UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UMA TRANSFORMADA RÁPIDA DE FOURIER PARA COMPUTAÇÃO RECONFIGURÁVEL DE ALTO DESEMPENHO

TRABALHO DE GRADUAÇÃO

Tiago de Albuquerque Reis

Santa Maria, RS, Brasil

2008

IMPLEMENTAÇÃO DE UMA TRANSFORMADA RÁPIDA DE FOURIER PARA COMPUTAÇÃO RECONFIGURÁVEL DE ALTO DESEMPENHO

por

Tiago de Albuquerque Reis

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Drª Andrea Schwertner Charão

Co-orientador: Dr. Haroldo Fraga de Campos Velho

Trabalho de Graduação Nº 259
Santa Maria, RS, Brasil

2008

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

IMPLEMENTAÇÃO DE UMA TRANSFORMADA RÁPIDA DE FOURIER PARA COMPUTAÇÃO RECONFIGURÁVEL DE ALTO DESEMPENHO

elaborado por
Tiago de Albuquerque Reis

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª Drª Andrea Schwertner Charão
(Presidente/Orientador)

Prof. Dr. Benhur de Oliveira Stein (UFSM)

Prof. Dr. Giovani Baratto (UFSM)

Santa Maria, 1 de fevereiro de 2008.

AGRADECIMENTOS

Agradeço inicialmente à minha família, em especial: aos meus avós Nildo e Gladir, que foram praticamente meus segundos pais; minha madrinha Nilda, uma das maiores culpadas (se não a maior) por eu fazer computação ao me presentear meu primeiro computador, e também por todo seu carinho e apoio sempre que precisei; ao meu irmão, Lucas, por me mostrar o quão mais tranqüila e chata seria a vida de filho único; ao meu padrinho Paulo, pelo seu apoio e por (juntamente com meu avô) me fazer ser colorado; e por último e mais importante, à minha mãe que simplesmente foi meu maior exemplo de vida.

Agradeço à minha orientadora Andrea Charão, por me despertar interesse em ser pesquisador (e professor) e, principalmente, pela enorme dedicação e paciência na minha iniciação como tal. Ao meu co-orientador Haroldo de Campos Velho, pela oportunidade de utilizar o INPE nesse trabalho e por sua orientação. À banca avaliadora, Benhur Stein e Giovani Baratto, por sua grande contribuição para a melhoria deste trabalho. Agradeço também a todos os professores do curso que através de bons e maus exemplos, ajudaram na minha formação.

À minha namorada, Thiely, que apesar da falta de paciência nos apertos de final de semestre, sempre esteve ao meu lado, amenizou os momentos difíceis e melhorou ainda mais os bons momentos.

Agradeço meus colegas por formarem uma turma sem igual e me ajudarem a agüentar a faculdade. Em especial ao Cristiano, Eduardo, Leandro, Márcio, Matheus e Rodolfo que além de colegas foram grandes amigos nessa jornada. EDF para sempre. Também a todos os membros do LSC, que ajudaram na minha iniciação científica.

Agradeço o Centro de Processamento de Dados da UFSM pela oportunidade de estágio e, em especial, a toda a Divisão de Redes e Sistemas Básicos (antigo Suporte) pelos ensinamentos e companheirismo.

Agradeço também a todos os meus amigos que de uma forma ou de outra me ajudaram a chegar até aqui.

"Tem gente que gasta seu tempo com compras, eu gasto meu tempo pirando nos sons da floresta." — XISPA DIVINA

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO DE UMA TRANSFORMADA RÁPIDA DE FOURIER PARA COMPUTAÇÃO RECONFIGURÁVEL DE ALTO DESEMPENHO
Autor: Tiago de Albuquerque Reis
Orientador: Prof[a] Dr[a] Andrea Schwertner Charão
Co-orientador: Dr. Haroldo Fraga de Campos Velho
Local e data da defesa: Santa Maria, 1 de fevereiro de 2008.

A computação reconfigurável é uma abordagem de desenvolvimento de sistemas em que a aplicação define a estrutura do processador. Essa abordagem vem ganhando espaço na computação de alto desempenho, pois permite criar sistemas híbridos implementados como uma combinação de *hardware* e *software*, apresentando desempenho superior ao de sistemas implementados unicamente em *software*. Neste trabalho, tem-se como objetivo a implementação de uma transformada rápida de Fourier para possível utilização em computação reconfigurável. Essa implementação pode ser usada para aumentar o desempenho de aplicações que utilizem esse método numérico. Este trabalho se faz importante por explorar este novo paradigma de computação de alto desempenho e, ao mesmo tempo, fazer com que a aplicações aproveitem melhor os recursos de *hardware* disponíveis em um computador híbrido Cray XD1, equipado com processadores reconfiguráveis do tipo FPGA (*Field-programmable Gate Array*). Para tal, deciciu-se implementar em FPGA um algoritmo que calcula uma transformada rápida de Fourier por esse algoritmo constituir um núcleo computacional bastante utilizado em diversas aplicações na área de meteorologia. Essa implementação foi simulada e apresentou resultados satisfatórios em número de ciclos de relógio necessários para realizar a função.

Palavras-chave: FPGA, transformada rápida de Fourier, computação reconfigurável, computação de alto desempenho.

ABSTRACT

Graduation Work
Graduation Program in Computer Science
Federal University of Santa Maria

IMPLEMENTATION OF A FAST FOURIER TRANSFORM FOR HIGH PERFORMANCE RECONFIGURABLE COMPUTING

Author: Tiago de Albuquerque Reis
Advisor: Prof[a] Dr[a] Andrea Schwertner Charão
Coadvisor: Dr. Haroldo Fraga de Campos Velho

Reconfigurable computing is an approach to system design where the application defines the processor structure. This approach is useful for high performance computing, allowing the creation of hybrid systems as a combination of hardware and software, presenting a higher performance than software-only implementations. This work has the goal of implementing a fast Fourier transform for possible use in reconfigurable computing. This implementation can be used to increase the performance of applications that make use of such numerical method. This work is important for exploring this new high performance paradigm and, at the same time, make applications use more efficiently the hardware resources available in a Cray XD1 supercomputer, equipped with FPGA (Field-programmable Gate Array) reconfigurable processors. The fast Fourier transform algorithm was chosen because it is a computational kernel used in several meteorological applications. This implementations was simulated and presented a satisfactory number of clock cycles to execute this function.

Keywords: FPGA, fast Fourier transform, reconfigurable computing, high performance computing.

LISTA DE FIGURAS

Figura 2.1 – Arquitetura de FPGA genérica. Fonte: (MAXFIELD, 2004) 15
Figura 2.2 – Elementos de um bloco lógico programável. Fonte: (MAXFIELD, 2004) 16

Figura 3.1 – Operação de borboleta - DIT 21
Figura 3.2 – Operação de borboleta - DIF 21
Figura 3.3 – Grafo do algoritmo *radix-2* por decimação no tempo 22
Figura 3.4 – Grafo do algoritmo *radix-2* por decimação na freqüência 23

Figura 4.1 – Visão lógica do chassis do XD1. Fonte: (CRAY, 2005a) 24
Figura 4.2 – Conexões do FPGA. Fonte: (CRAY, 2005a) 25
Figura 4.3 – Topo de um projeto HDL para o Cray XD1. Fonte: (CRAY, 2005a) . . . 28
Figura 4.4 – *RapidArray Transport Core*. Fonte: (CRAY, 2005b) 29

Figura 5.1 – Portas do módulo que calcula o fator *twiddle* 31
Figura 5.2 – Portas do módulo calcula a borboleta 32
Figura 5.3 – Arquitetura da borboleta implementada 32
Figura 5.4 – Arquitetura da FFT implementada 33
Figura 5.5 – Algoritmo FFT implementado em VHDL 34
Figura 5.6 – Resultado obtido em simulação 34
Figura 5.7 – Resultado obtido em programa de alto nível 35
Figura 5.8 – Portas utilizadas do módulo *RapidArray Transport Client* 36
Figura 5.9 – Estrutura dos *records* 37
Figura 5.10 – Exemplo de requisição via *hardware* 38
Figura 5.11 – Portas do módulo FFT 38
Figura 5.12 – Arquitetura implementada 39

LISTA DE TABELAS

Tabela 3.1 – Exemplo de ordenação por bit-invertido 21

Tabela 5.1 – Ciclos de relógio necessários para calcular a FFT 40

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
DIF	Decimation-in-frequency
DIT	Decimation-in-time
ESL	Eletronic System Level
FFT	Fast Fourier Transform
FPGA	Field-programmable Gate Array
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronic Engineering
IP	Intellectual Property
LUT	Lookup Table
QDR	Quad Data Rate
RAM	Random Access Memory
SRAM	Static Random Access Memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

SUMÁRIO

1 INTRODUÇÃO 12

2 COMPUTAÇÃO RECONFIGURÁVEL 14
2.1 Introdução 14
2.2 FPGA 15
2.3 Programação de FPGAs 16
2.3.1 VHDL 16
2.3.2 Ferramentas ESL 17
2.3.3 Ambiente de programação escolhido 18

3 TRANSFORMADA DE FOURIER 19
3.1 Introdução à transformada de Fourier 19
3.1.1 Contínua 19
3.1.2 Discreta 19
3.2 Transformada rápida de Fourier 20
3.2.1 Algoritmos *radix-2* 20
3.2.2 Outros algoritmos 21

4 CRAY XD1 24
4.1 Arquitetura 24
4.1.1 Interconexão *RapidArray* 25
4.1.2 Recursos de memória 25
4.2 Programando o Cray XD1 26
4.2.1 FPGA Control Utility 26
4.2.2 Operações administrativas 26
4.2.3 Transferência de Dados 27
4.3 Configurando o FPGA do XD1 28

5 IMPLEMENTAÇÃO 30
5.1 Introdução 30
5.2 Implementação da operação borboleta em VHDL 30
5.3 Implementação da FFT em VHDL 33
5.4 Conectando a FFT nos módulos da Cray 35
5.5 Avaliação 38

6 CONCLUSÃO 41

REFERÊNCIAS 42

1 INTRODUÇÃO

A computação reconfigurável é um ramo emergente na computação de alto desempenho que utiliza processadores especiais, ditos reconfiguráveis, cujos circuitos são definidos pelo algoritmo a ser executado. Sua utilização em ambientes de alto desempenho geralmente se dá em sistemas híbridos implementados em uma combinação de *hardware* e *software*. Essa abordagem tem se mostrado eficiente, pois explora níveis adicionais de paralelismo (PETERSON; SMITH, 2001). Por si só, a computação reconfigurável tem mostrado *speed-ups* significativos em relação a implementações somente em *software* (LEVINE, 1999).

Como exemplo de dispositivo reconfigurável podemos citar o *Field-programmable Gate Array* (FPGA), dispositivo semi-condutor com blocos lógicos e interligações programáveis. Os blocos lógicos, também chamados células lógicas, podem ser programados para realizar desde simples funções lógicas até funções matemáticas complexas.

A vantagem da utilização de FPGAs se dá pela possibilidade de alterações no próprio *design* do processador, ao invés de utilizar processadores programáveis por *software* (PETERSEN, 1995). Em arquiteturas de alto desempenho, uma tendência é a construção de sistemas que combinam processadores tradicionais e processadores reconfiguráveis. Nestes casos, as aplicações precisam ser adaptadas para tirar proveito do *hardware* reconfigurável. Para evitar o porte total de uma aplicação para FPGA, que é uma operação onerosa, pode-se implementar em *hardware* apenas certos núcleos computacionais que consomem grande parte do tempo de processamento e são empregados em diferentes aplicações. Um exemplo deste tipo de operação é a transformada rápida de Fourier, que está presente em diversas aplicações na área de meteorologia.

Uma transformada rápida de Fourier (*Fast Fourier Transform* - FFT) é um algoritmo eficiente para calcular a transformada discreta de Fourier e sua inversa. A FFT revo-

lucionou o uso de polinômios trigonométricos interpolatórios, organizando o problema de forma que o número de pontos usados pode ser facilmente fatorado em potências de dois (BURDEN; FAIRES, 1997). Esse método é de grande importância para várias aplicações, como resolução de equações diferenciais parciais e multiplicação de inteiros grandes (COOLEY; TUKEY, 1965).

O principal objetivo deste trabalho é implementar uma transformada rápida de Fourier em um processador reconfigurável. Essa transformada rápida de Fourier poderá ser aplicada no modelo meteorológico DYNAMO (VELHO; CLAEYSSEN, 1992; LYNCH; Ireland Meteorological Service, 1984), em uso no Instituto Nacional de Pesquisas Espaciais (INPE). Com isso, espera-se melhorar o desempenho de aplicações meteorológicas que têm esse método numérico como núcleo computacional, fazendo com que ele execute no FPGA.

A metodologia de desenvolvimento deste trabalho compreende um estudo aprofundado da transformada rápida de Fourier, seus algoritmos e dificuldades de implementação. Além disso, é necessária uma familiarização com a programação para FPGAs e seus ambientes de desenvolvimento, para então implementar, testar e simular a transformada rápida de Fourier em *hardware*.

O restante deste documento está organizado da seguinte forma: no capítulo 2 apresenta-se uma revisão sobre computação reconfigurável, processadores reconfiguráveis e configuração de FPGAs; no capítulo 3 descreve-se a transformada de Fourier, suas variações e algoritmos. No capítulo 4 apresenta-se o Cray XD1 e seus detalhes de programação e configuração. Por fim, no capítulo 5 descreve-se a implementação e apresenta-se uma avaliação da implementação proposta.

2 COMPUTAÇÃO RECONFIGURÁVEL

Este capítulo apresenta uma revisão de conceitos e tecnologias para computação reconfigurável. Inicialmente apresenta-se uma breve introdução a este paradigma e, em seguida, discute-se FPGAs e sua utilização no contexto da computação reconfigurável. Por fim, apresenta-se uma introdução à programação de FPGAs, mostrando as opções e ferramentas para esse fim.

2.1 Introdução

Computação reconfigurável é um paradigma de computação que combina o alto desempenho do *hardware* com a flexibilidade do *software* utilizando processadores reconfiguráveis (TODMAN et al., 2005). Essa abordagem tem como objetivo preencher a lacuna entre o *hardware* e o *software*, alcançando assim um desempenho maior que o *software* enquanto mantém um nível de flexibilidade maior que o *hardware* (COMPTON; HAUCK, 2002). Na computação de alto desempenho, a utilização de computação reconfigurável é vantajosa pela possibilidade de explorar múltiplos tipos e níveis de paralelismo (PETERSON; SMITH, 2001).

O conceito de computação reconfigurável foi proposto na década de 1960 por Gerald Estrin, cuja idéia era ter um arranjo de processadores reconfiguráveis controlados por um processador tradicional. Nessa arquitetura, o processador configuraria o *hardware* reconfigurável para executar uma determinada tarefa e, assim que esse terminasse o processamento, o reconfiguraria para outra tarefa (ESTRIN et al., 1963).

Várias tentativas de se criar computadores reconfiguráveis apareceram nas décadas de 1980 e 1990, mas não obtiveram sucesso por limitações de tecnologia. Até que em 1991 surgiu o primeiro computador reconfigurável, o Algotronix CHS2X4 (KEAN, 2007). Processadores reconfiguráveis somente se firmaram comercialmente com o FPGA, inventado

em 1984 por Ross Freeman.

2.2 FPGA

Um FPGA é um dispositivo semi-condutor que possui blocos lógicos e interconexões programáveis. Esse dispositivo permite a implementação de circuitos lógicos relativamente grandes que consistem de um arranjo desses blocos lógicos e interconexões, contidos em um único circuito integrado. Assim, ele pode ser utilizado para computação híbrida, onde parte da execução se dá em um processador tradicional controlado por *software* e outra parte em um processador programado exclusivamente para realizar determinada tarefa.

Um FPGA é composto de três componentes básicos: blocos lógicos, chaves de interconexão e blocos de entrada e saída. Sua estrutura pode ser vista na figura 2.1, com exceção dos blocos de entrada e saída que se localizam nas extremidades da malha.

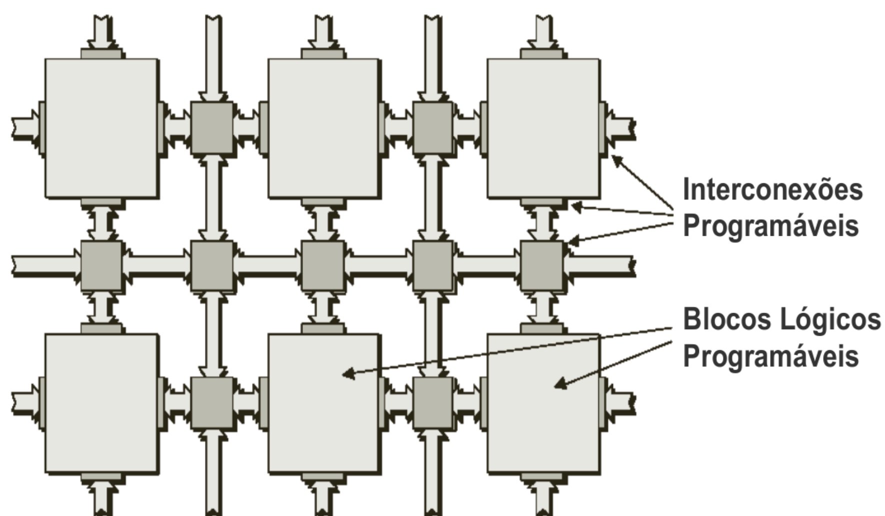

Figura 2.1: Arquitetura de FPGA genérica. Fonte: (MAXFIELD, 2004)

Os blocos lógicos são formados por uma tabela de busca, também chamada LUT (*Lookup Table*), um multiplexador e um *flip-flop*. Um exemplo de bloco lógico, como mostrado na figura 2.2, possui quatro entradas, duas saídas e um sinal de relógio. A tabela de busca é utilizada para implementar a lógica do bloco, visto que uma tabela de busca com *n* bits pode codificar um função Booleana de *n* entradas através de tabela verdade.

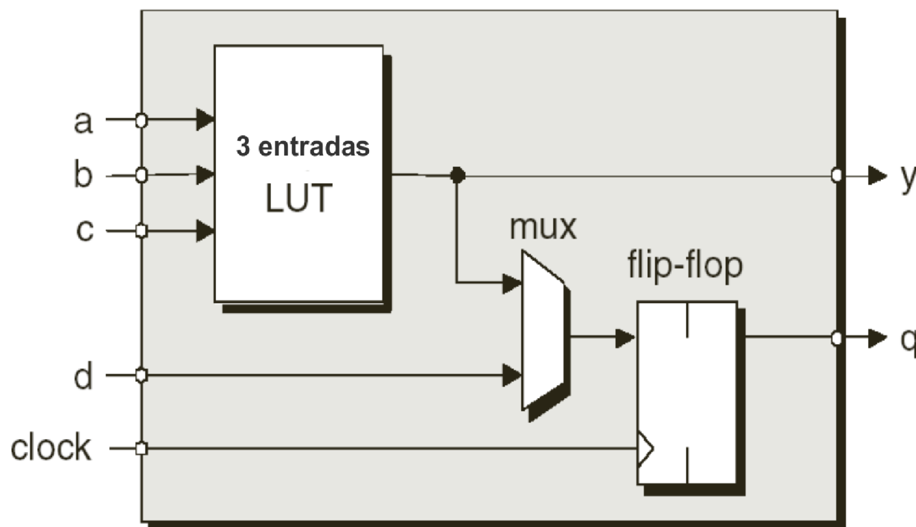

Figura 2.2: Elementos de um bloco lógico programável. Fonte: (MAXFIELD, 2004)

2.3 Programação de FPGAs

A definição do comportamento de um FPGA, assim como qualquer outro dispositivo de *hardware*, pode ser feita através de uma linguagem de descrição de *hardware* (*Hardware Description Language* - HDL). O circuito projetado em HDL é transformado em um *netlist*, utilizando alguma ferramenta de automação de *design* eletrônico, que então será transformada em um arquivo binário que configura o FPGA, utilizando ferramentas disponibilizadas pelos fabricantes do dispositivo.

2.3.1 VHDL

Esta linguagem surgiu na década de 1980 como linguagem de descrição de *hardware* VHSIC (*Very High Speed Integrated Circuits*) e em 1987 foi aprovado como padrão da IEEE (*Institute of Electrical and Electronic Engineering*) (MARRA et al., 2001). Atualmente é, juntamente com Verilog, a linguagem de descrição de *hardware* mais utilizada.

Do ponto de vista do desenvolvedor de *hardware*, VHDL é de grande utilidade, pois permite definir o comportamento do *hardware* de maneira parecida com linguagens de alto nível e permite também definir portas de entrada e saída do componente de maneira intuitiva.

Como linguagem de descrição de *hardware*, VHDL é bastante poderosa, mas do ponto de vista do programador, ela é uma linguagem de desenvolvimento lento, pois o programador precisa se preocupar com vários detalhes inexistentes em linguagens de alto nível.

Utilizar VHDL para programação é comparável à programar em linguagem Assembly.

Pensando nisso, principalmente pelo crescente uso de FPGAs em aplicações de alto desempenho, exitem esforços para aumentar a produtividade dos programadores através da metodologia ESL (*Eletronic System Level*).

2.3.2 Ferramentas ESL

ESL é um novo paradigma que promete alavancar a produtividade de desenvolvimento para FPGAs, elevando o nível de abstração em relação a linguagens de descrição de *hardware*. Com ele é possível desenvolver um programa em uma linguagem de alto nível, como C, e traduzi-lo de maneira otimizada em *hardware* (LASS, 2006).

Para o desenvolvimento deste trabalho, pesquisou-se algumas soluções deste tipo a fim de selecionar uma ferramenta adequada e, ao mesmo tempo, conhecer os avanços recentes nesta área. Estas ferramentas serão apresentadas nas próximas seções.

2.3.2.1 FpgaC

FpgaC é um compilador de código aberto para a linguagem C que gera, ao invés de binários, um circuito digital que executa o programa compilado. Tem como objetivo melhorar a eficiência do desenvolvimento para computação reconfigurável, substituindo as linguagens de descrição de *hardware* por uma linguagem de alto nível (PENTINMAKI; MENON; BASS, 2006).

Um programa em C compilado com FpgaC gera um *netlist* para ser transformado, utilizando ferramentas do fabricante do FPGA, no binário que é carregado para o FPGA.

Apesar de ser um alternativa que pode facilitar bastante o desenvolvimento de uma aplicação para FPGA, não será utilizado por incompatibilidade com a versão da ferramenta disponibilizada pelo fabricante do FPGA que será programado.

2.3.2.2 C2VHDL

C2VHDL é um tradutor de C para VHDL, desenvolvido na Universidade do Estado do Rio de Janeiro. Também tem como objetivo facilitar a construção de *hardware* através de linguagens de alto nível, mas de forma mais genérica pois um código VHDL pode ser importado para qualquer ferramenta de geração de binários para FPGA (MARRA et al., 2001).

Os desenvolvedores do C2VHDL não disponibilizam a ferramenta, o que impossibilita

sua utilização nesse trabalho.

2.3.2.3 Outras ferramentas

Existem diversas ferramentas para tradução de C para HDL (VHDL e/ou Verilog) e compilação de C para *netlists*. Dentre elas podemos citar: Impulse C, Catapult C e Handel C. Todas estas ferramentas são comercializadas a preços em torno de milhares de dólares.

2.3.3 Ambiente de programação escolhido

As ferramentas pesquisadas possuem características indesejáveis para este trabalho, como incompatibilidade com o modelo de FPGA (Virtex II Pro, da Xilinx) a ser utilizado e alto custo de aquisição. Assim, optou-se por implementar a solução diretamente em VHDL. Outro fator que levou a essa decisão foi o fato de que o algoritmo da FFT não é muito extenso e, portanto, o ganho de tempo com a utilização de ESL não compensaria o tempo gasto pesquisando outras opções.

Para a implementação em VHDL, foi utilizada a ferramenta VHDL Simili 3.1 (Symphony EDA, 2007), desenvolvido pela Symphony EDA. Esta escolha deu-se pela familiaridade obtida com essa ferramenta em outras oportunidades. Com essa ferramenta é possível editar, compilar e simular códigos em VHDL.

Para integração com FPGA será utilizada a ferramenta *Foundation* 9.2i da Xilinx. É através dela que será gerado o binário para ser carregado pelo FPGA.

3 TRANSFORMADA DE FOURIER

Neste capítulo apresenta-se uma introdução à transformada de Fourier. Inicialmente apresenta-se a transformada e sua discretização. Na seqüência, descreve-se a transformada rápida de Fourier, alguns algoritmos para seu processamento e suas características.

3.1 Introdução à transformada de Fourier

Formulada por Jean-Baptiste Joseph Fourier, a transformada de Fourier mostra que funções não periódicas podem ser expressas por integrais de funções senoidais de freqüências diferentes, cada uma multiplicada por um coeficiente próprio, desde que a área sob a curva dessa função seja finita.

3.1.1 Contínua

Para uma função contínua de uma variável f(t), a transformada de Fourier F(f) é definida por

$$F(f) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ft}dt$$

e sua inversa por

$$f(t) = \int_{-\infty}^{\infty} F(f)e^{j2\pi ft}df$$

onde $e^{j\theta} = cos(\theta) + jsen(\theta)$ e $j = \sqrt{-1}$.

3.1.2 Discreta

Seja uma série de números complexos x(k) com N amostras, a transformada X(k) é definida por

$$X(n) = \sum_{k=0}^{N-1} x(k)W_N^{kn}\text{, para n = 0, 1, ..., N-1}$$

e sua inversa por

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}, \text{ para n = 0, 1, ..., N-1}$$

onde $W_N = e^{-j2\pi/N}$ (fator *twiddle*).

Podemos observar que para calcular X(k) para um determinado valor de k, são necessárias N multiplicações complexas e N-1 somas complexas. Ou seja, são necessárias N^2 multiplicações complexas e N(N-1) somas complexas para se obter os N valores da série transformada.

Em termos de esforço computacional, a transformada discreta tem complexidade O(N^2), o que a caracteriza como lenta e ineficaz para aplicações de alto desempenho.

3.2 Transformada rápida de Fourier

Para otimizar a resolução da transformada discreta foi proposto um método eficiente para esse cálculo: a transformada rápida de Fourier (COOLEY; TUKEY, 1965).

A transformada rápida de Fourier (*Fast Fourier Transform* - FFT), também chamada algoritmo Cooley-Tukey, tem como idéia básica dividir a transformada discreta em transformadas menores recursivamente, afim de diminuir o esforço computacional. Existem diversos algoritmos para o cálculo da FFT, entre eles, os *radix-2* serão apresentados com mais detalhes nas próximas seções.

3.2.1 Algoritmos *radix-2*

Esses algoritmos foram desenvolvidos por Cooley e Tukey em 1965 e necessitam que N se seja potência de 2, daí seu nome. Podem ser aplicados no domínio do tempo, chamado de decimação no tempo ou DIT (*decimation-in-time*) e no domínio da freqüência, chamado de decimação na freqüência ou DIF (*decimation-in-frequency*).

A equação da transformada discreta pode ser reescrita utilizando a operação chamada "borboleta"(figuras 3.1 e 3.2), que é aplicada a dois pontos da série de cada vez. Essa operação consiste de uma multiplicação complexa e duas somas complexas.

Para aplicar o algoritmo de decimação no tempo a entrada precisa ser rearranjada para a ordem bit-invertido de índices (vide tabela 3.2.1), onde os elementos são separados em dois grupos de acordo com o bit menos significativo na primeira separação, com o segundo bit menos significativo na segunda separação e assim sucessivamente. Depois de re-arranjada, a série passa por $log_2 N$ etapas onde $N/2$ borboletas (como a da figura 3.1) são aplicadas, conforme a figura 3.3, onde o algoritmo é aplicado a uma série de 8

Tabela 3.1: Exemplo de ordenação por bit-invertido

Índice	0	1	2	3	4	5	6	7
Equivalente em binário	000	001	010	011	100	101	110	111
Binário em bit-invertido	000	100	010	110	001	101	011	111
Índice em bit-invertido	0	4	2	6	1	5	3	7

amostras.

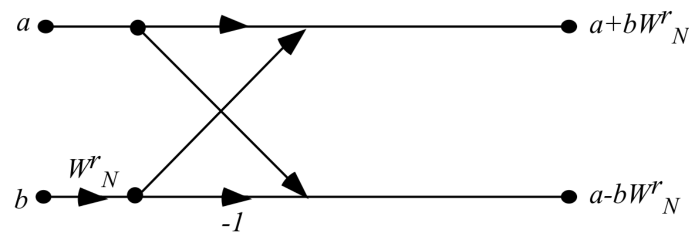

Figura 3.1: Operação de borboleta - DIT

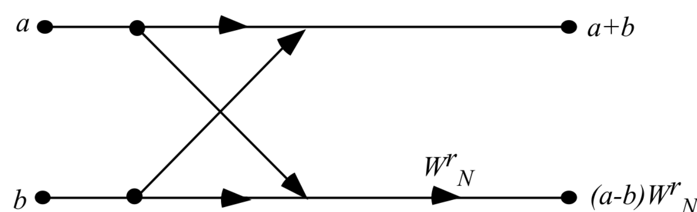

Figura 3.2: Operação de borboleta - DIF

O algoritmo de decimação na freqüência é ligeiramente diferente, tendo como entrada a série na ordem normal, mas seu resultado é ordenado em bit-invertido. Possui o mesmo número de etapas e borboletas (como a da figura 3.2), mas é aplicado conforme a figura 3.4.

Esses algoritmos aplicam um total de $(N/2)log_2N$ operações de borboleta, totalizando $(N/2)log_2N$ multiplicações complexas e $Nlog_2N$ somas complexas, o que em termos de esforço computacional significa uma complexidade de ordem O($Nlog_2N$).

3.2.2 Outros algoritmos

Os algoritmos *radix-2* podem ser estendidos para *radix-4*, *radix-8*, *radix-16*, que melhoram o desempenho por necessitarem de menos etapas e menos operações de borboleta por etapa, mas necessitam que o tamanho das séries sejam potências de 4, 8 e 16 respectivamente.

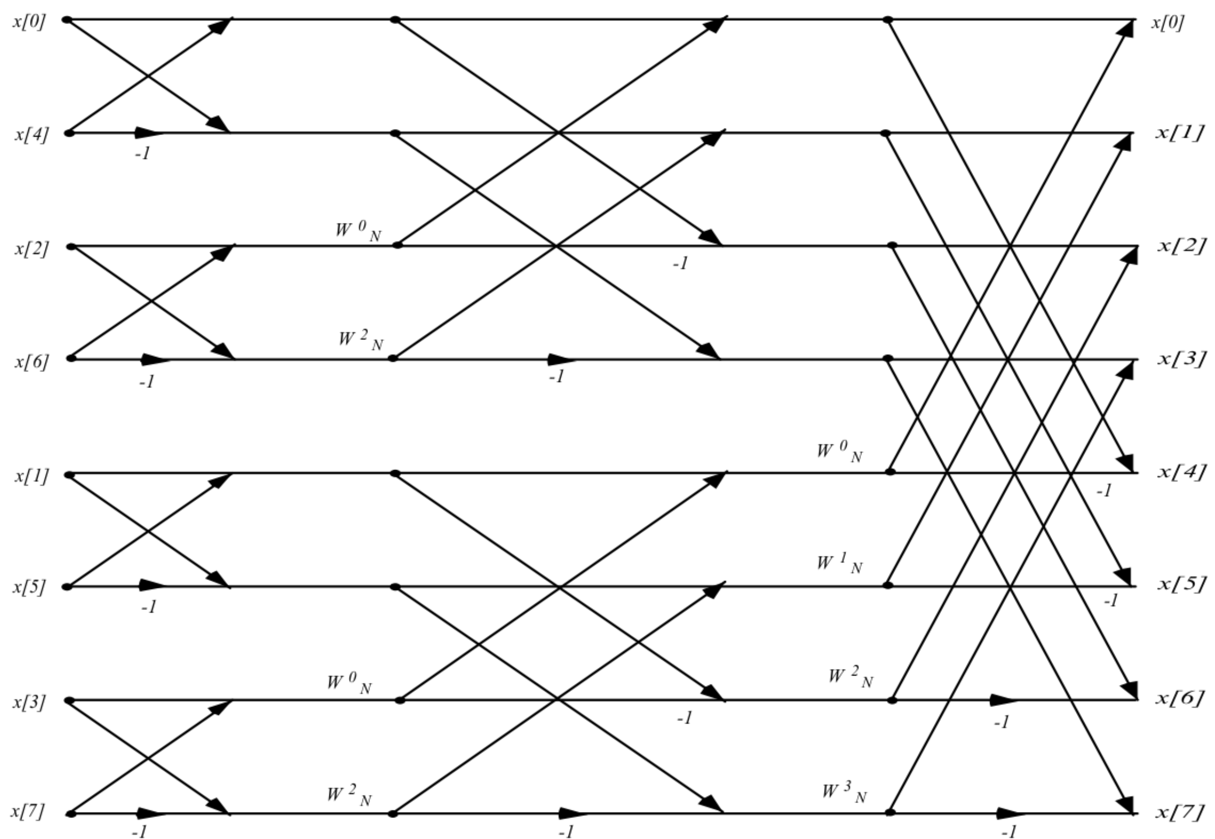

Figura 3.3: Grafo do algoritmo *radix-2* por decimação no tempo

Existem outras tentativas de melhorar o desempenho da FFT, como o algoritmo *split-radix* proposto por R. Yavne (YAVNE, 1968), que mescla borboletas de 2 e 4 entradas para diminuir o número total de operações aritméticas. Na sua forma original também possui a desvantagem de necessitar que o tamanho da série seja de potência de 4, mas pode ser combinado com qualquer outro algoritmo de FFT para suprir essa desvantagem.

Existem ainda outras variações como os algoritmos *prime-factor* (GOOD, 1958), de Bruun (BRUUN, 1978), de Rader (RADER, 1968) e de Bluestein (BLUESTEIN, 1968), mas suas características estão fora do escopo deste trabalho.

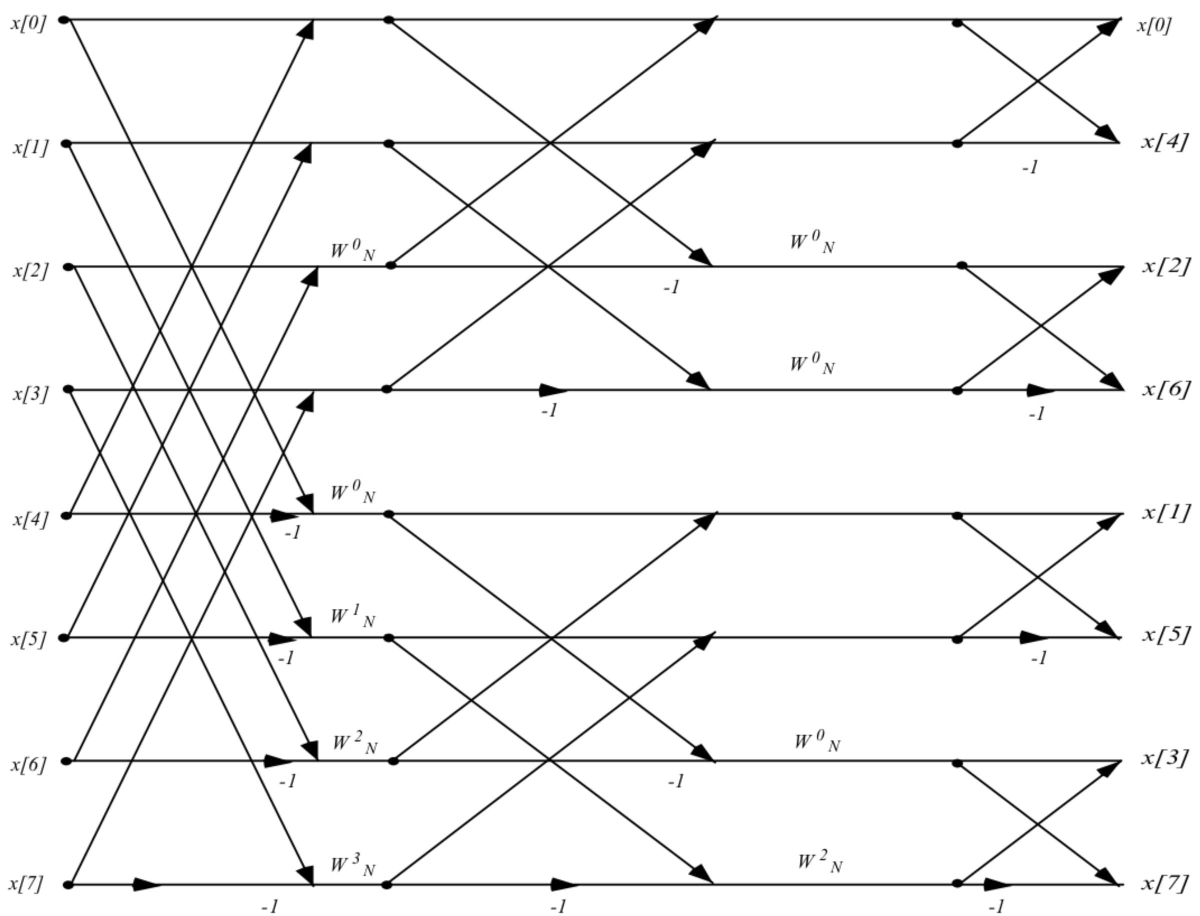

Figura 3.4: Grafo do algoritmo *radix-2* por decimação na freqüência

4 CRAY XD1

Neste capítulo apresenta-se uma descrição da arquitetura do supercomputador XD1 da Cray e suas particularidades de programação e utilização. Esse capítulo visa ressaltar aspectos importantes para o desenvolvimento de aplicações híbridas.

4.1 Arquitetura

O XD1 é composto por seis *blades*, onde cada *blade* possui dois processadores Opteron de 64 bits, de 1 a 8 GB de memória DDR SDRAM por processador, dois processadores *RapidArray*, um FPGA Virtex II Pro da Xilinx e um processador de gerenciamento, que monitora o *hardware* (CRAY, 2005a). Uma visão lógica do chassis do Cray XD1 por ser vista na figura 4.1.

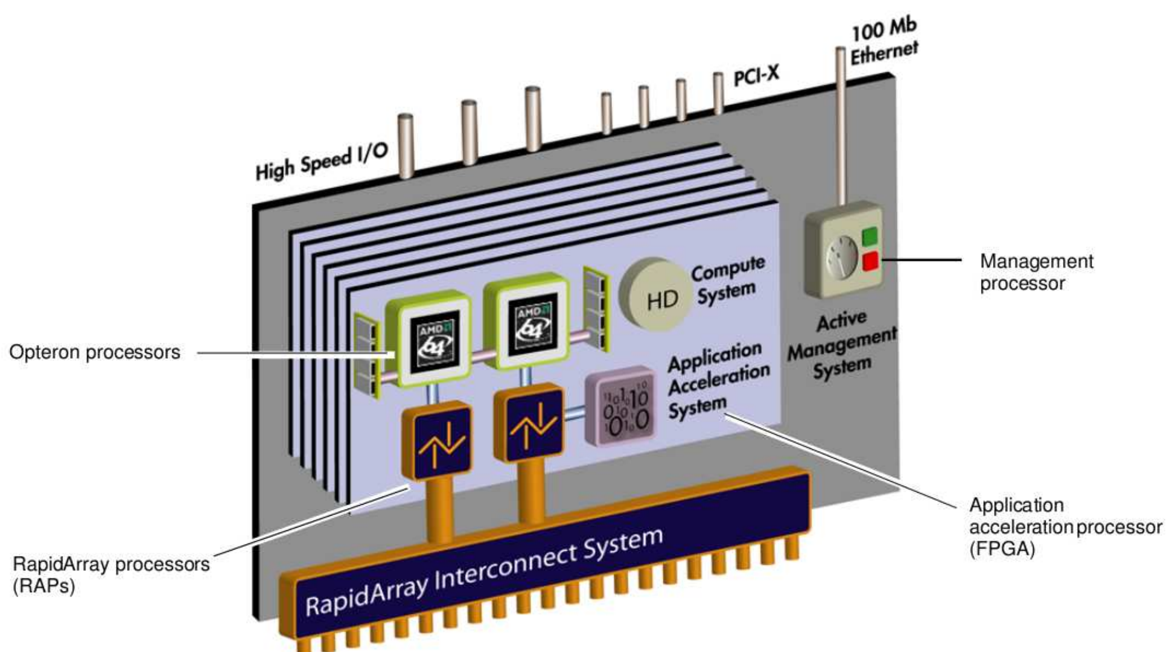

Figura 4.1: Visão lógica do chassis do XD1. Fonte: (CRAY, 2005a)

O FPGA se conecta com um dos processadores *RapidArray* locais para comunicação com os processadores Opteron, com processadores *RapidArray* de *blades* vizinhas e com quatro módulos de memória QDRII SRAM de 64 Mbytes, como mostra a figura 4.2.

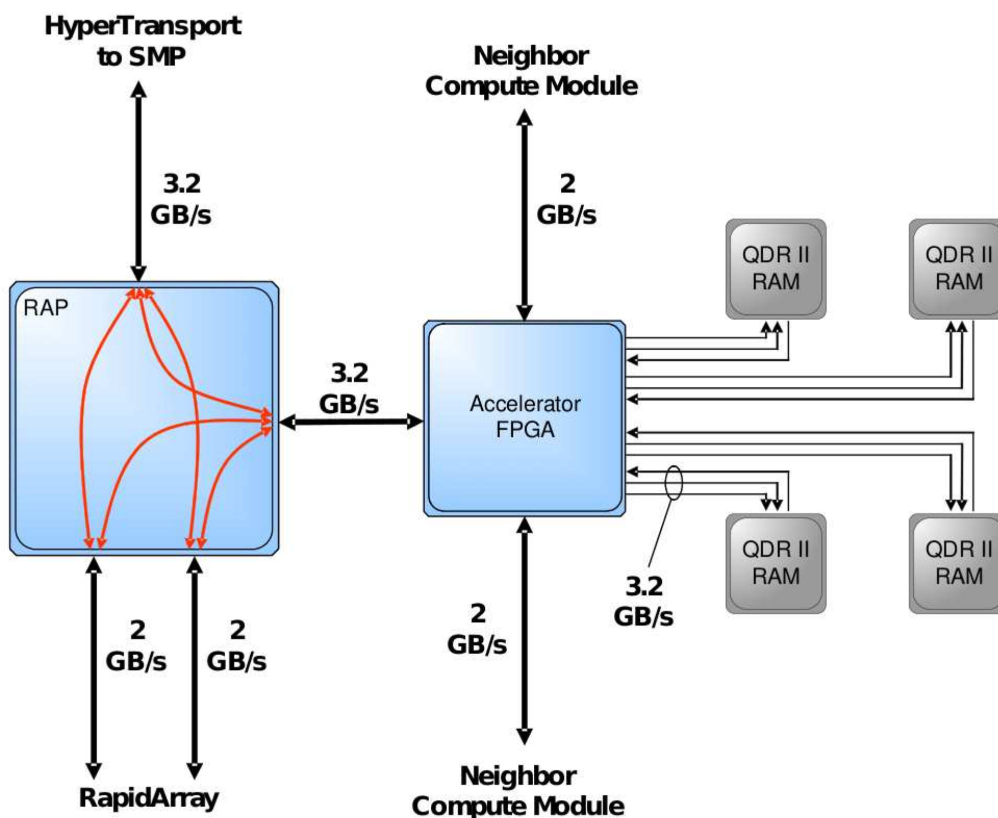

Figura 4.2: Conexões do FPGA. Fonte: (CRAY, 2005a)

4.1.1 Interconexão *RapidArray*

A interconexão RapidArray é o barramento utilizado para conectar processadores RapidArray dentro de um chassis e entre chassis e, conseqüentemente, processadores, memória e FPGAs. É um barramento de banda larga, baixa latência, não bloqueante que pode chegar a até 96 GB/s dentro de um chassis (CRAY, 2005a).

4.1.2 Recursos de memória

No XD1 o FPGA tem acesso a quatro tipos de memória. As mais rápidas são a RAM distribuída e a RAM de bloco (Block RAM), que residem dentro do Virtex II Pro, e possuem 18Kbits e 16 KBytes de tamanho respectivamente.

O FPGA possui acesso a quatro módulos de memória *Quad Data Rate* de segunda geração (QDR II) SRAM de 64 MBytes de tamanho cada. Também pode acessar a memória

principal do sistema que possui vários Gigabytes de espaço.

4.2 Programando o Cray XD1

Para a criação de programas que utilizem o FPGA, a Cray disponibiliza uma API para comunicação com o FPGA. Essa API permite realizar operações administrativas e transferência de dados. Também é disponibilizada uma ferramenta, o FPGA Control Utility, que auxilia nas operações administrativas.

4.2.1 FPGA Control Utility

Essa ferramenta é utilizada para tarefas como carregar e descarregar o binário, mas tem como principal função a conversão do binário gerado pelo *software* da Xilinx para o formato da Cray.

Para isso, a ferramenta utiliza informações como o *Part Number* da Cray e a freqüência de sinais de relógio desejada para converter para o novo formato.

4.2.2 Operações administrativas

As operações administrativas disponíveis na API englobam as operações necessárias para configurar o ambiente. A seguir, serão apresentadas as principais operações:

- fpga_open: cria um descritor de arquivo para o dispositivo, esse descritor será utilizado em todas as outras operações da API.
- fpga_load: carrega o binário, já convertido para o formato da Cray, para o FPGA. É equivalente a carregar o binário utilizando o FPGA Control Utility.
- fpga_reset: reseta o FPGA, ativando o sinal de *reset* presente no módulo *RapidArray Transport Core*.
- fpga_start: desativa o sinal de *reset*, fazendo com que a lógica comece a executar.
- fpga_status: retorna o estado do FPGA, que é o valor de um registrador do *RapidArray Transport Core*.
- fpga_is_loaded: retorna verdadeiro caso o FPGA possua um binário carregado.
- fpga_unload: apaga o binário carregado no FPGA.
- fpga_close: fecha o arquivo do dispositivo.

4.2.3 Transferência de Dados

Para a transferência de dados entre os Opteron e o FPGA exitem três opções: mapear a memória interna do FPGA no espaço de memória da aplicação, acessar uma diretamente uma posição única da memória do FPGA ou reservar um espaço na memória da aplicação para o FPGA acessar diretamente (CRAY, 2005c).

Para o mapeamento da memória interna do FPGA, primeiramente faz-se uma chamada de mapeamento que retorna um ponteiro para aquela região. Leituras e gravações nessa região são feitas através desse ponteiro, como se fosse um vetor. A função da API para esse tipo de transferência é:

- fpga_memmap: mapeia o número de bytes passado como parâmetro da memória interna do FPGA para o espaço de memória da aplicação, retornando um ponteiro para o primeiro byte dessa região.

O acesso direto à uma única posição da memória do FPGA é feito através de duas funções da API, uma de escrita e uma de leitura:

- fpga_wrt_appif_val: escreve um valor em um endereço, ambos passados como parâmetro, na memória interna do FPGA.

- fpga_rd_appif_val: lê o conteúdo de um endereço, passado como parâmetro, da memória interna do FPGA.

Para reservar um espaço da memória da aplicação para ser acessada pelo FPGA, a API disponibiliza um método que aloca essa região de memória para esse fim. Após essa região ser alocada, é necessário que o programa comunique o FPGA o endereço base da região, o que pode ser feito utilizando qualquer uma das abordagens descritas anteriormente. As funções da API destinadas à esse fim são:

- fpga_register_ftrmem: reserva um espaço da memória da aplicação para ser acessada pelo FPGA, do tamanho passado como parâmetro.

- fpga_dereg_ftrmem: libera a memória previamente alocada.

4.3 Configurando o FPGA do XD1

A configuração do FPGA com a lógica do usuário se dá pela inserção dessa lógica em um ambiente já estabelecido pela Cray. O ambiente consiste de um *RapidArray Transport Core*, de um *QDR II Core* e um gerador de ciclos de relógio. Esse ambiente pode ser visualizado na figura 4.3.

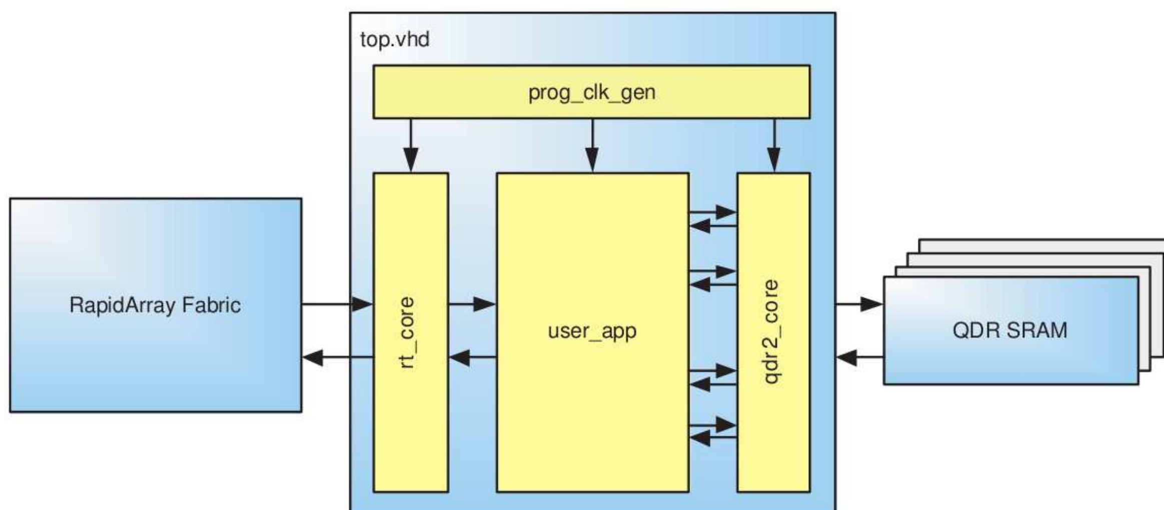

Figura 4.3: Topo de um projeto HDL para o Cray XD1. Fonte: (CRAY, 2005a)

A lógica do usuário deve ser adaptada para se comunicar com o módulo *RapidArray Transport Core* (figura 4.4), que é o responsável pelo tratamento das requisições vindas do processador e por mandar requisições para o processador. O *RapidArray Transport Core* possui quatro grupos de sinais, para envio e recebimento de requisições e para envio e recebimento de respostas à essas requisições.

O módulo *QDR II Core* é opcional, pois a lógica do usuário pode usar somente a memória interna. Possui quatro grupos de sinais, para leitura e escrita em cada uma dos quatro módulos de memória QDRII SRAM (CRAY, 2005d).

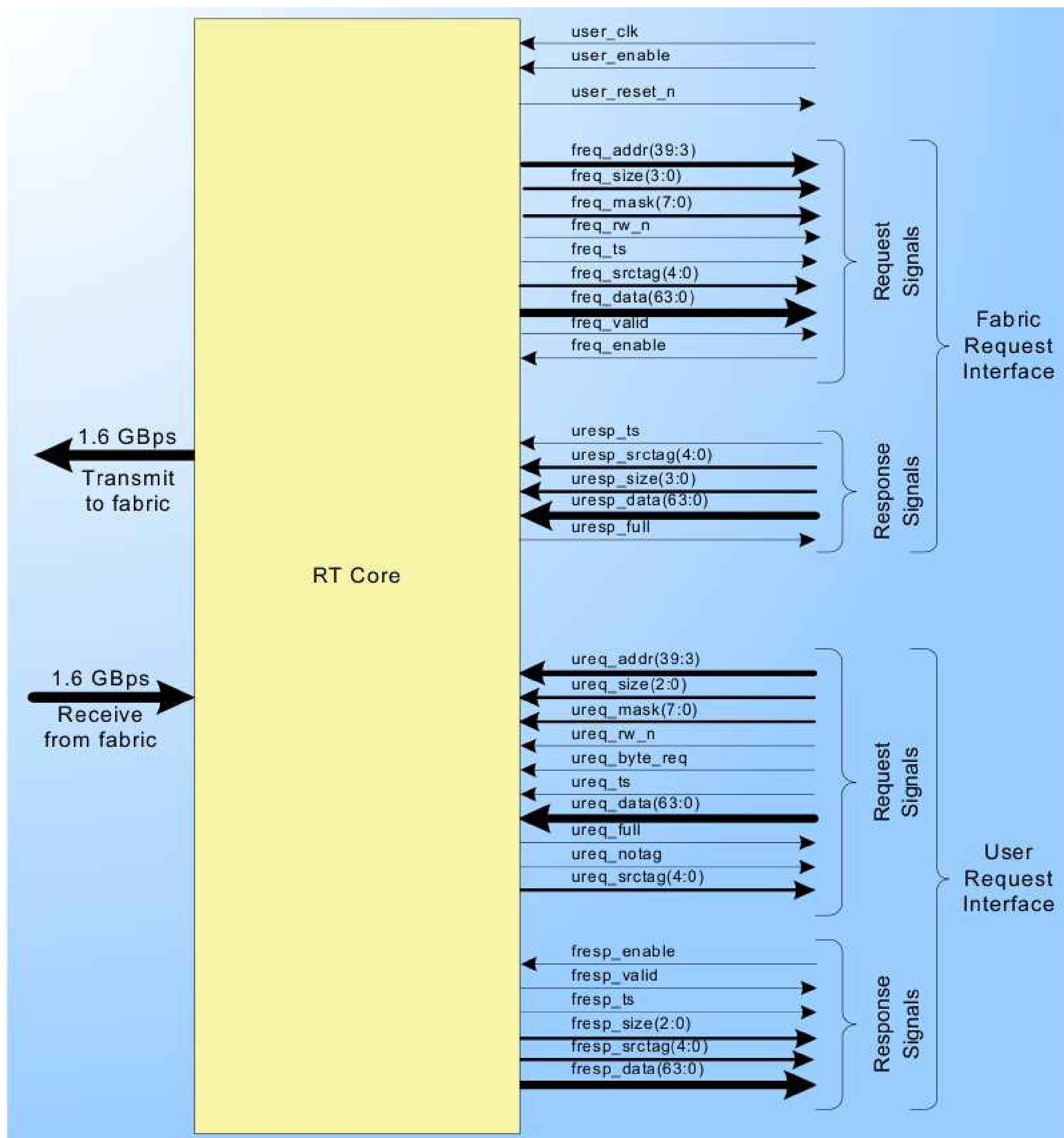

Figura 4.4: *RapidArray Transport Core*. Fonte: (CRAY, 2005b)

5 IMPLEMENTAÇÃO

Este capítulo descreve a implementação do trabalho, apresentando a implementação da FFT em *hardware* e sua adaptação para utilizar os módulos da Cray. Por fim, apresenta-se uma avaliação sobre o resultado.

5.1 Introdução

Para a implementação da FFT em VHDL, foi escolhido o algoritmo *radix-2* por decimação no tempo, por ser um algoritmo simples e, por ser menos otimizado, pode ressaltar o quão valiosa é a utilização de uma implementação híbrida para o cálculo da FFT.

O estudo da FFT mostrou que a parte central do algoritmo é a operação de borboleta, sendo que o resto do algoritmo se limita a calcular os parâmetros dessa operação. Sendo assim, iniciou-se a implementação por essa operação.

Todos os módulos a seguir foram implementados utilizando a biblioteca *math_real* da IEEE para operações com números reais. Essa biblioteca foi utilizada para acelerar o desenvolvimento da FFT e partir rapidamente para o acoplamento com os módulos da Cray e da Xilinx. Como essa biblioteca não é sintetizável para FPGA, é necessário que essas operações de números reais sejam implementadas. Por não haver tempo hábil, optou-se pelo acoplamento por ser algo menos explorado.

5.2 Implementação da operação borboleta em VHDL

Para o cálculo da borboleta do algoritmo *radix-2* são necessários como entrada os dois números complexos, o número da etapa de cálculo e o grupo de borboletas da etapa atual. Como saída, essa borboleta tem dois números complexos.

O número da etapa e o grupo de borboletas da etapa atual são utilizados para o cálculo do fator *twiddle* e esse cálculo muitas vezes serve para mais de uma borboleta. Por isso,

foi decidido, por questão de otimização, pela separação do cálculo do fator da borboleta. Assim, a borboleta passa a receber um número complexo (o resultado do cálculo do fator) como parâmetro ao invés do número da etapa e do grupo de borboletas.

Para a construção da borboleta em *hardware*, foi desenvolvido um registrador de números complexos. Esse registrador possui como entrada dois números de ponto flutuante, um sinal de carga e um sinal de relógio e tem como saída dois números de ponto flutuante. As entradas de ponto flutuante são armazenadas na borda de subida do relógio desde que o sinal de carga esteja ligado.

Esse registrador de números complexos será utilizado para armazenar os valores de saída da borboleta e o resultado do cálculo do fator *twiddle*.

Para o cálculo do fator, foram criados dois módulos que recebem o número da etapa e o grupo de borboletas, onde um calcula o seno e o outro o cosseno baseado nessas entradas. As operações de seno e cosseno foram calculadas utilizando as respectivas funções da biblioteca da IEEE. Essas operações poderiam ser facilmente implementadas utilizando consulta a uma tabela de resultados pois as funções de seno e cosseno são aplicadas a um número pequeno de valores. Depois do cálculo, esses valores são armazenados no registrador destinado a esse fim.

Para o cálculo da borboleta propriamente dita, foi criado um módulo que lê as entradas e o registrador do fator *twiddle* e grava o resultado nos registradores de saída. As portas de entrada e saída do módulo *twiddle* e do núcleo da borboleta podem ser vistos nas figuras 5.1 e 5.2 respectivamente.

```
entity twiddle is
    port(clock  : in std_logic; -- relogio
         grupo  : in real;      -- grupo de borboletas
         passo  : in real;      -- passo atual
         sr     : out real;     -- saida real
         si     : out real      -- saida imaginaria
    );
end twiddle;
```

Figura 5.1: Portas do módulo que calcula o fator *twiddle*

Com essa arquitetura, mostrada na figura 5.3, a borboleta implementada fica modularizada e completamente independente da plataforma. Nos testes, esse modelo mostrou que funciona e consegue calcular a borboleta em 2 ciclos de relógio, mas pode utilizar

```
entity nucleo_borboleta is
    port(ar      : in real;    -- entrada A real
         ai      : in real;    -- entrada A imaginaria
         br      : in real;    -- entrada B real
         bi      : in real;    -- entrada B imaginaria
         wr      : in real;    -- parte real do W
         wi      : in real;    -- parte imaginaria do W
         r1r     : out real;   -- saida 1 real
         r1i     : out real;   -- saida 1 imaginaria
         r2r     : out real;   -- saida 2 real
         r2i     : out real    -- saida 2 imaginaria
    );
end entity nucleo_borboleta;
```

Figura 5.2: Portas do módulo calcula a borboleta

pipeline para calcular o fator *twiddle* da próxima etapa enquanto calcula a borboleta da etapa atual. Dessa forma, obtém-se uma vazão de um ciclo de relógio.

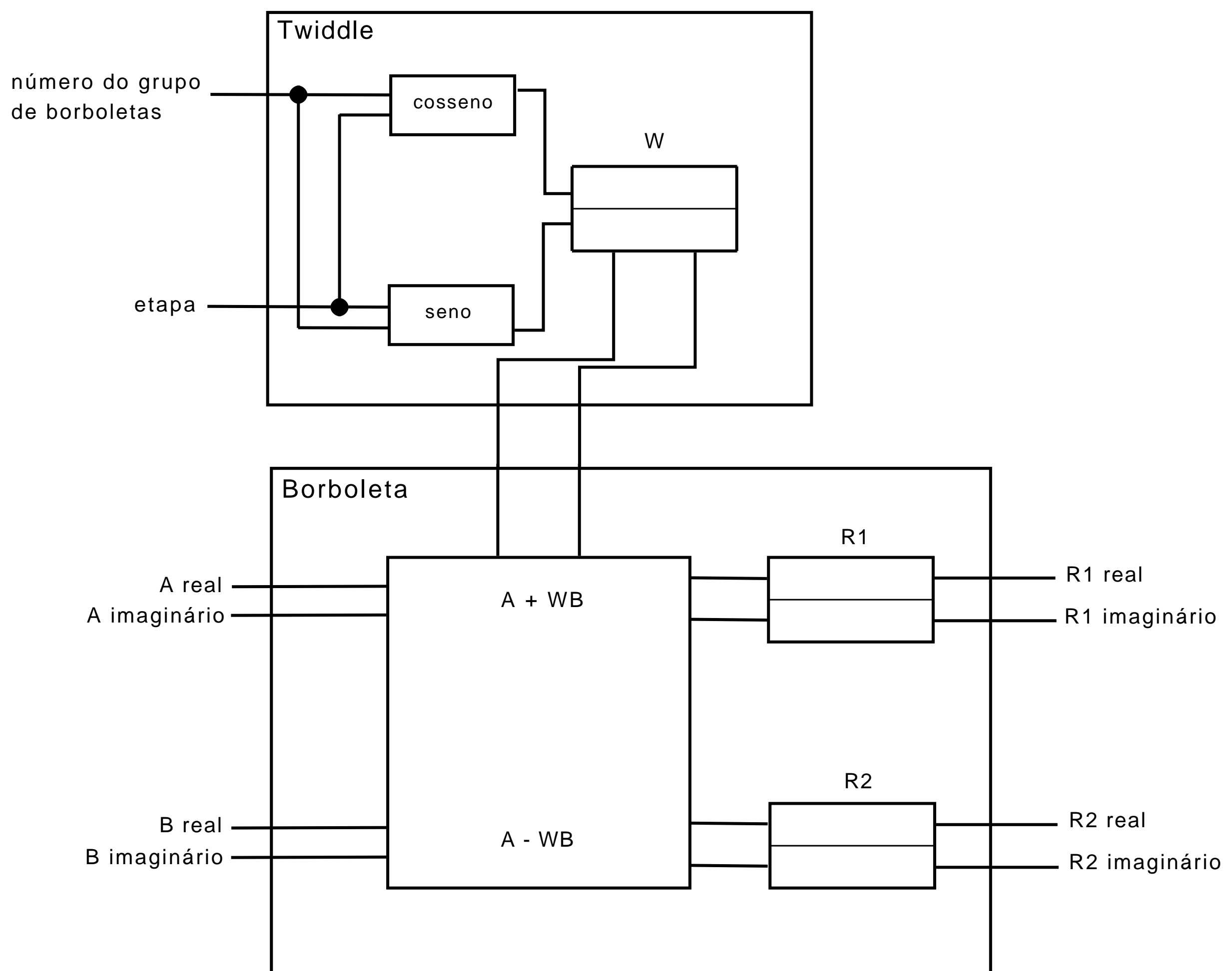

Figura 5.3: Arquitetura da borboleta implementada

5.3 Implementação da FFT em VHDL

Para o cálculo da FFT, foi criado um novo módulo que conecta o módulo da borboleta com o módulo do fator *twiddle* e utiliza-se um laço que implementa o algoritmo *radix-2*, semelhante ao código em C que realiza essa mesma função. Nessa implementação considera-se que o vetor de entrada já está ordenado por bit-invertido e não foi levado em consideração a interface entre a FFT em VHDL e o programa que irá executá-la, comportando-se como se os dados já tivessem sido copiados para a memória. Essa arquitetura pode ser vista na figura 5.4 e o algoritmo implementado na figura 5.5

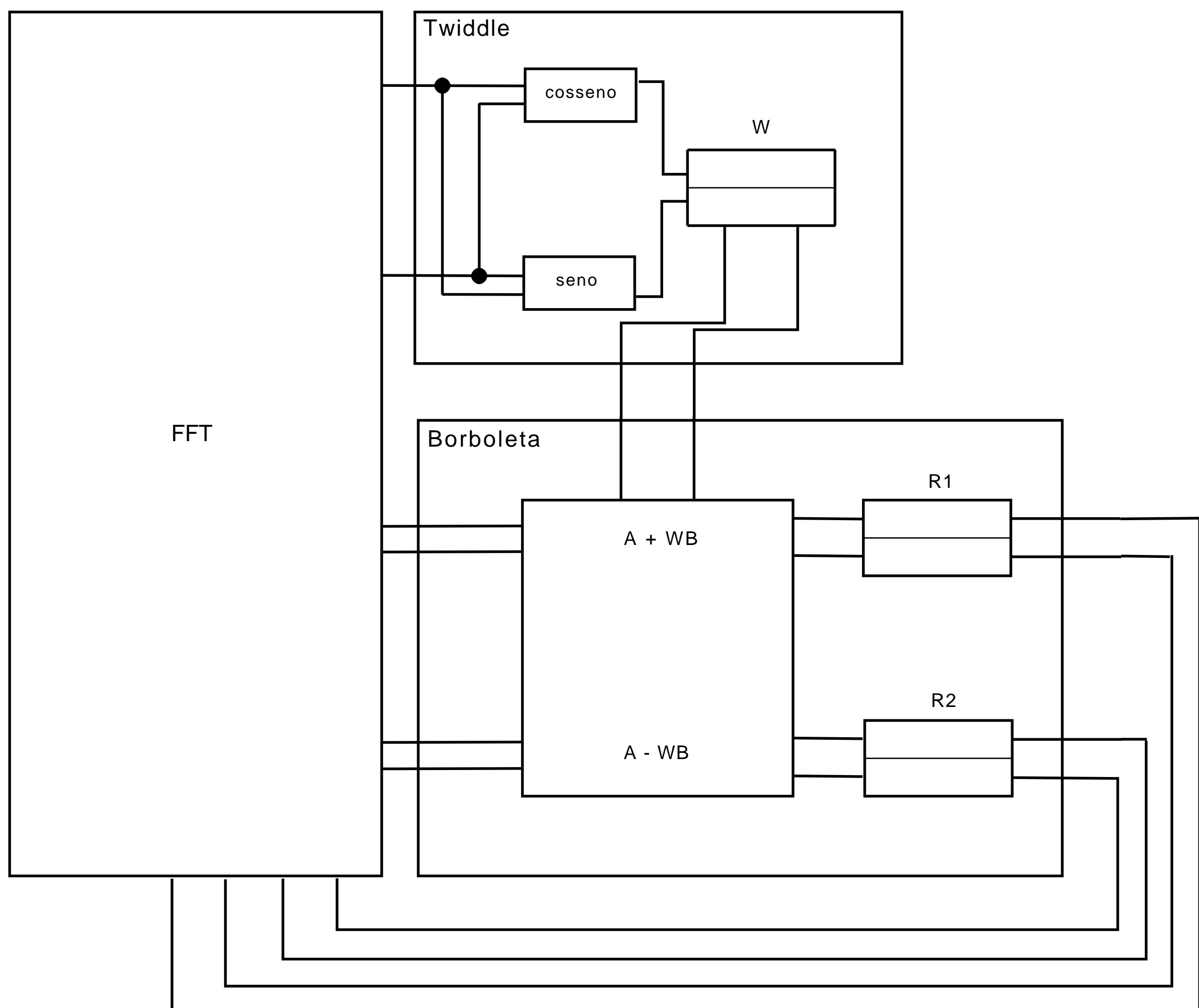

Figura 5.4: Arquitetura da FFT implementada

Essa implementação foi simulada utilizando-se vetores de entrada com quatro e oito números complexos. Os dois vetores precisaram, para serem totalmente computados, de um número de ciclos de relógio igual ao número de operações realizadas (soma do número de borboletas com o número de cálculos de fator *twiddle*), sendo 7 (4 + 3) para o vetor

```
wait until clock'event and clock = '0';
passo := 1;
while passo < N loop  -- percorre os logN passos
    grupo_borb := 0;
    while grupo_borb < passo loop
        grupo <= real(grupo_borb); -- envia para o twiddle
        etapa <= real(passo);      -- envia para o twiddle
        wait until clock'event;    -- espera um
        wait until clock'event;    -- ciclo de relogio
        i := grupo_borb;
        while i < N loop  -- percorre borboletas do grupo
            ar <= xreal(i);            --
            ai <= ximag(i);            -- envia os valores
            br <= xreal(i + passo);    -- para a borboleta
            bi <= ximag(i + passo);    --
            wait until clock'event;    -- espera um
            wait until clock'event;    -- ciclo de relogio
            xreal(i) := r1r;           --
            ximag(i) := r1i;           -- recebe os valores
            xreal(i + passo) := r2r;   -- da borboleta
            ximag(i + passo) := r2i;   --
            i := 2 * passo + i;
        end loop;
        grupo_borb := grupo_borb + 1;
    end loop;
    passo := passo * 2;
end loop;
```

Figura 5.5: Algoritmo FFT implementado em VHDL

de tamanho quatro e 19 (12 + 7) para o de tamanho oito. Genericamente, precisa-se de $(N/2)log_2N + \sum_{k=0}^{(log_2N)-1} 2^k$ ciclos de relógio para concluir a tarefa.

A corretude da implementação pode ser visualizada comparando os resultado obtidos com a simulação (figura 5.6) e com a execução de um programa escrito em linguagem C (figura 5.7) para o mesmo vetor de entrada.

27	-4.12132	4	0.12132	5	0.12132	4	-4.12132
0	3.29289	1	-4.70711	0	4.70711	-1	-3.29289

Figura 5.6: Resultado obtido em simulação

Real	Imaginario
27.000000	0.000000
-4.121320	3.292893
4.000000	1.000000
0.121320	-4.707107
5.000000	0.000000
0.121320	4.707107
4.000000	-1.000000
-4.121320	-3.292893

Figura 5.7: Resultado obtido em programa de alto nível

5.4 Conectando a FFT nos módulos da Cray

Para poder utilizar a FFT em um FPGA real é necessário conectá-la às entradas e saídas do *chip* e estabelecer um protocolo de comunicação com dispositivos, como processador e memória. Para ajudar nessas questões, a Cray disponibiliza algumas Propriedades Intelectuais (*Intellectual Properties* - IP), que são módulos pré-compilados em um formato não-visualizável.

O principal desses IPs é o *RapidArray Transport Core*, responsável por conectar os sinais da lógica do usuário e estabelecer um protocolo de comunicação com a malha *RapidArray Transport* e, conseqüentemente, ao processador e memória (CRAY, 2005b).

Dos sinais que esse módulo repassa à lógica do usuário, são utilizados somente os que correspondem às requisições do processador. Esses sinais são repassados para o módulo *RapidArray Transport Client* disponibilizado pela Cray para fazer a interface entre o *RapidArray Transport Core* e a lógica do usuário. Esse módulo possui diversos sinais (figura 5.8) mas somente dois foram usados, para entrada e saída com a Block RAM. Esses sinais são do tipo *record* e suas estruturas e seus subtipos são mostradas na figura 5.9.

Para gravar, por exemplo, o valor '42' na posição 0 da Block RAM o sinal *t_rt_req* é preenchido como mostrado na figura 5.10.

Utilizando esses dois sinais tanto o processador quanto o FPGA (mais precisamente os módulos *RapidArray Transport Client* e FFT) se comunicam com a Block RAM e com os registradores internos. À exemplo do *RapidArray Transport Client*, o módulo FFT recebe um *t_rt_responses* da Block RAM e envia *t_rt_req*, como pode ser visto na figura 5.11.

Essa abordagem foi escolhida pois o FPGA não precisa iniciar uma transação, a implementação fica simplificada se o FPGA apenas responder à requisições do processador.

```
entity rt_client is
    port (
        -- Global signals
        reset_n      : in  std_logic;
        user_clk     : in  std_logic;
        user_enable  : in  std_logic;
        rt_ready     : in  std_logic;
        -- RT Interface
        -- Fabric Request Interface
        ...
        -- User Response Interface
        ...
        -- Block RAM Interface
        rt_responses : in  t_rt_responses;
        rt_req       : out t_rt_req
    );
end entity rt_client;
```

Figura 5.8: Portas utilizadas do módulo *RapidArray Transport Client*

Essas requisições são todas feitas à memória interna e é através dela que se dá a comunicação entre o *RapidArray Transport Client* e a FFT.

Dessa forma, a aplicação que utiliza essa implementação primeiramente mapeia a memória interna do FPGA para a transferência dos dados e, após o fim da transferência, grava o valor '1' em um registrador de configuração interno do FPGA. Isso sinaliza para a lógica do usuário que todos os dados já estão na memória interna e que o módulo da FFT já pode começar a executar.

Após o término do cálculo da FFT, o módulo grava no mesmo registrador o valor '0', o que sinaliza a aplicação que o cálculo já terminou e que a aplicação já pode utilizar os dados.

A memória interna é instanciada dentro da lógica do usuário utilizando o IP Block RAM da Xilinx. Esse IP é gratuito, mas só pode ser usado em FPGAs desse fabricante. Para acessar a Block RAM, utiliza-se outro módulo que funciona como interface entre a lógica do usuário e a Block RAM.

Como tanto o *RapidArray Transport Core* quanto o módulo FFT necessitam acessar a memória interna, foi necessário colocar um multiplexador na sua entrada e um demultiplexador em sua saída, ambos controlados pelo módulo FFT. O mesmo foi feito para os registadores internos. Para acesso a esses registradores também é usado um IP da Xi-

```
type t_rt_req is record
    rd    : t_rt_sel;
    wr    : t_rt_sel;
    addr  : t_rt_block_addr;
    wdata : t_byte_array(7 downto 0);
    mask  : std_logic_vector(7 downto 0);
end record t_rt_req;

type t_rt_sel is array (t_rt_blocks) of std_logic;

subtype t_rt_block_addr is
    std_logic_vector(t_rt_block_decode'low-1 downto 3);

subtype t_rt_block_decode is
    std_logic_vector(26 downto 23);

type t_byte_array is
    array(natural range <>) of
        std_logic_vector(7 downto 0);

type t_rt_responses is array (t_rt_blocks) of t_rt_resp;

type t_rt_resp is record
    rdata : t_byte_array(7 downto 0);
    busy  : std_logic;
end record t_rt_resp;

type t_rt_blocks is (regs, bram, qdr2);
```

Figura 5.9: Estrutura dos *records*

linx que os implementa e um módulo de interface. Foi necessário também colocar um multiplexador na entrada dos registradores, mas a saída é ligada diretamente ao módulo *RapidArray Transport Core*, pois a FFT não lê valores dos registradores.

A arquitetura final da implementação pode ser vista na figura 5.12. Essa arquitetura se conecta com o módulo *RapidArray Transport Core*, mostrado na figura 4.4, e sua entrada e saída correspondem aos sinais *Fabric Request Interface* explicitados nessa figura.

Essa solução começou a ser testada no FPGA do Cray XD1 do INPE, mas por problemas técnicos ocorridos durante os testes, estes tiveram que ser cancelados e, portanto, a avaliação se dará somente por simulação.

```
valor := "00000000000000000000000000000000
          00000000000000000000000000101010"; -- 42

bytes(7) := valor(63 downto 56);
bytes(6) := valor(55 downto 48);
bytes(5) := valor(47 downto 40);
bytes(4) := valor(39 downto 32);
bytes(3) := valor(31 downto 24);
bytes(2) := valor(23 downto 16);
bytes(1) := valor(15 downto 8);
bytes(0) := valor(7 downto 0);

rt_req.wr(bram) <= '1';     -- bram == Block RAM
rt_req.rd(bram) <= '0';
rt_req.addr  <= "00000000000000000000"; -- 0
rt_req.mask  <= "11111111"; -- quais bytes serao escritos
rt_req.wdata <= bytes;
```

Figura 5.10: Exemplo de requisição via *hardware*

```
entity fft is
    port(user_clk  : in std_logic;
         reset_n   : in std_logic;
         start     : in std_logic;   -- sinal para comecar
         bram_resp : in t_rt_responses;
         to_reg    : out std_logic;  -- para registradores
         fft_req   : out t_rt_req
    );
end entity fft;
```

Figura 5.11: Portas do módulo FFT

5.5 Avaliação

Essa avaliação foi feita através de simulação utilizando a ferramenta ISE Foundation da Xilinx. Como essa implementação não foi utilizada em um FPGA real, sua avaliação se dá em número de ciclos de relógio necessários para terminar a tarefa. Esses números de ciclos de relógio foram obtidos através de simulação e poderão mudar em uma implementação sintetizável.

Inicialmente, é necessário um ciclo para inicializar e 2 ciclos para começar a gravar na Block RAM, devido ao multiplexador na entrada e à interface da Block RAM, e N ciclos para gravar N números reais de 64 bits.

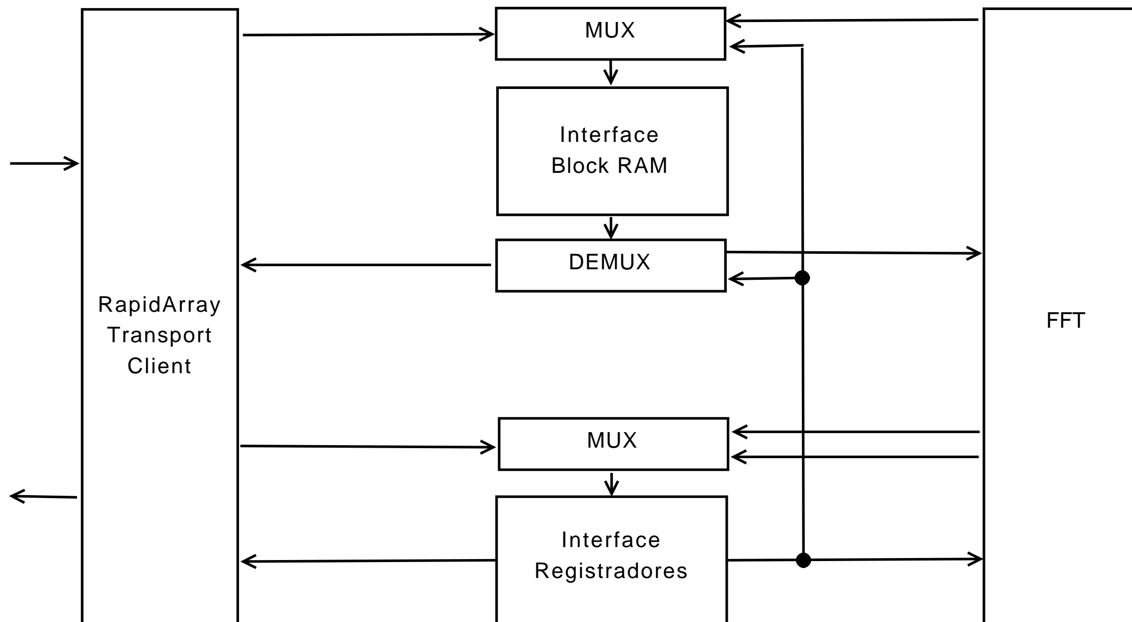

Figura 5.12: Arquitetura implementada

Após o termino da gravação dos dados na memória interna, são necessários 3 ciclos para gravar no registrador e para o módulo FFT ler do registrador o sinal para começar a executar.

Como mencionado anteriormente, o módulo FFT necessita de $(N/2)log_2N+\sum_{k=0}^{(log_2N)-1} 2^k$ ciclos de relógio para calcular a FFT de N números complexos. Como a aplicação não precisa copiar os dados de volta para a memória principal do sistema, o resultado é obtido com $(N/2)log_2N + \sum_{k=0}^{(log_2N)-1} 2^k + 2N + 6$ ciclos de relógio.

Considerando que a memória interna do Virtex II Pro tem 16 Kbytes de espaço, somente 2048 números reais, ou 1024 números complexos, podem ser armazenados simultaneamente. Caso N ultrapasse 1024, é necessário dividir o vetor e aplicar a FFT separadamente.

Para isso, é necessário copiar os dados já calculados para a memória principal do sistema. Isso necessita de 3 ciclos de relógio para o módulo FFT sinalizar que terminou de calcular, 3 ciclos para começar a ler os dados da Block RAM e 2048 ciclos para ler os 1024 números complexos. Note que esse número é fixo para cada troca de contexto, pois caso N seja maior que 1024 ele será um múltiplo de 1024 deixando a memória sempre cheia.

Caso N seja 2048, por exemplo, a FFT precisará ser aplicada 4 vezes, o que necessita de 3 cópias para a memória principal, totalizando uma penalidade de 6162 ciclos de re-

Tabela 5.1: Ciclos de relógio necessários para calcular a FFT

Tamanho do vetor	Ciclos necessários
128	837
256	1797
512	3845
1024	8197
2048	38950
4096	120958

lógio. A tabela 5.5 mostra o número de ciclos de relógio necessários para calcular a FFT para alguns tamanhos do vetor de entrada.

Considerando que a freqüência máxima do Virtex II Pro é de 200 MHz, ou seja, 200 milhões de ciclos por segundo, nota-se que, mesmo com as cópias para a memória principal, o número de ciclos se mantém bem abaixo desse valor.

6 CONCLUSÃO

Este trabalho apresentou a implementação parcial de uma transformada rápida de Fourier em *hardware* para ser executada em um FPGA da Xilinx, contido no supercomputador XD1 da Cray.

Primeiramente, foi estudada a transformada rápida de Fourier, para entender seu funcionamento e suas possíveis dificuldades de implementação. Em seguida, foram estudadas diversas ferramentas para desenvolvimento em *hardware* para escolher a melhor maneira de implementar a FFT.

Com a FFT em VHDL implementada, foram estudados os módulos disponibilizados pela Cray e pela Xilinx para fazer o acoplamento do módulo FFT com o resto do sistema. Assim foi possível fazer uma simulação em *software* usando uma implementação muito parecida com a implementação final em *hardware*.

O estágio atual de implementação está próximo de executar em um FPGA real e pode ser complementado futuramente para se tentar acelerar aplicações que utilizem FFT. Para isso, é necessário a implementação das operações com números reais (incluindo seno e cosseno) para substituir a biblioteca não sintetizável da IEEE e alguns testes para verificar se a comunicação simulada com o processador condiz com a real comunicação com o processador.

A avaliação se deu através de simulação e mostrou que a implementação é válida e consegue calcular a FFT em um número de ciclos de relógio razoavelmente pequeno. Essa implementação deixa espaço para melhorias de desempenho, principalmente na aplicação da FFT, aplicando várias operações de borboleta em paralelo.

Como trabalho futuro, pode-se terminar o acoplamento da implementação com um FPGA real e medir o possível ganho de desempenho que pode ser obtido com essa abordagem.

REFERÊNCIAS

BLUESTEIN, L. I. A linear filtering approach to the computation of the discrete Fourier transform. **Northeast Electronics Research and Engineering Meeting Record 10**, [S.l.], p.218–219, 1968.

BRUUN, G. z-Transform DFT filters and FFTs. **IEEE Trans. on Acoustics, Speech and Signal Processing (ASSP) 26**, [S.l.], p.56–63, 1978.

BURDEN, R. L.; FAIRES, J. D. **Numerical Analysis**. [S.l.]: Cole Publishing Company, 1997.

COMPTON, K.; HAUCK, S. Reconfigurable Computing: a survey of systems and software. **ACM Computing Surveys**, [S.l.], v.34, n.2, p.171–210, 2002.

COOLEY, J. W.; TUKEY, J. W. An Algorithm for the Machine Calculation of Complex Fourier Series. **Mathematics of Computation**, [S.l.], v.19, n.90, p.297–301, 1965.

CRAY, I. **Cray XD1 FPGA Development**. [S.l.]: Cray, Inc., Seattle, WA, 2005.

CRAY, I. **Design of Cray XD1 RapidArray Transport Core**. [S.l.]: Cray, Inc., Seattle, WA, 2005.

CRAY, I. **Cray XD1 Programming**. [S.l.]: Cray, Inc., Seattle, WA, 2005.

CRAY, I. **Design of Cray XD1 QDR II SRAM Core**. [S.l.]: Cray, Inc., Seattle, WA, 2005.

ESTRIN, G.; BUSSEL, B.; TURN, R.; BIBB, J. Parallel Processing in a Restructurable Computer System. **IEEE Transactions on Electronic Computers**, [S.l.], v.12, n.5, p.747–755, 1963.

GOOD, I. J. The interaction algorithm and practical Fourier analysis. **J. R. Statist. Soc. B 20**, [S.l.], p.361–372, 1958.

KEAN, T. **Algotronix History**. [S.l.]: Algotronix, 2007.

LASS, S. ESL Tools Make FPGAs Nearly Invisible to Designers. **Xcell Journal, Third Quarter 2006**, [S.l.], p.6–8, 2006.

LEVINE, B. **A Systematic Implementation of Image Processing Algorithms on Configurable Computing Hardware**. 1999. Dissertação (Mestrado) — University of Tennessee, Knoxville.

LYNCH, P.; Ireland Meteorological Service. **Dynamo: a one-dimensional primitive equation model**. [S.l.]: Ireland Meteorological Office, 1984.

MARRA, F.; COELHO, V.; MOURELLE, L.; NEDJAH, N. Tradutor de C para VHDL–C2VHDL. **Revista Eletrônica de Iniciação Científica, Ano I, Vol. I, No. I**, [S.l.], 2001.

MAXFIELD, C. M. **The Design Warrior's Guide to FPGAs**. [S.l.]: Elsevier, 2004.

PENTINMAKI, I.; MENON, R.; BASS, J. **FPGA C Compiler**. [S.l.]: Sourceforge, 2006.

PETERSEN, R. J. **An Assessment of the Suitability of Reconfigurable Systems for Digital Signal Processing**. 1995. Tese (Doutorado) — Brigham Young University. Dept. of Electrical and Computer Engineering.

PETERSON, G.; SMITH, M. Programming High Performance Reconfigurable Computers. **SSGRR 2001**, [S.l.], 2001.

RADER, C. M. Discrete Fourier transforms when the number of data samples is prime. **Proc. IEEE 56**, [S.l.], p.1107–1108, 1968.

Symphony EDA. **VHDL Simili - Version 3.1**. [S.l.]: Symphony EDA, 2007.

TODMAN, T.; CONSTANTINIDES, G.; WILTON, S.; MENCER, O.; LUK, W.; CHEUNG, P. Reconfigurable Computing: architectures and design methods. **IEE Proceedings: Computer & Digital Techniques, Vol. 152, No. 2, March 2005**, [S.l.], p.193–208, 2005.

VELHO, H. F. C.; CLAEYSSEN, J. C. R. Singular Value Decomposition in the Numerical Integration of an Atmospheric Model. **XIII Congresso Íbero-Latino-americano de Métodos Computacionais para a Engenharia**, [S.l.], 1992.

YAVNE, R. An economical method for calculating the discrete Fourier transform. **Proc. AFIPS Fall Joint Computer Conf. 33**, [S.l.], p.115–125, 1968.