Análise e Implementação da Transformada Rápida de Fourier Otimizada

J. P. Cerquinho Cajueiro e G. Jerônimo da Silva Jr.

***Resumo*— Este artigo apresenta uma análise das diferentes implementações da transformada rápidas de Fourier otimizada e da pequena transformada rápida de Fourier de Winograd, os quais são os melhores algoritmos que implementam a transformada discreta de Fourier, em termos de complexidade multiplicativa. As implementações são analisadas quanto ao consumo de blocos lógicos, velocidade máxima de processamento e número de estágios utilizados para a realização da transformada.**

***Palavras-Chave*— FFT, implementação, complexidade multiplicativa, hardware.**

***Abstract*— This paper presents an analysis of implementations of the optimized fast Fourier transformed and of the Winograd small fast Fourier transform, which are the best algorithms that implement the discrete Fourier transform, in terms of multiplicative complexity. The implementations are analyzed regarding the consumption of logic blocks, processing speed and number of steps used to perform the transform.**

***Keywords*— FFT, implementation, multiplicative complexity, hardware.**

I. Introdução

O Processamento digital de sinais passou a ser largamente utilizado com o desenvolvimento dos circuitos integrados digitais, das técnicas de integração VLSI e da tecnologia CMOS para a fabricação de *chips* [1]. Em paralelo a isso, ocorreu uma evolução nas técnicas utilizadas para a implementação de algoritmos aritméticos com um número reduzido de operações aritméticas. Tais algoritmos são comumente conhecidos como algoritmos rápidos. Em 1952, Cooley e Tukey apresentaram um dos mais famosos algoritmos rápidos, a transformada rápida de Fourier (FFT) de Cooley-Tukey [2], que implementa a transformada discreta de Fourier (DFT) [3]. A DFT tem diversas aplicações em telecomunicações e em outras áreas da engenharia [4].

A implementação e o desenvolvimento de algoritmos rápidos passou a ser um ramo amplamente estudado na engenharia [5-7], levando à criação da FFT de Winograd [8], da transformada rápida de Hartley [9], do algoritmo de Goertzel para uma componente da DFT [10], entre outros. De uma maneira geral, transformadas rápidas atuam sobre um vetor de entrada, cujo número de componentes desse vetor é chamado de "comprimento da transformada" e representado por N. As FFT de grandes comprimentos são implementadas utilizando FFT de comprimentos menores, o que pode ser visto como uma "fatoração" de algoritmos [7], assim, as pequenas FFTs[1] determinam a complexidade de vários algoritmos. O avanço das técnicas de fabricação de circuitos integrados a partir do uso da linguagem de descrição de *hardware* (HDL) e o surgimento dos dispositivos FPGA possibilita o projeto e teste do desempenho de um DSP com diferentes FFTs.

O principal parâmetro que quantifica a complexidade computacional de um algoritmo rápido é o número de multiplicações necessário, chamado de complexidade multiplicativa. O número mínimo de multiplicações para a realização da DFT foi calculado na tese de Heideman, em 1980 [11]. A transformada rápida de Fourier otimizada (OFFT) [12], atinge o número mínimo de multiplicações calculado por Heideman para a realização de uma DFT, superando a melhor FFT sistemática conhecida até então, que era a FFT de Winograd [8], em termos da complexidade multiplicativa.

Este artigo apresenta uma metodologia de análise das implementação de transformadas, a partir das matrizes da transformada rápida. Uma arquitetura de *hardware* é proposta para implementar a FFT de Winograd e a OFFT para comparálas, nos comprimentos em que esses algoritmos são diferentes em termos de complexidade[2], sobre os aspectos do consumo de área do dispositivo, tempo de processamento e número de estágios. A próxima seção apresenta uma breve descrição das duas FFTs analisadas. A seção III mostra a arquitetura de implementação, a representação numérica utilizada e a implementação das transformadas de ambos os algoritmos. A seção IV define a metodologia utilizada para análise das transformadas e a seção V apresenta os resultados obtidos. As conclusões são apresentadas na Seção VI. O Apêndice I apresenta alguns algoritmos implementados e o Apêndice II mostra uma implementação em HDL da OFFT.

II. OFFT e a FFT de Winograd

Uma FFT é uma algoritmo rápido que implementa a DFT. A DFT de uma sequência real de N pontos $x[n]$, $n = 0, 1, \ldots, N-1$, é a sequência complexa de N pontos $X[k]$, $k = 0, 1, \ldots, N-1$, definida por

$$X[k] \triangleq \sum_{n=0}^{N-1} x[n] W_N^{nk}, \tag{1}$$

J. P. Cerquinho Cajueiro, Departamento de Engenharia Mecânica, Universidade Federal de Pernambuco, Recife, PE, E-mail: joaopaulo@ee.ufpe.br.

G. Jerônimo da Silva Jr., Grupo de Processamento de Sinais, Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife, PE, E-mail: gilsonjr@gmail.com.

[1]Neste artigo, chama-se "pequena FFT" os algoritmos de FFT implementados para pequenos comprimentos, com analogia ao termo em inglês *small fast Fourier transform* [7].

[2]Para alguns comprimentos, os algoritmo são idênticos em complexidade aritmética [11, 13].

em que $W_N \triangleq e^{-j\frac{2\pi}{N}}$ e $j \triangleq \sqrt{-1}$. Para uma DFT de N pontos, pode-se definir a matriz de transformação

$$W \triangleq \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{(N-1)2} & \dots & W_N \end{bmatrix}, \quad (2)$$

e os vetores

$$\vec{X} \triangleq \begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix} \quad (3)$$

e

$$\vec{x} \triangleq \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \quad (4)$$

para escrever a DFT de N pontos pela equação matricial

$$\vec{X} = W\vec{x}. \quad (5)$$

As principais FFT para pequenos comprimentos são algoritmos que implementam a matriz W na forma fatorada

$$W = CBA, \quad (6)$$

em que as matrizes A e C são matrizes de números racionais, ou seja, tem apenas somas e subtrações mas nenhuma multiplicação, e a matriz B é uma matriz diagonalque portanto apenas tem multiplicações [5, 7, 11, 13], de modo que a complexidade multiplicativa está relacionada à matriz B e a complexidade aditiva está relacionada às matrizes A e C.

Tanto a OFFT quanto a FFT de Wnograd são escritas da forma dada pela equção 6. A OFFT fatora a matriz W utilizando os conceitos de base ciclotômica e a decomposição de um conjunto de matrizes em matrizes de posto unitário [11]. Mostra-se que a menor complexidade multiplicativa é obtida quando se utiliza a OFFT [13]. A pequena FFT de Winograd utiliza uma combinação da permutação de Rader para transformar a DFT numa convolução circular [7], a qual é implementada utilizando o algoritmo de convolução de Winograd [7]. A diferença prática entre as duas FFTs são as matrizes A, B e C.

De modo geral, uma FFT é aplicada a sinais no domínio dos reais, entretanto, algoritmos que utilizam pequenas transformadas, como a FFT de Cooley-Tukey [2] e a FFT de Good-Thomas [14], necessitam calcular a DFT para entradas complexas. Considerando a parte real e imaginária do vetor de entrada, isto é, $\vec{x} = \vec{x}_r + j\vec{x}_i$, então a transformada de $\vec{x}$ é dada por

$$\vec{X} = \vec{X}_r + j\vec{X}_i = W\vec{x}_r + jW\vec{x}_i, \quad (7)$$

que pode ser implementada com duas transformadas reais, como mostra a Figura 1.

As matrizes A e C podem ser fatoradas em matrizes bielementares [15], o que ajuda na determinação da complexidade

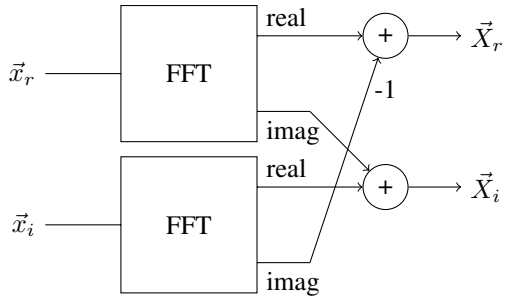

Fig. 1. Transformada discreta de Fourier para entrada complexa, implementada a partir de duas FFTs para entradas reais.

aditiva e diminui o número de adições da implementação, considerando que o sintetizador do circuito a partir da linguagem de descrição de *hardware* (HDL) não faz isso automaticamente.

A seguir, são apresentados aspectos importantes da implementação dessas transformadas em linguagem de descrição de *hardware*, bem como a metodologia utilizada para a comparação das FFTs.

III. Arquiteturas para a Implementação da FFT

Dois aspectos importantes na implementação de transformadas são a representação numérica para os valores das componentes dos vetores $\vec{x}$ e $\vec{X}$ e a arquitetura dos estágios das FFT. Uma possível arquitetura é através do uso de registradores com uma técnica conhecida como *pipeline* [16], outra possibilidade é através da execução de comandos sequenciais via máquina de estados. Neste trabalho foi escolhido o paradigma de *pipeline*, de modo que cada estágio corresponde à aplicação de uma matriz.

Para representar todos os números foi escolhida a notação Q1.31 complemento a 2, um tipo de notação ponto fixo de 32 bits [17]. Essa escolha permite que adições e subtrações sejam implementadas de maneira mais simples e as multiplicações de inteiros podem ser feitas utilizando blocos multiplicadores, descartando os bits menos significativos.

Um outro parâmetro variado na implementação foi o formato das matrizes A e C. Em um caso cada uma delas foi implementada como um único estágio de *pipeline*, enquanto que no outro elas foram implementadas por vários estágios, definidos pela fatoração bielementar [15], de modo que cada estágio apresenta uma única complexidade aditiva para cada 2 ramos ou uma multiplicação por ramo da *pipeline*. Multiplicações por $\pm 2^k$, com k inteiros são implementadas por deslocamento e não produzem atraso. Essas multiplicações são consideradas "multiplicações triviais" [5, 7, 12, 13]. As figuras 2 e 3 mostram exemplos desta implementação para o algoritmo de Winograd e para a OFFT, respectivamente, quando $N = 5$.

O número de matrizes bielementares utilizadas para decompor as matrizes A e C determina o número de estágios no *pipeline*. Para $N = 5$, por exemplo, nota-se que a matriz A_o da OFFT é decomposta em 2 matrizes bielementares, enquanto no algoritmo de Winograd são necessárias 3 matrizes, portanto, esse algoritmo requer um estágio a mais na *pipeline* que a

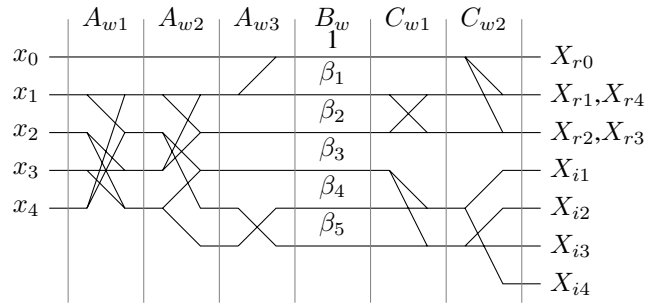

Fig. 2. Implementação da pequena FFT de Winograd, para $N = 5$, em pipeline, com fatoração em matrizes bielementáres aplicada às matrizes A e C.

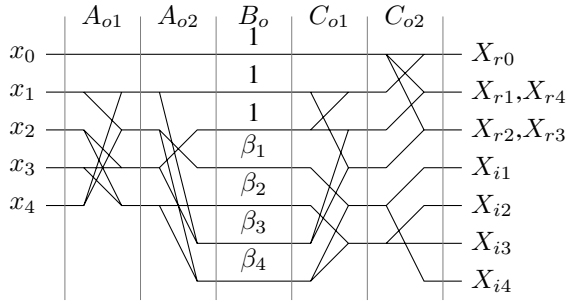

Fig. 3. Implementação da OFFT, para $N = 5$, em pipeline, com fatoração em matrizes bielementáres aplicada às matrizes A e C.

OFFT, exigindo um tempo maior para a computação em ciclos de máquina e um maior número de registradores.

A próxima seção apresenta a análise da OFFT e da pequena FFT de Winograd utilizando a arquitetura e a representação numérica definida.

IV. Implementações Propostas e Parâmetros de Síntese

Todas as implementações foram sintetizadas utilizando o programa *Quartus II 11.1 web edition*, da Altera, usando como dispositivo alvo o FPGA EP3C120F780 da família *Cyclone* III[3]. Este dispositivo possui blocos multiplicadores de 9 bits que podem ser inferidos a partir do HDL e que são bem mais rápidos que multiplicadores feitos a partir de blocos lógicos. Tendo em vista o uso destes elementos, foram feitas duas sínteses para cada descrição: uma habilitando o uso dos multiplicadores embarcados de 9 bits e a outra desabilitando este uso. Utilizou-se *Verilog* como HDL. O Apêndice II mostra a implementação em *Verilog* da OFFT para $N = 3$, utilizando as matrizes A e C originais.

Para cada FFT sintetizada, analisou-se as seguintes características: número de blocos lógicos utilizados, número de blocos multiplicadores embarcados (se habilitados), velocidade máxima do *clock* do sistema e o número de estágios da implementação. O parâmetro de velocidade é calculado automaticamente pelo software e leva em consideração a tensão de alimentação e temperatura de operação do circuito.

[3]No *website* "www.altera.com" encontram-se informações sobre o dispositivo FPGA e o sintetizador *Quartus* utilizado nas implementações.

Neste trabalho, considerou-se o pior caso: tensão de $1.2\,\mathrm{V}$ e temperatura de $85\,^{\circ}\mathrm{C}$.

V. Análise das Implementações Propostas

Os resultados obtidos são mostrados nas Tabelas I, II, e III. Nestas tabelas o campo *mult.* é o número de blocos multiplicadores de 9 bits usados (0 quando desabilitados) e o campo matriz é quanto ao uso das matrizes A e C da forma fatorada ou original.

TABELA I
RESULTADOS DA SÍNTESE DAS FFTS PARA $N = 3$.

Tipo	blocos	clock [MHz]	mult.	matriz	estágios
Winograd	478	116,1	8	fatorada	4
	870	99,85	0	fatorada	4
OFFT	474	116,13	8	fatorada	4
	869	99,74	0	fatorada	4
Winograd	476	116,18	8	original	3
	864	102,93	0	original	3
OFFT	474	114,86	8	original	3
	866	97,24	0	original	3

TABELA II
RESULTADOS DA SÍNTESE DAS FFTS PARA $N = 5$.

Tipo	blocos	clock [MHz]	mult.	matriz	estágios
Winograd	1289	114,85	32	fatorada	6
	2686	92,5	0	fatorada	6
OFFT	1169	115,62	32	fatorada	5
	2577	93,91	0	fatorada	5
Winograd	1255	114,76	32	original	3
	2666	94,74	0	original	3
OFFT	1190	114,96	32	original	3
	2604	93,92	0	original	3

TABELA III
RESULTADOS DA SÍNTESE DAS FFTS PARA $N = 7$.

Tipo	blocos	clock [MHz]	mult.	matriz	estágios
Winograd	2413	113,48	64	fatorada	7
	5446	94,99	0	fatorada	7
OFFT	2412	115,26	56	fatorada	7
	5143	64,3	0	fatorada	7
Winograd	2445	111,23	64	original	3
	5453	93,73	0	original	3
OFFT	2388	104,06	56	original	3
	4857	93,14	0	original	3

Pelo resultado, é possível verificar dois aspectos importantes. O primeiro é que a implementação utilizando a fatoração das matrizes A e C em matrizes bielementares não melhora, significativamente, a velocidade do dispositivo, como esperado. Uma explicação para isso é o fato da multiplicação ser bem mais complexa, em termos de *hardware*, que as adições, para a representação numérica utilizada, de modo que o bloco de multiplicadores (matriz B) limita a velocidade. Nota-se também que a velocidade não mudou significativamente entre os algoritmos.

O segundo aspecto é o fato que, em algumas situações, a implementação utilizando matrizes bielementares consumiu

mais blocos lógicos que a implementação utilizando as matrizes originais. A explicação desse resultado só foi solucionada através da visualização da implementação em RTL [18], disponível no programa *Quartus*. Nessa visualização, observa-se que a sintetização do programa simplifica o resultado a nível de porta lógica e não de somadores, isso confere ao sintetizador um nível mais baixo e mais detalhista de simplificação, não considerado na fatoração bielementar.

Uma curiosidade verificada na sintetização é que a complexidade multiplicativa da pequena FFT de Winograd como calculada em [7, 8, 11, 13] para os comprimentos 3 e 5, difere da obtida neste trabalho. Isto se deve a uma das multiplicações contabilizadas ser entre uma variável e um número racional (ou seja, é trivial), o que é implementado pelo sintetizador por um deslocamento e uma soma. Com esta consideração, a complexidade multiplicativa da FFT de Winograd para $N = 3$ é uma multiplicação e para $N = 5$ são quatro multiplicações. Isso explica a igualdade no número de multiplicadores para esses algoritmos nesses comprimentos.

VI. Conclusões

Este trabalho analisou diferentes implementações da OFFT e da pequena FFT de Winograd para os comprimentos 3, 5 e 7 (são os comprimentos que têm diferença nas complexidades multiplicativas) em *Verilog* para o dispositivo EP3C120F780 da Altera. Descobriu-se que os algoritmos analisados, nos comprimento 3 e 5, têm a mesma complexidade multiplicativa.

Um resultado interessante para a implementação com ponto fixo de 32 bits é que o estágio de multiplicação limita a velocidade de processamento, devido a alta complexidade de *hardware*. A fatoração em matrizes bielementares não melhora a velocidade de processamento. A implementação utilizando as matrizes originais, utilizando a OFFT, é a maneira mais indicada para esta situação, pois diminui o número de registradores, o número de estágios e o número de blocos lógicos. Em alguns casos, a sintetização a nível de blocos lógicos do programa *Quartus* é mais eficiente que a fatoração em matrizes bielementares, que sintetiza a nível de somadores.

Referências

[1] A. S. Sedra and K. C. Smith, *Microeletrônica*, 5th ed. Pearson Prentice Hall, 2007.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[3] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 3rd ed. New Jersey: Prentice Hall, 2010.

[4] I. Z. Emiris and V. Y. Pan, "Algorithms and theory of computation handbook," M. J. Atallah and M. Blanton, Eds. Chapman & Hall/CRC, 2010, ch. Applications of FFT and structured matrices, pp. 18–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882757.1882775

[5] S. Winograd, *Arithmetic Complexity of Computations*. Bristol: SIAM Publications, 1980.

[6] L. I. Kronsjö, *Algorithms: Their Complexity and Efficiency*. John Wiley & Sons, 1979.

[7] R. E. Blahut, *Fast Algorithms for Signal Processing*, 2nd ed. New York: Cambridge University Press, 2010.

[8] S. Winograd, "On computing the discrete Fourier transform," *Mathematics of Computation*, vol. 32, pp. 175–199, Jan. 1978.

[9] R. Bracewell, "The fast Hartley transform," *Proceedings of the IEEE*, vol. 72, no. 8, pp. 1010–1018, 1984.

[10] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *The American Mathematical Monthly*, vol. 65, no. 1, pp. 34–35, 1958.

[11] M. T. Heideman, *Multiplicative Complexity, Convolution, and the DFT*. New York: Springer-Verlag, 1988.

[12] G. J. da Silva Jr. and R. M. Campello de Souza, "Transformada rápida de Fourier otimizada," *XXIX Simpósio Brasileiro de Telecomunicações*, vol. 24, p. 5, Outubro 2011.

[13] G. J. da Silva Jr., "A teoria da complexidade aritmética aplicada à otimização de transformadas lineares," Ph.D. dissertation, Universidade Federal de Pernambuco - UFPE, Recife - PE, 2012.

[14] I. J. Good, "The interaction algorithm and practical Fourier analysis," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, pp. 361–372, August 1958.

[15] G. J. da Silva Jr. and R. M. Campello de Souza, "Teoria da complexidade aditiva para transformadas," *XXIX Simpósio Brasileiro de Telecomunicações*, vol. 24, p. 5, Outubro 2011.

[16] W. Stallings, *Arquitetura e Organização de Computadores*, 8th ed. Pearson, 2010.

[17] W. T. Padgett and D. V. Anderson, "Fixed-point signal processing," *Synthesis Lectures on Signal Processing*, vol. 4, no. 1, pp. 1–133, 2009.

[18] M. D. Ercegovac, T. Lang, and J. H. Moreno, *Introdução aos Sistemas Digitais*. Bookman, 2000.

[19] G. J. da Silva Jr. and R. M. Campello de Souza, "Cyclotomic basis for computing the discrete Fourier transform," *International Telecommunications Symposium*, vol. 7, pp. 1–5, September 2010.

Apêndice I
Algoritmos Implementados

A. *OFFT para* $N = 5$

$$A_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix},$$

$$B_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta_4 \end{bmatrix},$$

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} = \begin{bmatrix} -j[\text{sen}(\frac{2\pi}{5}) - \text{sen}(\frac{4\pi}{5})] \\ -j[\text{sen}(\frac{2\pi}{5}) + \text{sen}(\frac{4\pi}{5})] \\ \cos(\frac{2\pi}{5}) \\ -j\text{sen}(\frac{4\pi}{5}) \end{bmatrix}$$

e

$$C_o = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\frac{1}{2} & 1 & 0 & 1 & 1 \\ 1 & -\frac{1}{2} & 0 & 0 & 2 & -1 & 1 \\ 1 & -\frac{1}{2} & 0 & 0 & -2 & -1 & -1 \\ 1 & 0 & -\frac{1}{2} & -1 & 0 & 1 & -1 \end{bmatrix}.$$

B. OFFT para $N = 7$

$$A_o = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & -1 & 0 & -1 \\ 0 & 1 & 1 & -1 & 1 & -1 & -1 \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & \frac{1}{4} & -\frac{5}{4} & -\frac{5}{4} & \frac{1}{4} & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix},$$

$$B_o(i,i) = \begin{bmatrix} -j\frac{1}{3}[\text{sen}(\frac{2\pi}{7}) - 2\text{sen}(\frac{4\pi}{7}) - \text{sen}(\frac{6\pi}{7})] \\ -j\frac{1}{3}[2\text{sen}(\frac{2\pi}{7}) - \text{sen}(\frac{4\pi}{7}) + \text{sen}(\frac{6\pi}{7})] \\ -j\frac{1}{3}[\text{sen}(\frac{2\pi}{7}) + \text{sen}(\frac{4\pi}{7}) + 2\text{sen}(\frac{6\pi}{7})] \\ -j\frac{1}{3}[\text{sen}(\frac{2\pi}{7}) + \text{sen}(\frac{4\pi}{7}) - \text{sen}(\frac{6\pi}{7})] \\ \cos(\frac{2\pi}{7}) - \frac{1}{2}\cos(\frac{4\pi}{7}) \\ \cos(\frac{2\pi}{7}) + \frac{5}{4}\cos(\frac{4\pi}{7}) \\ \frac{1}{2}\cos(\frac{4\pi}{7}) \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

e

$$C_o = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & -\frac{1}{2} \\ -1 & 1 & 0 & 1 & 0 & -2 & 2 & 1 & 0 & -\frac{1}{2} & 0 \\ 0 & 1 & 1 & -1 & -1 & 1 & -3 & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & -1 & -1 & 1 & -1 & 1 & -3 & 1 & -\frac{1}{2} & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & -2 & 2 & 1 & 0 & -\frac{1}{2} & 0 \\ -1 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & -\frac{1}{2} \end{bmatrix}.$$

C. Outras Implementações

A implementação da OFFT para $N = 3$ pode ser encontrada em [19]. As implementações da pequena FFT de Winograd podem ser encontradas no Apêndice de [7], em diversos comprimentos.

APÊNDICE II
IMPLEMENTAÇÃO EM HDL

Implementação em *Verilog* do módulo que computa a DFT de comprimento 3 utilizando as matrizes A e C originais.

```
module o2fft3(reset,clk,
              v0,v1,v2,
              Vr0,Vr1,Vr2,
              Vi0,Vi1,Vi2);
  input reset, clk;
  input signed [31:0] v0,v1,v2;
  output reg signed [31:0] Vr0,Vr1,Vr2;
  output reg signed [31:0] Vi0,Vi1,Vi2;

  wire signed [31:0]       alout[2:0];
  wire signed [63:0]       bout[2:0];
  wire signed [31:0]       Vrt[2:0];
  wire signed [31:0]       Vit[2:0];

  reg signed [31:0]        alout_reg[2:0];
  reg signed [31:0]        bout_reg[2:0];

  always @(posedge clk or posedge reset)
    if (reset) begin
       alout_reg[0] = 0;
       alout_reg[1] = 0;
       alout_reg[2] = 0;

       bout_reg[0] = 0;
       bout_reg[1] = 0;
       bout_reg[2] = 0;

       Vr0 = 0;
       Vr1 = 0;
       Vr2 = 0;

       Vi0 = 0;
       Vi1 = 0;
       Vi2 = 0;

  end else begin// if (reset)
       alout_reg[0] = alout[0] ;
       alout_reg[1] = alout[1] ;
       alout_reg[2] = alout[2] ;

       bout_reg[0] = bout[0][62:31];
       bout_reg[1] = bout[1][62:31];
       bout_reg[2] = bout[2][62:31];

       Vr0 = Vrt[0];
       Vr1 = Vrt[1];
       Vr2 = Vrt[2];

       Vi0 = Vit[0];
       Vi1 = Vit[1];
       Vi2 = Vit[2];
end

//A
assign alout[0] = v0 + v1 + v2;
assign alout[1] = v0 - {v1[31],v1[31:1]}-{v2[31],v2[31:1]};
assign alout[2] = v1 - v2;

//B
assign bout[0] = {alout_reg[0][31],alout_reg[0],31'b0}; //1
assign bout[1] = {alout_reg[1][31],alout_reg[1],31'b0}; //1
assign bout[2] = alout_reg[2]*(-1859775393);     // -sin t
.
//C
assign Vrt[0] = bout_reg[0];
assign Vrt[1] = bout_reg[1];
assign Vrt[2] = bout_reg[1];

assign Vit[0] = 0;
assign Vit[1] = bout_reg[2];
assign Vit[2] = -bout_reg[2];

endmodule
```