ARQUITETURA PARA EMULAÇÃO DE MICRORREDES EM FPGA

Gabriel Sousa*, Jean Patric da Costa*, Emerson Giovani Carati*, Rafael Cardoso*,
Carlos Marcelo de Oliveira Stein*, Zeno Luiz Iensen Nadal†

*Universidade Tecnológica Federal do Paraná
Programa de Pós-Graduação em Engenharia Elétrica
Pato Branco, Paraná, Brasil

†Copel Distribuição S.A.
Curitiba, Paraná, Brasil

Emails: sousa@alunos.utfpr.edu.br, jpcosta@utfpr.edu.br, emerson@utfpr.edu.br ,
rcardoso@utfpr.edu.br, cmstein@utfpr.edu.br, zeno.nadal@copel.com

Abstract— This paper presents a architecture made for Field Programmable Gate Array (FPGA) to emulate linear dynamic systems, especialy power distribuition systems, as a tool for microgrid study. Examples of alternative applications are shown for each component proposed in the paper. It also shows an example of a distribution network and compared the results with simulations in MATLAB.

Keywords— Microgrid Emulation, Linear Systems Emulation, FPGA, Fixed-Point Arithmetic.

Resumo— Neste trabalho é apresentada uma arquitetura para *Field Programmable Gate Arrays* (FPGA) para emulação de sistemas dinâmicos lineares, com ênfase na emulação de sistemas de distribuição, como ferramenta no estudo de microrredes. São mostrados exemplos de aplicações alternativas para cada componente proposto no trabalho. Também é mostrado um exemplo de uma rede de distribuição e comparada os resultados com simulações em MATLAB.

Palavras-chave— Emulação de Microrredes, Emulação de Sistemas Lineares, FPGA, Aritmética de ponto fixo.

1 Introdução

O conceito *microrrede* surge como uma alternativa ao paradigma de geração centralizada, onde o uso da geração distribuída, além de suprir de forma mais eficiente a crescente demanda de energia elétrica, permite o controle mais flexível do sistema de distribuição (Hatziargyriou, 2013; Katiraei et al., 2008; Lasseter, 2002). Contudo, o uso de unidades de geração distribuída, principalmente aquelas que possuem interfaceamento eletrônico de potência, dão à microrrede um comportamento sensivelmente distinto das redes de distribuição convencionais, dificultando o uso de técnicas clássicas para redes convencionais abordadas por autores conceituados como Kundur (1994).

Uma diferença entre redes de distribuição convencionais e microrredes é a presença da geração distribuída e a natureza das fontes primárias de energia. Em sistemas de distribuição convencionais, as fontes de energia primárias têm capacidade de operar de forma despachável, onde a potência gerada acompanha a demandada de forma que a geração e o consumo estejam sempre equilíbrio. No entanto, em microrredes, o uso de fontes renováveis torna a geração variável e dificilmente acompanha a demanda de energia (Hatziargyriou, 2013).

Quando a microrrede está conectada à rede, esse desbalanço é desejável para que seja possível exportar o excedente, o que é uma vantagem para o proprietário porque tal excedente pode ser monetizado, isso estabelece um desafio que não existe em sistemas convencionais, que é a maximização da exportação de potência ativa. Contudo, permitir que as unidades de geração injetem toda a potência ativa disponível pode levar a problemas de sobre tensão em momentos de baixa demanda e alta disponibilidade da fonte primária (Bevrani et al., 2017; Hatziargyriou, 2013).

Como forma de mitigar problemas de tensão e ao mesmo tempo permitir uma maior exportação de potência ativa, é possível utilizar a potência reativa como forma de controlar os níveis de tensão (Trivedi and Singh, 2018; Casavola et al., 2017; Thale et al., 2015; Bolognani and Zampieri, 2010). No entanto essa técnica possui limitações que não estão presentes em sistemas de geração convencionais com alta apacidade (Kundur, 1994), é a exigência legal que o fator de potência da unidade consumidora, a microrrede nesse caso, se mantenha em níveis aceitáveis (ANEEL, 2010).

Outro desafio que surge ao estudar microrredes, é a simulação do sistema quando se deseja exemplificar ou validar um método proposto. Em sistemas convencionais, as dinâmicas são mais lentas o que permite o uso de modelos médios, além de permitir a linearização de certos componentes sem comprometer a representatividade do modelo (Kundur, 1994).

No entanto, em microrredes, há uma coexistência entre dinâmicas rápidas, por exemplo o chaveamento dos inversores que ocorre na escala de

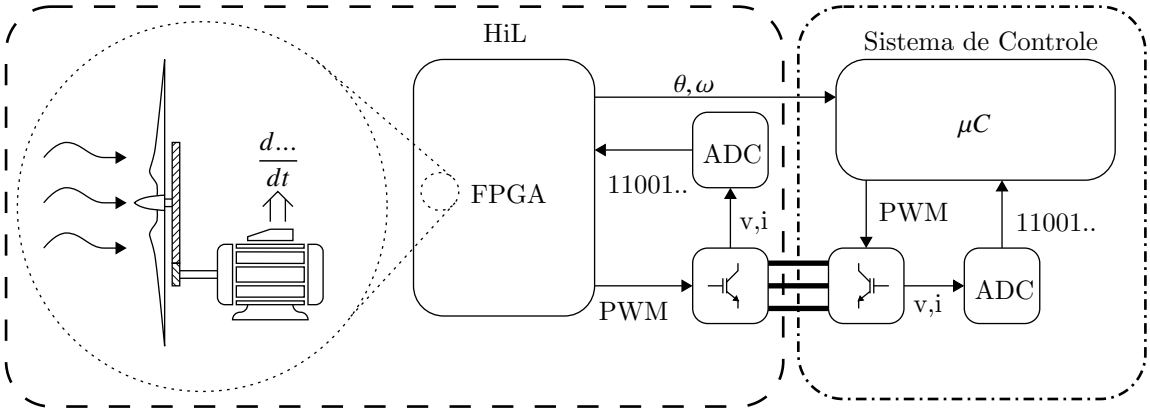


Figura 1: Sistema de *Hardware in the Loop* para emulação de uma turbina eólica.

microssegundos, e dinâmicas lentas, como a variação da incidência solar ou velocidade do vento que estão na escala de minutos ou horas (Bevrani et al., 2017). Dependendo do tipo de estudo, negligenciar uma das escalas de tempo pode impedir a observação de comportamentos presentes no sistema real. Essa eventual necessidade de representar dinâmicas rápidas e lentas podem tornar o modelo complexo e custoso computacionalmente, com isso o uso de dispositivos com alta capacidade de paralelização, como FPGAs, podem agilizar simulações ou até mesmo permitir a emulação em tempo real do sistema, possibilitando o uso de técnicas de teste e validação como *Hardware-in-the-Loop* (HiL). Na Figura 1 é mostrado um sistema de emulação de uma turbina eólica, onde o modelo matemático juntamente com a interface de potência interage com o sistema de controle da mesma maneira como uma turbina real, tanto na interação elétrica de potência como na troca de sinais digitas (angulo e velocidade do rotor), permitindo representações fiéis mesmo não dispondo do sistema real.

A capacidade de executar diversas operações em paralelo é um desafio que tornou as FPGAs em um assunto de interesse na área de processamento de sinais. A possibilidade de processar grandes quantidades de dados em um curto espaço de tempo e baixa latência são características desejáveis em filtros ou analisadores de espectro (Mills et al., 2016; Zhao et al., 2015; Herbert et al., 2016; Seneviratne et al., 2015). No entanto, isso também permite que multiplicações matriciais sejam executadas tornando possível a simulação de dinâmicas rápidas e lentas em uma microrrede (Trivedi and Singh, 2018; Li et al., 2017; Guzman et al., 2014).

O uso de FPGAs para implementar plataformas de simulação em tempo real estão presentes na literatura como opções alternativas quando existem impedimentos nos testes sobre os sistemas reais, devido aos custos, riscos ou necessidade de agilidade nos testes. Contudo, estas soluções são direcionadas a casos de estudos particulares onde é desejado simular apenas um componente de um sistema, por exemplo maquinas indutivas (Liu and Dinavahi, 2018; Tavana and Dinavahi, 2016; Fleming and Edrington, 2016; Karimi et al., 2008). Quando se faz necessário um sistema de emulação para propósito geral, soluções comerciais, como as plataformas Typhoon HIL e dSpace, podem ser extremamente caras, o que pode ser impeditivo caso o estudo não justifique os custos com esses equipamentos.

Neste trabalho, é proposta uma arquitetura para operações vetoriais e matriciais com foco na emulação de sistemas lineares baseada na representação em espaço de estados e que seja simples o suficiente para ser implementada em FPGAs menos sofisticadas, para que o custo seja menor que as soluções comerciais.

Na seção 2 é proposto um componente que efetue o produto interno entre dois vetores e a partir deste componente, na seção 3, é elaborado um multiplicador matricial. Por fim, é mostrado esses elementos são utilizados para efetuar as operações da representação em espaço de estados.

2 Produto interno

Nesta seção, é apresentado o funcionamento do componente capaz de efetuar o produto interno entre dois vetores. Este componente foi projetado para que se utilize a base Q de forma que seja possível operar com números racionais, e também é feita a compensação nos resultados de forma que o erro, devido o deslocamento de bits, seja reduzido.

2.1 Descrição do operador matemático $\langle \cdot, \cdot \rangle$

Definição: Seja um espaço vetorial $V \subseteq \mathbb{R}^n$, o produto interno usual(Boldrini et al., 1980) ($\langle \cdot, \cdot \rangle : V \times V \rightarrow P \subseteq \mathbb{R}$), pode ser definido como:

$$\langle \mathbf{v}_a, \mathbf{v}_b \rangle = \sum_{i=0}^{n-1} v_{ai} \cdot v_{bi}. \tag{1}$$

onde $\mathbf{v}_a, \mathbf{v}_b \in V$ e, v_{ai} e v_{bi} são os elementos de $\mathbf{v}_a$ e $\mathbf{v}_b$, respectivamente, na coordenada i.

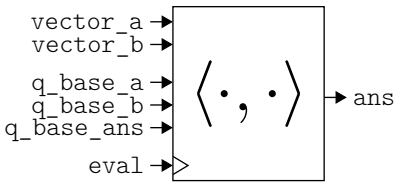


Figura 2: Diagrama representando os sinais de entrada e de saída do componente `inner_product`.

2.2 Realização do operador $\langle\cdot,\cdot\rangle$

Para o funcionamento do componente `inner_product`, foram definidos, a principio, dois sinais de entrada `vector_a` e `vector_b`, que são arranjos unidimensionais de comprimento `n` que representam os vetores $\mathbf{v}_a$ e $\mathbf{v}_b$ mostrados em (1), e um sinal de saída `ans`, que é um número inteiro que representa o resultado do operador $\langle\cdot,\cdot\rangle$ definido em (1).

Como consequência da utilização da base Q e para que o componente possa ser reutilizado repetidas vezes em situações onde são empregadas bases Q com valores distintos, foram adicionados os sinais de entrada `q_base_a` e `q_base_b`, que são arranjos unidimensionais de comprimento `n` que armazenam as bases Q dos elementos dos sinais `vector_a` e `vector_b`, respectivamente. O sinal de entrada `q_base_ans` armazena a base Q do sinal de saída `ans`.

A atualização do sinal de saída `ans`, com o valor do produto interno dos vetores de entrada, é feita quando ocorrer uma borda de subida no sinal de entrada `eval`.

Com estes sinais de entrada e saída, o diagrama que representa esse componente pode ser visto na Figura 2.

O cálculo do produto interno se inicia pela multiplicação alinhada dos elementos do sinal `vector_a` pelos elementos do sinal `vector_b`. O resultado de cada uma destas multiplicações é então armazenada nos elementos do sinal `vector_multiplied`, que é um arranjo unidimensional de comprimento `n` cujos elementos têm o dobro da largura, em bits, do que a largura dos elementos dos sinais de entrada `vector_a` e `vector_b`.

Para reduzir o erro devido o truncamento de dados, causado pelo deslocamento dos bits na etapa de conversão entre bases Q, o sinal `vector_multiplied` é compensado somando-se um valor c. O resultado dessa soma é armazenado no sinal `vector_compensated`. O valor de c é tomado de forma que o valor resultante, após o deslocamento dos bits do sinal `vector_compensated`, seja um valor arredondado ao invés de ser um truncamento para o número, dentro da resolução da base, imediatamente inferior ao número original. O valor de c_i é determinado como:

$$c_i = 2^{(\texttt{q\_base\_a}_i + \texttt{q\_base\_b}_i - \texttt{q\_base\_ans} - 1)}. \qquad (2)$$

A equação (2) é implementada em VHDL como uma operação de deslocamento de bits, onde a quantidade de bits deslocados é igual ao expoente.

Após a etapa de compensação do erro, é feito o deslocamento dos bits para que as bases Q dos números armazenados no sinal `vector_compensated` sejam ajustadas para a base Q do sinal `ans` ao mesmo tempo que é feita uma redução na largura de dados para que seja compatível com o sinal `ans`. O resultado deste deslocamento e redução é armazenado no sinal `vector_shifted`.

Estas etapas de multiplicação, compensação e deslocamento de bits são mostradas no diagrama da Figura 3. Neste diagrama, é mostrado o processo para a multiplicação dos i-ésimos elementos dos vetores $\mathbf{v}_a$ e $\mathbf{v}_b$ e neste exemplo estes vetores são compostos por números de 16 bits. É ressaltada a alteração da largura de dados entre os sinais, explicado pelo fato da multiplicação de dois números inteiros com largura de b bits gerar um resultado de $2b$ bits e, na Figura 3, é evidenciado o resultado de 32 bits gerado pela multiplicação de dois números de 16 bits.

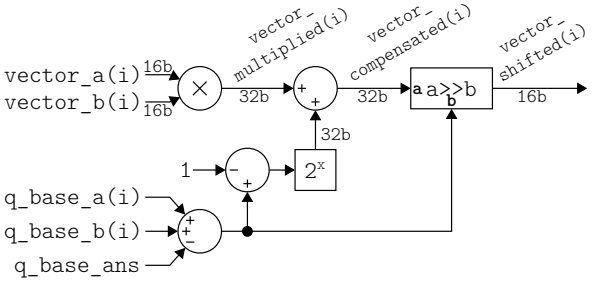


Figura 3: Diagrama representando as etapas de multiplicação dos i-ésimos elementos dos vetores $\mathbf{v}_a$ e $\mathbf{v}_b$, a compensação para diminuição do erro, e deslocamento para o ajuste entre bases. Em alguns sinais é mostrado a largura de dados em bits para enfatizar a mudança desta largura ao longo das operações. Esta estrutura se repete para todos os n elementos dos vetores $\mathbf{v}_a$ e $\mathbf{v}_b$.

Após o deslocamento e redução dos bits, os elementos do sinal `vector_shifted` são somados em cadeia, como mostrado na Figura 4. O resultado dessa soma é colocado na entrada de um *flip-flop* D que aguardará uma borda de subida no sinal `eval` para então transferir o valor da soma para o sinal de saída `ans`.

2.3 Exemplo de aplicação do componente `inner_product`

Um exemplo de aplicação para este componente é a utilização para efetuar as operações de uma equação a diferenças que tenha a forma:

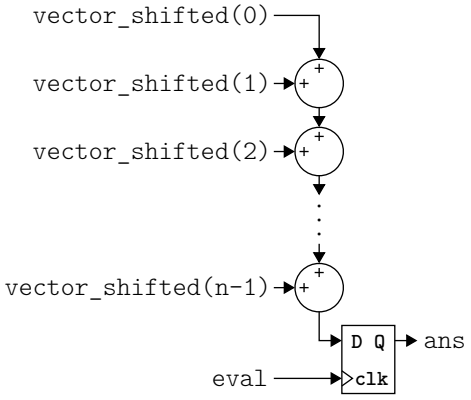


Figura 4: Diagrama representando a soma em cadeia do sinal `vector_shifted` e o *flip-flop* tipo D que retem o valor da soma até que haja uma borda de subida no sinal de entrada `eval`, repassando o valor para o sinal de saída `ans`.

$$y[n] = \sum_{j=1}^{M} c_j \cdot y[n-j] + \sum_{i=0}^{P} d_i \cdot x[n-i] \quad (3)$$

onde $x[n-i]$ são as amostras do sinal de entrada deslocadas no tempo, e $y[n-j]$ são os resultados passados da equação.

Podemos tornar os elementos c_j e d_i em um único vetor $\mathbf{k}$ da seguinte forma:

$$\mathbf{k} = [c_1, \cdots, c_M, \quad d_0, \cdots, d_P]\,. \quad (4)$$

De forma análoga, podemos organizar as amostras $y[n-j]$ e $x[n-i]$ em um vetor $\mathbf{w}[n]$:

$$\mathbf{w}[n] = [y[n-1], \cdots, y[n-M],\ x[n], \cdots, x[n-P]]\,. \quad (5)$$

Desta maneira, utilizando (4) e (5), é possível reescrever a Equação (3) como:

$$y[n] = \langle \mathbf{k}, \mathbf{w}[n] \rangle. \quad (6)$$

A Equação (4) é facilmente implementada em VHDL como dois arranjos unidimensionais de números inteiros, um armazenando os valores dos coeficientes em base Q, e o outro armazenando as bases de cada coeficiente. No entanto, a Equação (5) requer uma estrutura não trivial devido o deslocamento das amostras. Esta estrutura é basicamente dois registradores de deslocamento, um utilizado no deslocamento entre os elementos $y[n-j]$ do vetor $\mathbf{w}[n]$ e o outro para os elementos $x[n-i]$. O funcionamento desta estrutura é mostrado na Figura 5.

Na Figura 5, os *flip-flops* ativos por borda de descida são utilizados por motivos de sincronia, para evitar que o deslocamento ocorra durante uma possível atualização dos valores $y[n]$ ou $x[n]$.

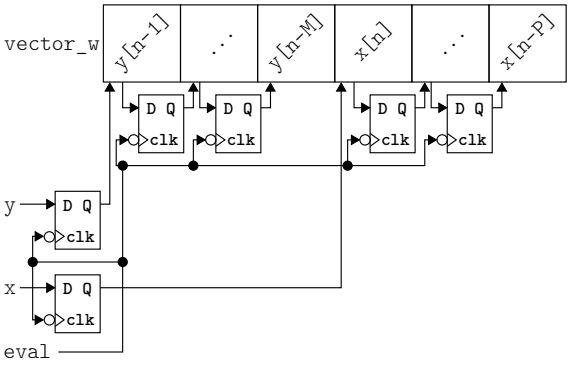


Figura 5: Diagrama representando o deslocamento dos valores armazenados no vetor $\mathbf{w}[n]$.

Para testar o funcionamento deste componente, foi elaborado um filtro passa-baixas de segunda ordem. Este filtro possui uma frequência de corte em 100Hz e foi projetado utilizando a aproximação de Butterworth. O filtro foi discretizado em uma taxa de amostragem de 10kHz utilizando a transformação bilinear e foi feito o *prewarping* para corrigir o desvio da frequência de corte do filtro discretizado. A expressão em equação de diferenças deste filtro é:

$$\begin{aligned} y[n] \;=\; & 1{,}9112y[n-1] \;\; -0{,}914976y[n-2] \\ & +0{,}000945x[n] \;\; +0{,}00189x[n-1]\ . \\ & +0{,}000945x[n-2] \end{aligned} \quad (7)$$

Na Figura 6, é mostrada a resposta do filtro ao sinal senoidal de 100Hz simulado na ferramenta ISIM . Nesta figura, o valor dos sinais de entrada e saída foram convertidos novamente em valores reais. É notável nesta figura que a amplitude do sinal filtrado é de $0{,}706$, valor próximo a uma atenuação de 3dB, o que valida o funcionamento da arquitetura e também o projeto do filtro.

3 Multiplicação Matricial

Nesta seção, é apresentado o funcionamento do componente encarregado de efetuar multiplicações matriciais. Este componente foi projetado de forma que efetue a multiplicação através de produtos internos, de forma que possa ser aproveitado o componente descrito na seção anterior.

Esta seção foi organizada da seguinte maneira: 1. Descrição da operação de multiplicação matricial; 2. Realização da multiplicação matricial em VHDL, onde é descrito como foi implementado o componente; 3. Exemplo de aplicação do componente `matrix_multiplication` mostrando a utilidade do componente.

3.1 Descrição da multiplicação matricial

Definição: Sejam as matrizes $\mathbf{A} = \left[a_{ij}\right]_{m\times n}$ e $\mathbf{B} = [b_{rs}]_{n\times p}$, a multiplicação $\mathbf{AB} = \mathbf{C}_{m\times p} = [c_{uv}]_{m\times p}$

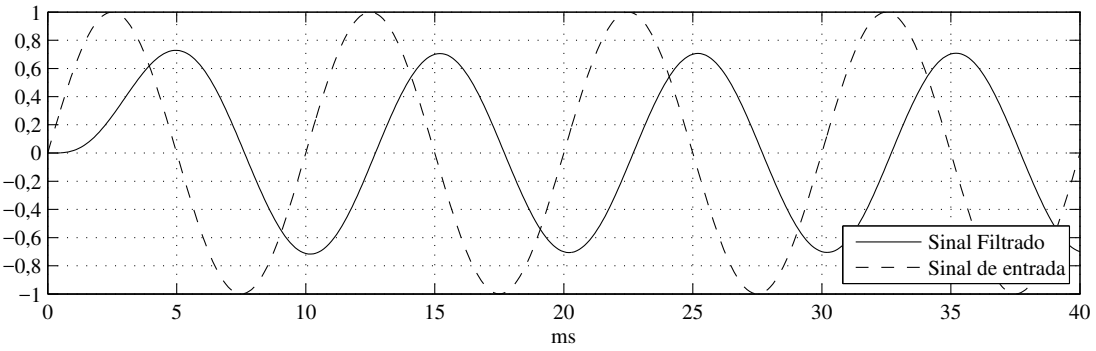


Figura 6: Resposta do filtro passa-baixas para um sinal de 100Hz.

é definida como (Boldrini et al., 1980):

$$c_{uv} = \sum_{k=0}^{n-1} a_{uk} \cdot b_{kv}. \quad (8)$$

Reescrevendo a matriz $\mathbf{A}$ como um arranjo de m vetores linha $\mathbf{a}_i$ de dimensão n e a matriz $\mathbf{B}$ como um arranjo de p vetores coluna $\mathbf{b}_s$, também de dimensão n, temos:

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0 \\ \cdots \\ \mathbf{a}_{m-1} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{b}_0 & \cdots & \mathbf{b}_{p-1} \end{bmatrix}. \quad (9)$$

Utilizando a Equação (9), podemos reescrever (8) como:

$$c_{uv} = \langle \mathbf{a}_u, \mathbf{b}_v \rangle. \quad (10)$$

Desta maneira, é possível reutilizar o componente `inner_product` para implementar um novo componente `matrix_multiplication`.

3.2 Realização da multiplicação matricial

O componente `matrix_multiplication` tem dois sinais de entrada, `matrix_A` e `matrix_B`, que são arranjos bidimensionais de números inteiros com sinal. Estes sinais representam as matrizes $\mathbf{A}$ e $\mathbf{B}$ definidas em (9).

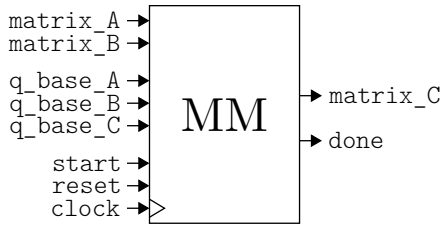


Figura 7: Diagrama representando os sinais de entrada e saída do componente `matrix_multiplication`.

Semelhante ao componente `inner_product`, foram definidos dois sinais bidimensionais, `q_base_A` e `q_base_B`, que armazenam o valor da base Q dos elementos dos sinais `matrix_A` e `matrix_B`, respectivamente. Também é definido um sinal bidimensional, `matrix_C`, que representa a matriz resultante da multiplicação definida em (10), e junto a este sinal, é definido um sinal de entrada bidimensional `q_base_C`, onde são definidos os valores das bases Q dos elementos da matriz resultante.

Uma forma de implementar este multiplicador, seria empregar um componente `inner_product` para cada operação de produto interno definido em (10). Embora esta abordagem efetue a multiplicação de forma mais veloz, ela requer um número elevado de multiplicadores dedicados, o que limitaria as dimensões das matrizes envolvidas na operação. Dado este fato, e para que as dimensões sejam as maiores possíveis, foi implementada uma arquitetura que efetua os produtos internos em série, utilizando um mesmo componente.

Para que fosse implementada uma máquina de estado no componente `matrix_multiplication`, foi necessário um sinal de clock. Foi definido um sinal de entrada, `start`,que sinaliza o inicio da operação, um sinal de reinicialização, `reset`, capaz de interromper a operação em andamento e um sinal de saída, `done`, que informa o fim da operação. A Figura 7 mostra os sinais de entrada e saída deste componente.

Na Figura 9, é mostrada a máquina de estado implementada para efetuar a multiplicação matricial. Nesta máquina de estado, as transições ocorrem quando houver uma borda de descida no sinal `clock` e a condição mostrada for satisfeita. A transição do estado 2 para o estado 3 não possui outra condição além da borda de descida.

No estado 0, é feito uma reinicialização dos sinais internos, `u` e `v`, preparando a máquina para um novo ciclo de operação. Neste estado, o sinal `done` é colocado em nível alto, sinalizando que um novo resultado está pronto e a máquina está preparada para efetuar uma nova multiplicação. Se o sinal `start` estiver em nível lógico baixo, a máquina avança para o estado 1, caso o contrário, ela permanece no mesmo estado.

No estado 1, a máquina aguarda o sinal `start` ser colocado em nível alto, e feito isso, o sinal `done`

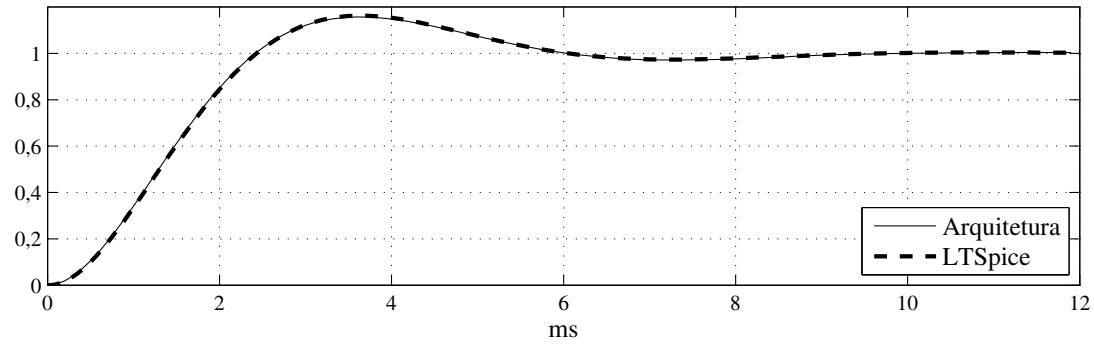


Figura 8: Comparação entre a resposta ao impulso gerada pela arquitetura e o circuito simulado pelo software LTSpice.

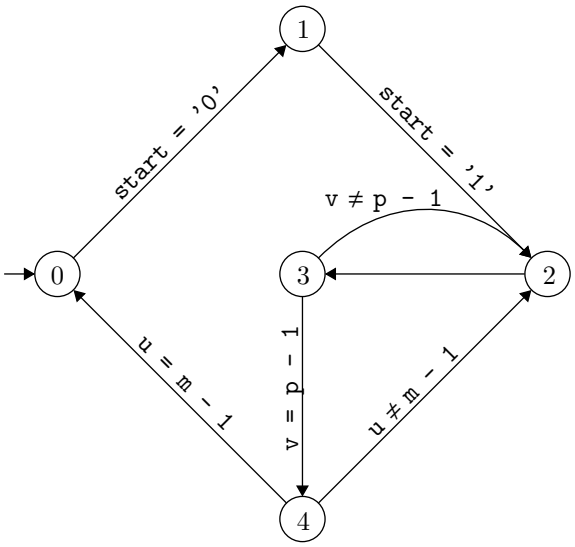


Figura 9: Máquina de estados do componente `matrix_multiplication`.

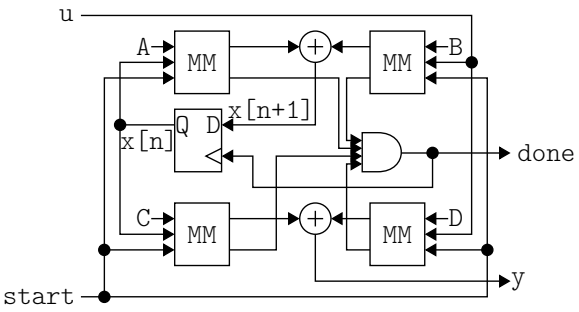


Figura 10: Arquitetura para emulação utilizando a representação por espaço de estados.

é colocado em nível baixo e a máquina avança para o estado 2.

No estado 2, o sinal `eval` do componente `inner_product` utilizado é colocado em nível alto, efetuando assim o produto interno definido em (10) e a máquina avança para o estado 3.

No estado 3, o sinal `eval` é colocado em nível baixo, e o resultado do produto interno é armazenado no elemento c_{uv} da matriz resultante. Neste

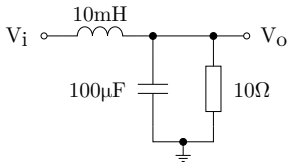


Figura 11: Circuito RLC utilizado no teste do componente `matrix_multiplication` e da arquitetura de emulação.

momento é feita a verificação se o valor do sinal `v` é igual ao número $p-1$, e caso seja, significa que todas as colunas, da linha atual, da matriz resultante foi processada e a máquina segue para o estado 4. Se o valor do sinal `v` for diferente, então é incrementado o seu valor e a máquina retorna ao estado 2 para processar a próxima coluna da linha atual.

No estado 4, a máquina verifica se o valor do sinal `u` é igual à $m-1$. Se esta condição for verdadeira, então significa que todos os elementos da matriz resultante foram processados, e a máquina retorna ao estado 0 para um novo ciclo. Caso o valor seja diferente, o sinal `u` é incrementado e máquina volta ao estado 2 para processar a próxima linha da matriz resultante.

3.3 Exemplo de aplicação do componente `matrix_multiplication`

Uma aplicação para este componente é a emulação de sistemas lineares utilizando a representação em espaço de estados. A representação discretizada em espaço de estados segue a seguinte forma:

$$\begin{cases} \mathbf{x}[n+1] &= \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n] \\ \mathbf{y}[n] &= \mathbf{C}\mathbf{x}[n] + \mathbf{D}\mathbf{u}[n] \end{cases}. \qquad (11)$$

Nesta equação, se torna clara a possibilidade de utilização do componente `matrix_multiplication` nas multiplicações entre as matrizes e os vetores. Dessa maneira, foi elaborada a arquitetura mostrada na Figura 10.

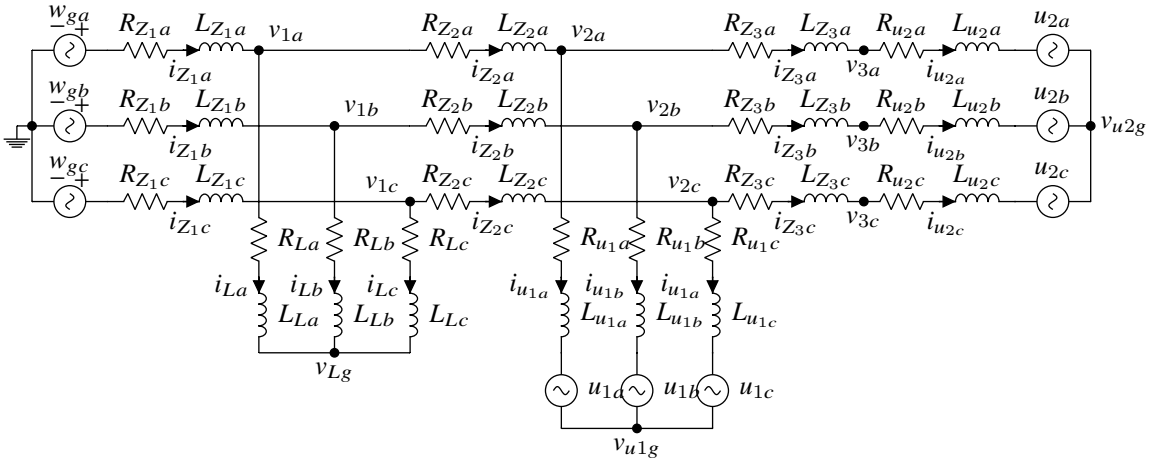
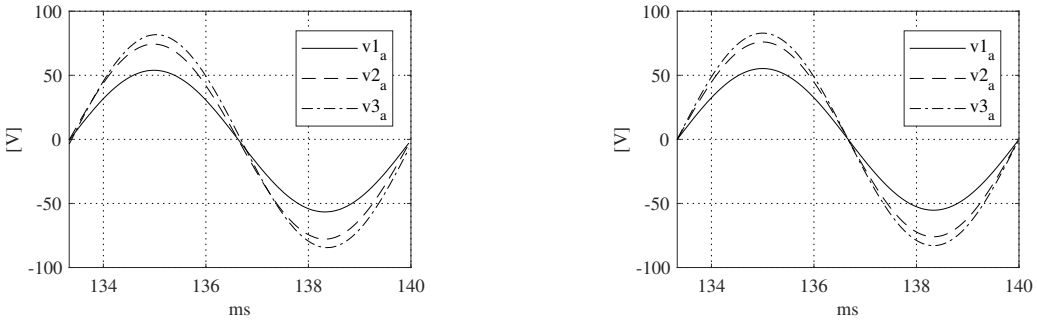


Figura 12: Modelo de rede de distribuição utilizado para o teste de esforço da arquitetura.



(a) (b)

Figura 13: Tensão de fase, referente à fase A, nas três barras de interesse. Resultados obtidos pela arquitetura (a) e MATLAB (b).

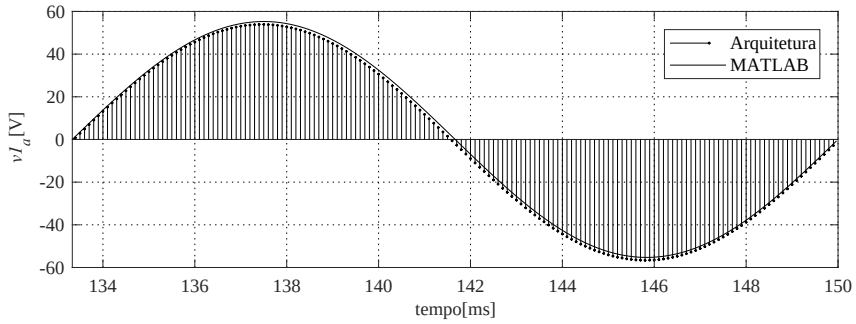


Figura 14: Comparativo entre a precisão dos números em ponto fixo utilizados na arquitetura proposta e o resultado da simulação em MATLAB, em ponto flutuante.

Para teste o funcionamento do componente `matrix_multiplication` e a arquitetura de emulação proposta na Figura 10, foi utilizado o circuito RLC mostrado na Figura 11. O modelo de tempo contínuo é mostrado na Equação (12), e seu equivalente discretizado a uma taxa de 10kHz é mostrado na Equação (13). Na Figura 8, são mostradas as respostas geradas pela arquitetura e pelo circuito simulado na ferramenta LTSpice, onde é possível perceber a semelhança entre as duas respostas, comprovando o funcionamento da arquitetura e do modelo matemático do circuito.

$$\begin{cases} \dot{\mathbf{x}} &= \begin{bmatrix} -\frac{1}{RC} & \frac{1}{C} \\ -\frac{1}{L} & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} \mathbf{u} \\ \mathbf{y} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x} + [0]\,\mathbf{u} \end{cases} . \quad (12)$$

$$\begin{cases} \mathbf{x}[n+1] &= \begin{bmatrix} 0,90016250 & 0,95004083 \\ -0,00950040 & 0,99516658 \end{bmatrix} \mathbf{x}[n] + \\ & \quad \begin{bmatrix} 0,00483341 \\ 0,00998374 \end{bmatrix} \mathbf{u}[n] \\ \mathbf{y}[n] &= \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}[n] + \begin{bmatrix} 0 \end{bmatrix} \mathbf{u}[n] \end{cases} . \quad (13)$$

4 Aplicação da arquitetura na emulação de microrredes

Uma aplicação para a arquitetura proposta é a emulação de sistemas elétricos de potência. Na Figura 12, é mostrado o diagrama unifilar de uma topologia simples, conectada à rede de distribuição pública. Neste modelo, o secundário do transformador é modelado como uma fonte de tensão ideal e as unidades de geração distribuída foram modeladas como uma fonte de tensão ideal em série com um componente indutivo e resistivo. O circuito trifásico equivalente é mostrado na Figura 12 e os referencias de tensão e corrente utilizados na modelagem matemática seguem conforme é mostrado.

Devido uma simplificação, o modelo matemático resultante é de 15ª ordem, possuindo 9 entradas e saídas, porque optou-se por exibir apenas as tensões (trifásicas) $V1$, $V2$ e $V3$. Esta quantidade de entradas e saídas justifica a escolha pela representação em espaço de estados. Na Figura 13 são mostradas as ondas resultantes da emulação feita pelo sistema proposto e o software MATLAB.

Na Figura 14 é possível ver um comparativo entre o resultado da simulação feita em MATLAB e a resposta dada pela arquitetura, onde pose ser observado o desvio causado pelo erro numérico do sistema em ponto fixo (arquitetura) e ponto flutuante (MATLAB).

5 Conclusão

Neste trabalho foram obtidos diversos componentes que podem ser uteis em aplicações onde sejam necessárias operações vetoriais e matriciais, destacando-se processamento de sinais e emulação de sistemas dinâmicos. Estes componentes se mostraram flexíveis, podendo ser utilizados em aplicações distintas sem necessidade de adaptações. A compensação do erro de truncamento trouxe resultados com menor *offset*. A proximidade dos resultados obtidos com resultados teórico ou de ferramentas com histórico de aprovação, mostrou que é possível utilizar números inteiros como alternativa aos números de ponto flutuante.

O uso da aritmética de ponto fixo permitiu a redução do uso dos recursos do FPGA sem comprometer os resultados da emulação.

Como visto, sistemas de distribuição simples podem resultar em modelos complexos, principalmente quando é explicitado todas as fases do sistema. Com visto, a utilização de FPGAs pode tornar realizável a emulação de sistemas elétricos de potência sem recorrer a simplificações que possam prejudicar a representação de certas dinâmicas.

O modelo proposto permite a implementação dos modelos em FPGAs de médio e alto desempenho, que possuam multiplicadores para números inteiros. Embora esses FPGAs possam ser mais caros que computadores de propósito geral ainda custam menos que plataformas de emulação comerciais como *dSpace* e *TyphonHiL*.

Agradecimentos

Este trabalho foi financiado pelo projeto de Pesquisa e Desenvolvimento PD-2866-0468/2017, concedido pela Agência Nacional de Energia Elétrica (ANEEL) e pela Companhia Paranaense de Energia (COPEL). Os autores agradecem também à FINEP, CAPES, SETI, CNPq, Fundação Araucária e UTFPR pelas bolsas e financiamentos adicionais.

Referências

ANEEL (2010). Resolução Normativa Nº 414 de setembro de 2010. Estabelece as Condições Gerais de Fornecimento de Energia Elétrica de forma atualizada e consolidada, *Diário Oficial da União* .

Bevrani, H., Francois, B. and Ise, T. (2017). *Microgrid Dynamics and Control*, John Wiley & Sons, Inc., Hoboken, NJ, USA.

Boldrini, J. L., Costa, S. I. R., Figueiredo, V. L. and Wetzler, H. G. (1980). *Álgebra Linear*, 3 edn, Harper & Row do Brasil, São Paulo.

Bolognani, S. and Zampieri, S. (2010). Distributed Quasi-Newton Method and its Application to the Optimal Reactive Power Flow Problem, *IFAC Proceedings Volumes* **43**(19): 305–310.

Casavola, A., Tedesco, F. and Vizza, M. (2017). Command Governor Strategies for the Online Management of Reactive Power in Smart Grids With Distributed Generation, *IEEE Transactions on Automation Science and Engineering* **14**(2): 449–460.

Fleming, F. E. and Edrington, C. S. (2016). Real-time emulation of switched reluctance machines via magnetic equivalent circuits, *IEEE Transactions on Industrial Electronics* **63**(6): 3366–3376.

Guzman, C., Cardenas, A. and Agbossou, K. (2014). Load sharing strategy for autonomous ac microgrids based on fpga implementation of adaline fll, *IEEE Transactions on Energy Conversion* **29**(3): 663–672.

Hatziargyriou, N. (2013). *Microgrids*, John Wiley and Sons Ltd, Chichester, United Kingdom.

Herbert, J., S. Wilson, A. R. and Taimre, T. (2016). FPGA implementation of a high-speed, realtime, windowed standard deviation filter, *Electronics Letters* pp. 22–23.

Karimi, S., Gaillard, A., Poure, P. and Saadate, S. (2008). Fpga-based real-time power converter failure diagnosis for wind energy conversion systems, *IEEE Transactions on Industrial Electronics* **55**(12): 4299–4308.

Katiraei, F., Iravani, R., Hatziargyriou, N. and Dimeas, A. (2008). Microgrids management, *IEEE Power and Energy Magazine* **6**(3): 54–65.

Kundur, P. (1994). *Power System Stability and Control*, 1 edn, McGraw-Hill Education, Toronto.

Lasseter, R. H. (2002). Microgrids, *2002 IEEE Power Engineering Society Winter Meeting.*, Vol. 1, pp. 305–308 vol.1.

Li, P., Wang, Z., Wang, C., Fu, X., Yu, H. and Wang, L. (2017). Synchronisation mechanism and interfaces design of multi-fpga-based real-time simulator for microgrids, *IET Generation, Transmission Distribution* **11**(12): 3088–3096.

Liu, P. and Dinavahi, V. (2018). Real-time finite-element simulation of electromagnetic transients of transformer on fpga, *IEEE Transactions on Power Delivery* **PP**(99): 1–1.

Mills, A., Jones, P. H. and Zambreno, J. (2016). Parameterizable FPGA-based Kalman Filter Coprocessor Using Piecewise Affine Modeling, *2016 IEEE International Parallel and Distributed Processing Symposium Workshops*, IEEE, pp. 139–147.

Seneviratne, V., Madanayake, A. and Udayanga, N. (2015). Wideband 32-element 200-MHz 2-D IIR beam filters using ROACH-2 Virtex-6 sx475t FPGA, *2015 IEEE 9th International Workshop on Multidimensional (nD) Systems (nDS)*, IEEE, pp. 120–125.

Tavana, N. R. and Dinavahi, V. (2016). Real-time nonlinear magnetic equivalent circuit model of induction machine on fpga for hardware-in-the-loop simulation, *IEEE Transactions on Energy Conversion* **31**(2): 520–530.

Thale, S. S., Wandhare, R. G. and Agarwal, V. (2015). A Novel Reconfigurable Microgrid Architecture With Renewable Energy Sources and Storage, *IEEE Transactions on Industry Applications* **51**(2): 1805–1816.

Trivedi, A. and Singh, M. (2018). L_1 adaptive droop control for ac microgrid with small mesh network, *IEEE Transactions on Industrial Electronics* **65**(6): 4781–4789.

Zhao, G., Ge, Q. and Zhang, Y. (2015). Dynamically Reconfigurable FIR Filter Design Based on FPGA, *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, IEEE, pp. 1631–1635.