL), geralmente em nível comportamental, ou RTL, constituindo um **modelo HDL**. Os tamanhos das palavras especificados no item anterior são utilizados na descrição comportamental do sistema. O controle, as operações matemáticas e o caminho dos dados devem ser especificados pelo projetista para se atingir as restrições de consumo, área, *throughput* e latência. Uma simulação lógica é realizada para avaliar o comportamento do circuito descrito a partir de estímulos de entrada gerados no *testbench*.

D. Implementação em FPGA

Por fim, o sistema pode ser sintetizado e **implementado na FPGA** percorrendo as etapas de mapeamento, localização e roteamento. Em seguida, procede-se a **análise em FPGA** com o intuito de validar os resultados.

III. Plataforma Proposta

O desenvolvimento de uma plataforma para simulação de sistemas MIMO embarcada em FPGA deve possuir algumas características especiais que justifiquem a adoção de um dispositivo reconfigurável. Caso contrário, a simulação em linguagem de alto nível (e.g., Matlab) tende a ser mais atrativa. Desse modo, as características abaixo relacionadas são atendidas com a proposta desse trabalho:

- Avaliação extremamente rápida graças ao alto potencial de processamento da FPGA;
- Plataforma em *hardware* configurável para analisar diferentes técnicas de comunicação;
- Flexibilidade garantida pela utilização do Matlab como fonte de dados para a FPGA, permitindo que esta emule qualquer modelo de desvanecimento previamente estabelecido, como AWGN, Rayleigh, Rice e Nakagami;
- Versatilidade para alteração dos blocos implementados em *hardware*. O sistema de comunicação foi pensado em blocos, conforme ilustra a Fig. 2, para permitir fácil expansão e/ou modificação da técnica de comunicação implementada. Para exemplificar é possível incorporar facilmente o bloco codificador/decodificador de fonte ou o bloco estimador de canal ao sistema de comunicação;
- Acompanhamento do andamento da simulação por meio de interfaces incorporadas à FPGA;
- Otimização dos recursos da FPGA através da emulação de comportamento do *hardware* em *software* Matlab;
- Proximidade de um dispositivo ASIC.

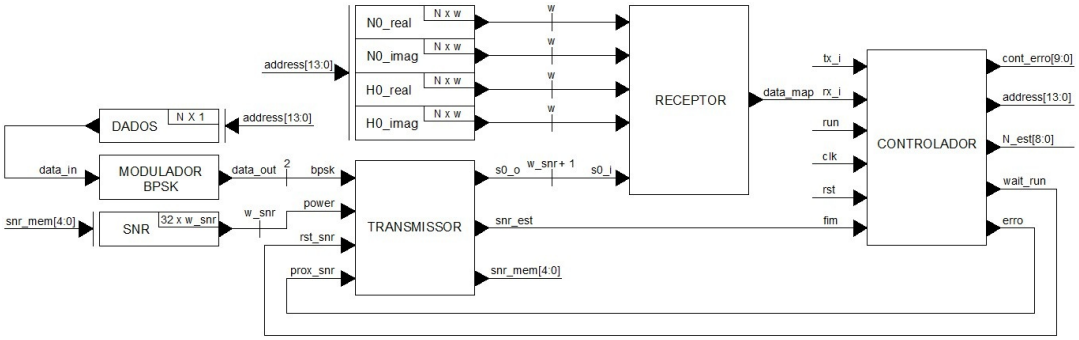

Fig. 2. Topologia SISO implementada em *hardware*.

Para atender essas características uma plataforma Matlab/FPGA foi desenvolvida. A Fig. 1 sugere a distinção entre dois grandes blocos, um atendendo o desenvolvimento em linguagem de alto nível e outro voltado para o *design* digital.

IV. Estudo de Caso: Sistema SISO

O esquema SISO foi adotado inicialmente em função de sua simplicidade e por fornecer a base para implementação das técnicas MIMO, através da alteração de alguns blocos mostrados na Fig. 2.

A. Algoritmo Matlab

Inicialmente, o *software* Matlab é utilizado para implementar um simulador MIMO, incluindo transmissor, canal de comunicação e receptor. Esse sistema fornece a flexibilidade necessária para combinar transmissor e receptor com diferentes condições de canal. Baseadas na flexibilidade providenciada pelo Matlab, as subseções seguintes se organizam para culminar na completa emulação de uma FPGA genérica em *software* Matlab, determinando otimizações para a etapa de implementação em *hardware* FPGA.

B. Emulação de Canal

Em [2], o gerador de canal é implementado em *hardware*. Neste artigo, o canal é gerado em Matlab e, em seguida, é exportado e armazenado em memórias RAMs na FPGA. Esse procedimento permite duas importantes vantagens em relação a [2]: diferentes modelos de canais podem ser facilmente emulados; e a otimização pode ser realizada via *software*.

A principal limitação para o procedimento proposto é o número de canais necessários (ocupação em *hardware*) para a obtenção de pelo menos um erro de bit por ponto da curva de desempenho. A estimação do número mínimo de canais para que pelo menos um erro de bit seja atingido (mínima BER) é obtida através do Matlab. Isso evita a sobrestimação do número de canais necessários para uma dada requisição de BER, otimizando a ocupação em *hardware*. A estatística para a simulação é garantida pela requisição de uma quantidade mínima de erros para cada SNR.

C. Conversão de Ponto Flutuante para Ponto Fixo

Em Matlab, grandezas complexas são manipuladas no padrão em ponto flutuante IEEE-754 com 64 bits. Todavia, a representação em ponto flutuante, além de representar um aumento de complexidade e consumo, exige uma grande área do dispositivo FPGA. Alternativamente, utiliza-se a representação em ponto fixo em formato complemento de 2. A comparação de desempenho entre os dois formatos de representação numérica é comumente utilizada para validar a representação em ponto fixo. O grande desafio é minimizar o erro de quantização, potencializando a exatidão entre o número real e sua representação em ponto fixo.

A técnica de conversão utilizada para obter um número inteiro y consiste na multiplicação de um número real x por uma constante inteira $k = 2^f$ (em que f representa o número de posições para o deslocamento da vírgula na representação binária). A esse efeito multiplicativo realiza-se um arredondamento para o inteiro mais próximo, conforme mostra a Equação (1). Essa abordagem distingue-se, por exemplo, de [2] e [8], nas quais a representação em ponto fixo utiliza alguns bits da palavra binária para a parte inteira e o restante para a parte fracionária do número real.

$$y = \text{round}(x \times 2^f) \tag{1}$$

Como consequência do efeito do arredondamento ocorre o erro de quantização que será tanto menor quanto maior for k. Além disso, o número inteiro y é armazenado em uma variável de w bits com sinal, cujo máximo positivo e mínimo negativo em valores absolutos são $2^{w-1} - 1$ e 2^{w-1}, respectivamente.

Dada uma sequência $X = x_1, x_2, \ldots, x_n$ em que $x_{\max}$ é o valor máximo de X, a relação entre w e f é dada pelas seguintes equações

- Para números negativos:

$$2^{w-1} \geq x_{\max} \times 2^f \tag{2}$$

e, portanto,

$$f \leq w - 1 - \log_2(x_{max}) \tag{3}$$

- Para números positivos:

$$2^{w-1} - 1 \geq x_{\max} \times 2^f \tag{4}$$

e, então,

$$f \leq \log_2 \left(\frac{2^{w-1} - 1}{x_{\max}} \right) \tag{5}$$

Para evitar *overflow* nas memórias adota-se sempre a equação para números positivos que resulta em f menor.

A sequência X acima pode representar qualquer grandeza real necessária para a simulação de alguma técnica MIMO, por exemplo, canal AWGN e Rayleigh.

D. Emulação da FPGA em Software Matlab

A junção das três etapas anteriores completa a emulação das características da FPGA em Matlab. Essa emulação permite obter a otimização em *hardware* a partir de simulações em *software*.

E. Implementação em FPGA

Os blocos apresentados na Fig. 2 foram descritos utilizando as linguagens VHDL e Verilog e implementados no kit de desenvolvimento *Altera Stratix II NIOS*. As informações de canal AWGN (*N0_real*, *N0_imag*), canal Rayleigh (*H0_real*, *H0_imag*), *DADOS* e *SNR* foram geradas no Matlab e exportadas para a FPGA. Elas possuem o tamanho em bits otimizado que foi determinado em simulação Matlab.

Os blocos do sistema SISO são:

Modulador BPSK: a entrada desse bloco (*data_in*) é composta por bits de dados a serem transmitidos. Cada símbolo BPSK é representado por dois bits em complemento de 2. O endereçamento da memória *DADOS* é sequenciado pelo barramento *address*. A variável *address* conta a quantidade de bits recebidos até um total de N bits.

Transmissor: recebe o sinal *bpsk* proveniente do modulador e sobre esse sinal aloca a informação de SNR armazenada na memória *SNR*. A posição de memória somente é alterada quando uma quantidade mínima de erros é atingida no bloco controlador. Quando esta quantidade de erros é atingida o sinal *prox_snr* proveniente do bloco controlador assume nível lógico 1, incrementando a posição da memória *SNR*. Quando o valor mínimo de BER é atingido o sinal *snr_est* vai para nível lógico alto, indicando para o bloco controlador o *fim* da simulação.

Receptor: realiza a detecção dos bits recebidos, gerando o sinal *data_map*. No bloco receptor são acrescentados ao sinal transmitido os efeitos do desvanecimento de pequena escala (do tipo Rayleigh, por exemplo) e do ruído do canal. Por fim, os símbolos recebidos são mapeados, baseados na mínima distância Euclidiana quadrática, em uma tentativa de reconstrução do sinal transmitido.

Controlador: possui duas estruturas principais que são um comparador e dois contadores. O comparador verifica se os bits provenientes do receptor (*rx_i*) e do transmissor (*tx_i*, da memória *DADOS*) são idênticos. Nesse caso, diz-se que não houve erro de transmissão. Caso contrário, há o incremento da variável *cont_erro* até o valor máximo de *min_error*. O envio de símbolos para uma mesma SNR continua enquanto esse valor não for atingido. O outro contador (*address*) é responsável pela contagem da quantidade total de bits transmitidos, tendo como limitante superior N bits. Caso haja o estouro de N bits recebidos sem que *min_error* seja atingido o contador *N_est* registra a quantidade de estouros. Assim, através desse bloco, pode-se determinar a taxa de erro de bit através da equação:

$$\text{BER} = \frac{min\_error}{address + N \times N\_est} \tag{6}$$

Quando a quantidade mínima de erros for atingida o sinal *erro* vai para estado lógico 1 e no transmissor uma nova SNR é ativada para simulação. Caso não haja mais valores de SNR a serem simuladas, então o sinal *fim* vai para nível lógico alto. Por conseguinte, a simulação encerra-se, os registradores do bloco controlador são zerados e o sinal *wait_run* vai para 1, ocasionando, também, a inicialização dos registradores do bloco transmissor. Assim, o sistema fica pronto para reiniciar uma nova simulação a partir de uma borda de subida de *run*.

V. Resultados

O ponto de partida para minimizar o erro de quantização consiste em definir a BER mínima desejada. A simulação é executada até que o valor de BER mínima definido seja atingido, trazendo como resultado o tamanho w da palavra binária envolvida no processamento. A Fig. 3 apresenta uma simulação paramétrica de w, para os canais AWGN e Rayleigh do esquema SISO com modulação BPSK. Para otimização, deve-se escolher o valor mínimo de w em bits capaz de armazenar o máximo valor (em módulo) de um número real $x_n \in X$, evitando-se, com isso, a ocorrência de *overflow*.

Observe que $w = 9$ bits (com sinal) é considerado ótimo para uma BER mínima de aproximadamente 10^{-4}, pois rastreia de forma bastante fidedigna a curva que utiliza o modelo em ponto flutuante. Isso ocorre, pois à medida que w aumenta os caminhos de processamento são afetados positivamente, alimentando com mais precisão o estimador dos símbolos. A partir da alteração dos tamanhos w das memórias RAMs da FPGA chega-se a resultados idênticos na implementação em FPGA e, portanto, não são repetidamente apresentados. Salienta-se, no entanto, que a composição dessas curvas na FPGA é realizada uma a uma, já que é necessário a reconfiguração dos tamanhos de suas memórias. Observe que a seleção do tamanho de w ótimo é recomendável, pois em análise do tipo de Monte Carlo é necessário uma enorme quantidade de canais a fim de garantir a estatística adequada para a simulação. Por exemplo, para um espaço amostral com um milhão de canais, cada qual com tamanho $w = 9$ bits, será necessário uma capacidade de armazenamento de $(9 \times 10^6 \times 4)/8 = 4,5$ MB, em que '4' representa as partes real e imaginária dos canais Rayleigh e AWGN.

O número de canais (AWGN e Rayleigh) necessário para $w = 9$ bits e $BER = 10^{-4}$ é 13.954 (utiliza-se $2^{14} = 16.384$). Essa quantidade (mínima) garante que *pelo menos* um erro seja localizado para o maior valor de SNR. A adoção do número mínimo de canais associada à otimização de w permite a redução da ocupação de memória, redução do consumo e tempo de processamento da FPGA. A estatística da simulação é assegurada pela requisição algorítmica de uma quantidade mínima de erros para cada SNR. Se apenas um erro é localizado em N canais, então, o simulador fica em *loop* por K vezes até que K erros sejam localizados.

A partir da Fig. 3, é possível determinar um modelo analítico que relaciona w com a BER. Desse modo, basta que apenas alguns valores de w sejam simulados a fim de prever o tamanho de w para qualquer BER de interesse. Assim,

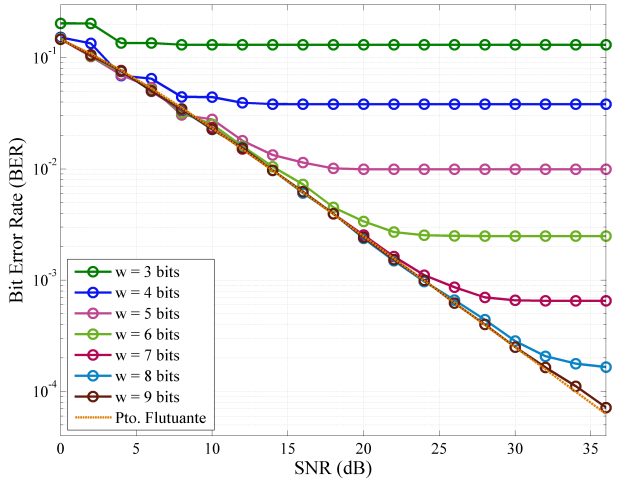

Fig. 3. Simulação SISO com variação paramétrica de w.

utiliza-se a equação,

$$w = 3,3562BER^{-115,3510\times10^{-3}} \text{ (bits)} \quad (7)$$

e arredonda-se o resultado para o inteiro superior.

As curvas w menores do que 9 bits apresentadas na Fig. 3 seguem uma tendência constante a partir de um dado valor de BER. Há, a partir desse ponto, um efeito chamado de *saturação* das informações do canal. Basicamente, essa é a manifestação do erro de quantização, ou da falta de exatidão dos dados.

O esquema clássico de Alamouti foi simulado utilizando modulação BPSK. A varredura paramétrica de w é mostrada na Fig. 4. Observe que há um comportamento bastante similar ao que ocorre para as curvas SISO. A palavra binária otimizada para uma BER de 10^{-4} é de 6 bits, e o número mínimo de canais é 78.120.

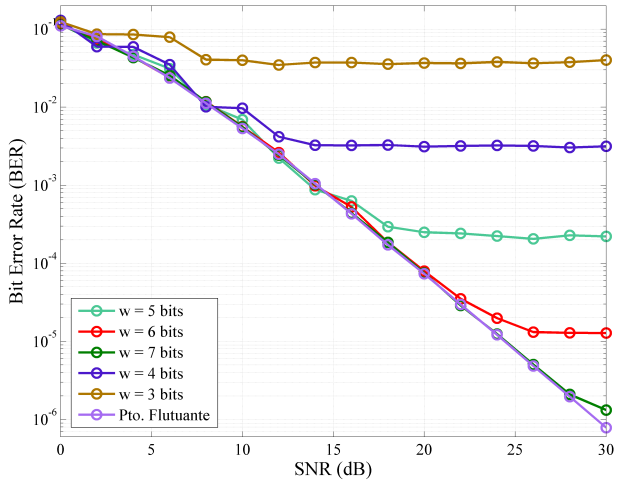

Fig. 4. Simulação do esquema de Alamouti com variação paramétrica de w.

A equação que relaciona w em função da BER para o caso de Alamouti foi obtida por linearização. O tamanho w é obtido pelo arredondamento para o inteiro superior da Equação

$$w = 3,4883BER^{-53,9799\times10^{-3}} \text{ (bits)} \quad (8)$$

VI. Conclusões e Considerações Finais

Este artigo buscou aproveitar as melhores qualidades que as ferramentas matemáticas de simulação podem oferecer. No caso, a flexibilidade para a otimização em *software* e o elevado potencial de processamento da FPGA para a rápida análise de desempenho em *hardware*. Um estudo de caso baseado no sistema SISO foi realizado, considerando-se a otimização em *software*, seguida da implementação em *hardware* para validar a plataforma proposta. Em consequência, o simulador apresenta uma redução do consumo de energia e da área ocupada, além de aumentar o desempenho da simulação. A implementação em *hardware* do sistema SISO realizada neste trabalho ilustra a redução temporal obtida na simulação por Monte Carlo, em comparação ao custo computacional da simulação SISO utilizando algum *software* matemático. Essa redução do custo computacional também se verifica em implementações MIMO. Para a sequência deste trabalho está prevista a avaliação de técnicas MIMO sob um canal *wireless* real. Para isso, um estimador de canal será implementado em *hardware*.

Agradecimentos

Este trabalho foi financiado em partes pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul.

Referências

[1] F. Borlenghi, D. Auras, E. M. Witte, T. Kempf, G. Ascheid, R. Leupers, and H. Meyr, "An FPGA-accelerated testbed for hardware component development in MIMO wireless communication systems," *Institute for Communication Technologies and Embedded Systems, RWTH Aachen University*, Germany, Mar. 2012, pp. 278–285.

[2] M. Véstias and K. Kum, "Design and test of a MIMO receiver based on the Alamouti scheme in FPGA", *IEEE International Conference on Consumer Electronics*, Lisbon, Portugal, Sept. 2012, pp. 107–111.

[3] P. Pinho and M. Véstias, "A high-rate MIMO receiver in an FPGA," *Antennas and Propagation Society International Symposium*, Instituto de Telecomunicações, Lisbon, Portugal, Jul. 2012, pp. 1–2.

[4] M.W. Numan, M.T. Islam, and N. Misran, "Implementation of Alamouti encoder using FPGA for MIMO Testbed", *International Conference on Advanced Computer Control*, Singapore, Jan. 2009, pp. 188–192.

[5] E. Lord, M. Devlin, N. Harold, and C. Sanderson, "Accelerating BER analysis using an FPGA based processing platform", *Proc. of Military Communications Conference*, California, Oct. 2002, pp. 218–223.

[6] P. Greisen, S. Haene, and A. Burg, "Simulation and emulation of MIMO wireless baseband transceivers," *EURASIP J. Wireless Commun. and Networking*, vol. 2010, pp. 1–12, Apr. 2010

[7] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 1–19, Jul. 2005.

[8] J. W. Weijers, V. Derudder, S. Janssens, F. Petré, and A. Boudoux, "From MIMO-OFDM algorithms to a real-time wireless prototype: A systematic matlab-to-hardware design flow," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 1–12, Feb. 2006.

[9] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," in *Proc. of Conference on Design, Automation and Test in Europe*, Munich, Germany, March 1999, pp. 271–276.

[10] W. Sung and K. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Transactions on Signal Processing*, vol.43, no. 12, pp. 3087–3090, Dec. 1995.

[11] S. M. Alamouti. "A simple transmit diversity technique for wireless communications," *IEEE Journal on Selected Areas in Communications*, vol 16, no. 8, pp. 1451–1458, Oct. 1998.