Universidade de São Paulo
Instituto de Física de São Carlos
Departamento de Física e Ciência dos Materiais

**PROJETO E CONSTRUÇÃO DE
SISTEMA DE AQUISIÇÃO E
PROCESSAMENTO DE DADOS PARA
ESPECTROSCOPIA POR
TRANSFORMADA RÁPIDA DE FOURIER**

Walter Soto Encinas Júnior

Dissertação apresentada ao Instituto
de Física de São Carlos - USP, para
obtenção do título de Mestre em
Ciências - Física Aplicada

ORIENTADOR: *Prof. Dr. Máximo Siu Li*

SÃO CARLOS

1995

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE WALTER SOTO ENCINAS JUNIOR APRESENTADA
AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 03/04/1995

Av. Dr. Carlos Botelho, 1465
Caixa Postal 369
CEP 13560-970 - São Carlos - SP
Brasil

COMISSÃO JULGADORA:

Prof. Dr. Máximo Siu Li

Prof. Dr. Luciano da Fontoura Costa

Profa. Dra. Maria Steia Veludo de Paiva

Dedico este trabalho à minha família, e em especial, à
minha mãe, que me apoiou em todos os momentos.

Agradecimentos

Ao Máximo, pela orientação, paciência e compreensão.

Ao Luciano, que indicou um caminho para este trabalho quando as coisas estavam nebulosas.

Ao Cacau, colega de jornada, que tanto trabalhou para o sucesso deste trabalho.

Ao pessoal do Laboratório de Instrumentação Eletrônica - Silvio, Rui, Ricardo e Marcelo - amigos que me ajudaram de tantas maneiras que fica difícil agradecer só com palavras.

Ao Paulinho, que sempre acreditou e contribuiu para este trabalho.

Ao César, Fátima e Romão, pela amizade e palavras de incentivo.

Ao Bomba, pela ajuda e interesse constante.

Ao Pião, que sem a compreensão da importância deste trabalho, nada seria possível.

Ao Ari, Leandro, Lázinho, Paulo, Rejane, Sidney, Tony e Sueli, pela ajuda e convivência sempre agradável.

Ao Silvío TC, pelas sugestões, interesse e pelas horas de discussões fundamentais sobre este e outros assuntos.

Aos técnicos do Laboratório de Instrumentação Eletrônica, Lírio, Ivanilda, Marquinhos e Ailton, sempre atenciosos e interessados.

A Regina, que me orientou na estruturação da tese, fornecendo-me um belo modelo.

Às bibliotecárias, cuja eficiência e compreensão foram fundamentais.

Ao pessoal da seção de alunos, atencioso e compreensivo.

Sumário

Lista de Figuras	v
Lista de Tabelas.....	vii
Resumo	viii
Abstract.....	ix
Abreviaturas Utilizadas	x
Introdução.....	1
1. Interferometria.....	3
1.1. Introdução.....	3
1.2. Interferômetro de Michelson.....	3
1.3. Fundamentos da Espectroscopia por Transformada de Fourier.....	4
1.4. Espectrofotômetro de Varredura Refrativa	5
1.5. Vantagens da Espectroscopia por Transformada de Fourier	6
1.5.1. Vantagem de Fellgett	6
1.5.2. Vantagem de Jacquinot	6
1.6. Considerações Práticas	6
1.7. Requisitos de Desempenho	8
1.7.1. Módulo de Aquisição	8
1.7.2. Módulo de Processamento	8
2. Transformada Rápida de Fourier.....	10
2.1. Introdução.....	10
2.2. FFT segundo Gold e Bially.....	10
2.3. Outros algoritmos de FFT	12
2.4. FFT de Vetores Reais.....	13
2.5. Funções de Apodização.....	14
2.5.1. Função Retângulo	15
2.5.2. Função de Hanning	15
2.5.3. Função de Blackman	16
3. Processador Vetorial de Sinais.....	18
3.1. Introdução.....	18
3.2. Arquitetura.....	18
3.2.1. RAM de Dados	19
3.2.2. Unidade de Execução	20
3.2.3. Unidade de Interface com Barramento.....	21

3.2.4. Registradores	21
3.3. Modos de Operação	22
3.4. Interrupções	22
3.5. Instruções	23
3.5.1. Instruções de I/O	23
3.5.2. Instruções Aritméticas de Um Operando	24
3.5.3. Instruções Aritméticas de Dois Operandos	25
3.5.4. Instruções de Controle	25
4. Módulo de Aquisição	26
4.1. Introdução	26
4.2. Arquitetura	26
4.2.1. Barramento	27
4.2.2. Buffer de Barramento e Decodificação	28
4.2.3. Buffer de Memória	29
4.2.4. Buffer do Conversor AD	30
4.2.5. Conversor AD	30
4.2.6. Memória Local	30
4.2.7. Gerador de Endereços	31
4.2.8. Portas de Entrada e Saída (PIO)	31
4.2.9. Contadores (CTD)	31
4.2.10. Registro de Controle	32
4.2.11. Registro de Status	32
4.3. Modos de Operação	33
4.3.1. Aquisição Individual	33
4.3.2. Acesso a Memória Local	33
4.3.3. Aquisição Automática	33
4.3.4. Aquisição Automática Disparada por Evento	33
4.4. Lógica de Controle	33
4.4.1. Sinais de Saída	34
4.4.2. Sinais de Entrada	34
4.4.2.1. Multiplex	35
4.4.2.2. Lógica Combinacional	35
4.4.3. Modos de Operação	35
4.4.3.1. Modo 0	35
4.4.3.2. Modo 1	35
4.4.3.3. Modo 2	36
4.4.3.4. Modo 3	36
4.4.4. Variáveis de Estados	36

4.4.5. Diagrama de Estados.....	37
4.4.6. Implementação.....	38
4.4.6.1. Lógica Sequencial e Combinacional	38
4.4.6.2. Multiplex	38
4.5. Detalhes Operacionais	38
5. Módulo de Processamento	40
5.1. Introdução.....	40
5.2. Algoritmo de FFT	40
5.2.1. Introdução	40
5.2.2. Escolha do Algoritmo	41
5.2.2.1. Eficiência de Implementação	41
5.2.3. Implementação	43
5.2.3.1. Introdução	43
5.2.3.2. Mapeamento de memória do VSP	43
5.2.3.3. Apodização.....	44
5.2.3.4. FFT horizontal	45
5.2.3.5. Multiplicação, ajuste de escala e FFT vertical	46
5.2.3.6. Ajuste de escala final.....	46
5.2.3.7. Determinação dos vetores H e G - 1ª parte	47
5.2.3.8. Determinação dos vetores H e G - 2ª parte	48
5.2.3.9. Composição dos vetores H e G	48
5.2.3.10. Cálculo do módulo do vetor.....	49
5.3. Sistema de Desenvolvimento	49
5.3.1. Comunicação Hospedeiro-VSP	50
6. Resultados.....	51
6.1. Módulo de Aquisição	51
6.1.1. Precisão na Conversão	52
6.1.2. Análise da Taxa de Aquisição	52
6.2. Módulo de Processamento.....	52
6.2.1. Desempenho	52
6.2.2. FFT de um Pulso	53
6.2.3. Análise de Erros.....	53
7. Conclusões	55
Referências Bibliográficas	57
Apêndices.....	60
A. Listagem do Algoritmo de FFT.....	A1
B. Listagem do Programa de Montagem.....	B1

C. Rotinas de Controle para o MAq	C1
D. Rotinas de Controle para o MPr	D1
E. Esquemas do MAq.....	E1
F. Equações da PAL Combinacional.....	F1
G. Equações da PAL Seqüencial	E1
H. Esquema do Circuito de Disparo de Aquisição.....	H1
I. Programa de Teste do MAq	I1
J. Programa de Teste do MPr	J1

Lista de Figuras

Fig 1.1 - Interferômetro de Michelson.....	4
Fig 1.2 - Interferograma de fonte monocromática tomado a partir do deslocamento do espelho	5
Fig 1.3 - Interferômetro de Varredura Refrativa.....	6
Fig 1.4 - Interferograma de fonte monocromática tomado a partir do tempo	7
Fig 2.1 - Esquema de FFT segundo Gold e Bially.....	12
Fig 2.2 - Esquema de FFT segundo o Algoritmo de Fatores Primos	13
Fig 2.3 - Influência do truncamento temporal no domínio da frequência	14
Fig 2.4 - Espectro de potência da função retângulo	15
Fig 2.5 - Espectro de potência da função de Hanning.....	16
Fig 2.6 - Espectro de potência da função de Blackman.....	16
Fig 3.1 - Arquitetura do VSP	19
Fig 3.2 - Organização da RAM de Dados.....	19
Fig 3.3 - Arquitetura da Unidade de Execução	21
Fig 4.1 - Arquitetura do Módulo de Aquisição	27
Fig 4.2 - Sinais nos conectores do barramento ISA	28
Fig 4.3 - Utilização da memória RAM do IBM-PC	29
Fig 4.4 - Mapeamento da memória do MAq na memória do hospedeiro	31
Fig 4.5 - Registro de Controle	32
Fig 4.6 - Registro de Status	32
Fig 4.7 - Diagrama de blocos da máquina sequencial.....	34
Fig 4.8 - Algoritmo de aquisição dos dados.....	37
Fig 4.9 - Diagrama de estados da máquina sequencial do MAq.....	37
Fig 4.10 - Exemplo da utilização do bloco de multiplex.....	38
Fig 5.1 - Velocidade de processamento dos algoritmos de CT e de GB	42
Fig 5.2 - Apodização	45
Fig 5.3 - FFT horizontal.....	45
Fig 5.4 - Multiplicação, ajuste de escala e FFT vertical.....	46

Fig 5.5 - Ajuste de escala final	47
Fig 5.6 - Determinação dos vetores H e G - 1a parte.....	47
Fig 5.7 - Determinação dos vetores H e G - 2a parte.....	48
Fig 5.8 - Composição dos vetores H e G.....	49
Fig 5.9 - Cálculo do módulo do vetor	49
Fig. 6.1 - Resultado da aquisição de senóide de 20 Hz e 4 V de amplitude	51
Fig. 6.2 - Transformada de um pulso de 8 pontos de largura	53
Fig. 6.3 - Análise de erros do algoritmo de FFT	54

Lista de Tabelas

Tab. 4.1 - Sinais de controle	34
Tab. 4.2 - Sinais de saída do bloco Multiplex	35
Tab. 4.3 - Sinais de saída do bloco Combinacional	35
Tab. 4.4 - Estados associados aos modos de operação	36
Tab. 5.1 - Duração das instruções MLTC e FFT	42
Tab. 5.2 - Velocidade de processamento dos algoritmos de CT e de GB	42
Tab. 5.3 - Variáveis utilizadas pelo programa de FFT	44
Tab. 6.1 - Desempenho e tamanho dos módulos do programa de FFT	53

Resumo

No presente trabalho procurou-se desenvolver um sistema computacional para a automação de um Espectrofotômetro de Varredura Refrativa. Este sistema é baseado em um computador hospedeiro da linha IBM-PC, que pode ser de qualquer modelo, desde que possua barramento ISA de 16 bits.

O sistema se divide em dois módulos - aquisição e processamento. O módulo de aquisição é capaz de atingir taxas de aquisição de até 200 KHz, com resolução de 12 bits. É independente de controle do hospedeiro pois possui lógica de controle e sistema de armazenamento próprios.

O módulo de processamento envolve o uso de um processador vetorial de sinais digitais ZORAN ZR34161 dedicado a efetuar eficientemente a Transformada Rápida de Fourier (FFT). Também opera independentemente do computador hospedeiro. Este módulo é capaz de efetuar a FFT em um vetor real de 4096 pontos em 19,64 ms.

Abstract

This work was intended to develop a computational system to control a fast Fourier spectrophotometer. This system is based in an IBM-PC type host computer, with ISA bus 16 bits wide.

The system has two modules - acquisition and processing. The acquisition module works at 200 KHz sample rate, with 12 bits of resolution, and also works without host control, due to its own memory system and control logic.

The processing module uses a Zoran ZR34161 vector signal processor, dedicated to do the Fast Fourier Transform (FFT) in the most efficient way. No management of host is required, and it can compute a FFT of a 4096 points real vector in 19,64 ms.

Abreviaturas Utilizadas

Foi procurado adotar no texto, sempre que possível, terminologia em português. As expressões em outro idioma foram utilizadas apenas quando são de uso consagrado, mas sempre tomando cuidado de se apresentar uma expressão equivalente no nosso idioma. Nesta lista não estão as abreviaturas referentes a instruções e parâmetros do Processador Vetorial de Sinais.

AD	Analógico / Digital
BFP	<i>Block Floating Point</i> (Ponto Flutuante em Blocos)
BIU	<i>Bus Interface Unit</i> (Unidade de Acesso ao Barramento)
CT	Cooley e Tukey
DFT	<i>Discrete Fourier Transform</i> (Transformada Discreta de Fourier)
DMA	<i>Direct Memory Access</i> (Acesso Direto a Memória)
DSP	<i>Digital Signal Processor</i> (Processador de Sinais Digitais)
EISA	<i>Extended Industry Standard Architecture</i> (Arquitetura Industrial Padrão Extendida)
EU	<i>Execution Unit</i> (Unidade de Execução)
FFT	<i>Fast Fourier Transform</i> (Transformada Rápida de Fourier)
GB	Gold e Bially
I/O	<i>Input / Output</i> (Entrada / Saída)
ISA	<i>Industry Standard Architecture</i> (Arquitetura Padrão Industrial)
MAq	Módulo de Aquisição
MCA	<i>Micro Channel Architecture</i> (Arquitetura Micro Canal)
MPr	Módulo de Processamento
PAL	<i>Program Array Logic</i> (Dispositivo de Lógica Programável)
PC	<i>Personal Computer</i> (Computador Pessoal)
PFA	<i>Prime Factors Algorithms</i> (Algoritmos por Fatores Primos)
RAM	<i>Random Memory Access</i> (Memória de Acesso Aleatório)
ULA	Unidade Lógica e Aritmética
VSP	<i>Vetorial Signal Processing</i> (Processador Vetorial de Sinais)
WFTA	<i>Winograd Fourier Transform Algorithms</i> (Algoritmos de Transformada de Fourier de Winograd)

Introdução

A Espectroscopia por Transformada de Fourier possui vantagens sobre os métodos convencionais no que diz respeito a sensibilidade e resolução [BEL-72], mas tem o inconveniente de exigir uma grande demanda computacional, na forma da Transformada Discreta de Fourier (DFT). Mesmo os algoritmos desenvolvidos para minimizar o número de cálculos envolvidos no seu processamento, chamados genericamente de Transformada Rápida de Fourier (FFT), ainda exigem grande poder computacional. Isso tem limitado o uso mais amplo deste tipo de espectroscopia, principalmente em aplicações que exijam rapidez na obtenção dos resultados.

Os Processadores de Sinais Digitais (DSP) foram desenvolvidos visando esta classe de aplicação. São processadores otimizados para a execução de algoritmos de processamento de sinais [SCH-92]. Entre suas principais características está um conjunto de instruções reduzido e orientado ao processamento de sinais, como por exemplo, instruções do tipo produto-soma [DYE-88], e o uso de paralelismo temporal, ou *pipe-lining* [HWA-86]. O desempenho dos DSP atuais supera largamente o dos primeiros computadores científicos e supercomputadores [KOR-79].

Existem DSP especializados na execução de uma classe de algoritmos, cuja arquitetura é ainda mais orientada a aplicação que a dos DSP comuns. O Processador Vetorial de Sinais (VSP) ZR34161, desenvolvido pela ZORAN, está dentro deste grupo, e é projetado para o processamento eficiente de FFT em vetores de dados. Esta foi a solução adotada neste trabalho para acelerar os cálculos de FFT, e o VSP é o componente principal do módulo de processamento (MP_r).

A aquisição dos dados é outra tarefa crítica para o desempenho do sistema. O Módulo de Aquisição (MA_q) foi projetado para cumpri-la de modo eficiente. Ele opera independentemente do controle do hospedeiro visando atingir taxas de aquisição mais elevadas.

Os módulos de aquisição e processamento estão na forma de cartões de expansão para barramento ISA de 16 bits, já que este tipo de barramento oferece maior velocidade de transferência de dados que o barramento ISA de 8 bits. Não foram considerados outros padrões de barramento mais rápidos, tais como EISA e MCA [ROS-90], pois o barramento ISA ainda é mais difundido.

As outras tarefas a serem executadas neste sistema são a exibição dos resultados do processamento e a comunicação com o usuário. Um microcomputador pessoal da linha IBM-PC modelo AT ou superior foi escolhido para ser o computador hospedeiro, responsável por elas.

O sistema ótico do interferômetro é descrito no capítulo 1, pois é ele que define os parâmetros de desempenho que guiaram o projeto do MA_q e a concepção do algoritmo de FFT do MP_r. São apresentados os conceitos físicos necessários para a compreensão do interferômetro e da geração do sinal de interferência, e posteriormente são apresentados os parâmetros de desempenho desejados.

O Capítulo 2 apresenta detalhadamente a FFT segundo Gold e Bially, que é uma versão do algoritmo de Cooley e Tukey com vantagens para programação no VSP. Outros algoritmos de FFT são apresentados e é justificada a razão da sua não adoção. São introduzidas algumas funções de apodização e são salientados seus benefícios.

O Capítulo 3 introduz a arquitetura e as instruções do VSP, com ênfase nos conceitos efetivamente empregados neste trabalho.

O MAq tem sua estrutura analisada no Capítulo 4. São expostos os seus recursos, blocos funcionais, lógica de controle e esquema de comunicação com o hospedeiro.

O Capítulo 5 descreve rapidamente a arquitetura do MPr, e em profundidade o programa que implementa a FFT. Discute mais detalhadamente o algoritmo de FFT apresentado no Capítulo 2, salientando sua implementação com as instruções vetoriais do VSP e estimando seu tempo de processamento. Apresenta também a técnica *autogen* [MOR-77] de geração de programas.

O Capítulo 6 apresenta os testes de validação dos módulos e seus resultados.

Finalmente o Capítulo 7 encerra o trabalho apresentando as conclusões obtidas e sugerindo linhas de desenvolvimento.

1. Interferometria

1.1. Introdução

A Interferometria, ou Espectroscopia por Transformada de Fourier, é uma ferramenta com vasto uso nas áreas científica e industrial [BEL-72]. É empregada em campos que vão do estudo de problemas ambientais à astronomia de espaço profundo, passando pela indústria aeronáutica na análise dos gases de exaustão de motores a reação. Ela foi desenvolvida pelo Dr. Albert A. Michelson em 1880, quando criou o interferômetro que leva seu nome. Sua primeira aplicação foi na medida da velocidade da luz no vácuo. Devido a ausência de computadores e de métodos eletrônicos de detecção de sinais, esta ferramenta teve uso limitado na época, embora Michelson tivesse criado um computador mecânico analógico para auxiliar o cálculo das transformadas de Fourier, chamado de analisador harmônico. O primeiro interferograma registrado por meios eletrônicos foi obtido por Rubens e Wood, em 1911 [BEL-72].

A Espectroscopia por Transformada de Fourier possui várias vantagens sobre as técnicas tradicionais de espectroscopia. As mais importantes são as chamadas vantagens de Fellgett e Jacquinot, discutidas mais adiante. Além delas, existem diversos outros aspectos positivos:

- elevado poder de resolução
- menor custo do equipamento ótico
- maior velocidade de aquisição
- menor peso e tamanho do interferômetro

As desvantagens desta técnica estão relacionadas ao fato dela exigir um sistema computacional para a obtenção do resultado, e nos atrasos envolvidos neste processamento numérico.

1.2. Interferômetro de Michelson

O interferômetro de Michelson foi o primeiro interferômetro desenvolvido, em 1880. Sua simplicidade ajuda na compreensão dos conceitos envolvidos na geração do interferograma. A Fig 1.1 representa o seu diagrama ótico simplificado.

Uma fonte de luz (A) emite a radiação na faixa espectral desejada- visível, infravermelho, etc. Essa radiação é colimada (B) em um feixe que é dividido em dois pelo divisor de feixe (*beamsplitter*) (C). Uma parte do feixe se reflete em um espelho fixo (D) e retorna ao divisor de feixe. A outra parte do feixe é refletida pelo espelho móvel (E), e também retorna ao divisor de feixe. As duas partes do feixe interferem-se e o divisor de feixe reflete essa interferência para um espelho (F) que a focaliza em um detetor (G), passando pela amostra em análise (H). O padrão de interferência que atinge o detetor é chamado interferograma.

Fig 1.1 - Interferômetro de Michelson

1.3. Fundamentos da Espectroscopia por Transformada de Fourier

Supondo que a fonte de radiação luminosa da Fig 1.1 seja uma fonte monocromática de comprimento de onda λ - um laser, por exemplo - o resultado da interferência dos dois feixes no divisor de feixe dependerá do caminho ótico percorrido por eles. Se eles estiverem a mesma distância do divisor de feixe, o caminho ótico dos feixes será exatamente o mesmo e os dois sinais se reforçam, gerando uma interferência construtiva. Deslocando-se o espelho móvel $\lambda/4$, de modo que a diferença dos caminhos óticos dos dois feixes seja $\lambda/2$, os feixes estarão defasados de 180° e a interferência será destrutiva, anulando o sinal de interferência. Continuando-se o deslocamento do espelho móvel, a defasagem dos dois feixes vai se reduzindo e o sinal de interferência aumenta, gerando um sinal periódico cosenoidal, como o exibido na Fig 1.2.

Este mesmo raciocínio pode ser adotado para analisar o comportamento de uma fonte não monocromática, devido a linearidade do sistema. Cada comprimento de onda da fonte de radiação se comporta da mesma maneira, e o interferograma resultante será a soma das interferências produzidas por todos comprimentos de onda.

Fig 1.2 - Interferograma de fonte monocromática tomado a partir do deslocamento do espelho

1.4. Espectrofotômetro de Varredura Refrativa

Este aparelho utiliza-se de um interferômetro de Varredura Refrativa. Nele, o dispositivo que varia o caminho ótico percorrido pela luz é uma cunha semitransparente. O feixe de radiação fornecido pela fonte (A) atravessa a amostra (G) e é enviado para a cunha fixa (C), onde é dividido em dois. Um destes feixes é refletido em um espelho canto de cubo fixo (D) e retorna à cunha. O outro feixe atravessa uma cunha móvel (B) e é refletido por outro espelho canto de cubo, igualmente fixo (E). Os feixes se interferem na cunha e são refletidos para o detetor (F). Como a velocidade da luz nas cunhas é diferente da sua velocidade no ar, é produzida uma diferença de fase entre os dois feixes luminosos. O interferograma é obtido através do deslocamento da cunha móvel, variando a seção da cunha percorrida pelo raio de luz. O diagrama do interferômetro de Varredura Refrativa é representado na Fig 1.3.

Fig 1.3 - Interferômetro de Varredura Refrativa

1.5. Vantagens da Espectroscopia por Transformada de Fourier

1.5.1. Vantagem de Fellgett

A vantagem de Fellgett, também conhecida por vantagem de multiplexação (*multiplex advantage*), vem do fato do detector receber informação de todos os componentes espectrais durante toda a leitura do interferograma, enquanto um espectrômetro tradicional dispersivo recebe apenas a informação da parte do espectro selecionada por uma fenda estreita.

Isto significa que nos espectrômetros tradicionais quanto maior a resolução menor será a energia de um dado componente espectral que excitará o detector, o que limita seu uso para sinais intensos. Já com os interferômetros não há essa limitação.

1.5.2. Vantagem de Jacquinot

A vantagem de Jacquinot, também conhecida por vantagem de transferência de potência (*throughput*) reza que um interferômetro apresenta menos perdas de potência luminosa no percurso do feixe da fonte até o detector que um espectrômetro. Nos espectrômetros dispersivos a radiação emitida pela fonte atinge o detector depois ter passado por duas fendas, pelo menos, e tais obstruções reduzem a transferência de potência do aparelho. Nos interferômetros só existe perda de potência na radiação que não é colimada pelos espelhos, mas esta perda é pequena se comparada com a dos espectrômetros.

1.6. Considerações Práticas

A leitura do interferograma deve ser feita em função da posição do espelho móvel. Isso é trivial se um motor de passo desloca o espelho. Mas se o deslocamento for feito por meio de motor de corrente contínua, não existe uma referência de posição simples. Se a leitura do interferograma for feita segundo uma base de tempo, a velocidade de deslocamento do espelho móvel passa a ser um fator relevante. Se ela for constante $v_d = \lambda/T_d$, o interferograma obtido é um sinal cosenoidal de frequência bem definida $f_d = 1/T_d$. Se a velocidade v_d não for constante, o interferograma perde sua forma e sua frequência característica, como exibido na Fig 1.1.

Pode-se providenciar uma referência de deslocamento para o interferômetro por meio de uma fonte adicional de radiação monocromática. Este feixe de luz percorre quase os mesmos caminhos que o feixe original, mas é alinhado de maneira que sua interferência não se incorpore ao interferograma original. O seu interferograma tem um aspecto cosenoidal e é lido separadamente, em outro detetor, fornecendo a referência de deslocamento necessária. Neste caso a velocidade de deslocamento do espelho móvel não é um fator crítico, mas ainda assim deve-se ter cuidado para evitar que a frequência instantânea do interferograma supere a frequência máxima suportada pelo sistema de detecção.

No interferômetro implementado, o deslocamento da cunha é feito pelo motor de corrente contínua, e o sinal de referência é dado por um laser de HeNe.

Outro aspecto importante diz respeito ao início da aquisição do interferograma. Este início pode ser determinado usando a referência de posição fornecida pela fonte monocromática, ou pode ser determinado pelo próprio interferograma, através de um sistema eletrônico que discrimine quando o interferograma atinja determinada intensidade.

Fig 1.4 - Interferograma de fonte monocromática tomado a partir do tempo

1.7. Requisitos de Desempenho

1.7.1. Módulo de Aquisição

O MAq deve ser capaz de fazer as aquisições comandado pelo sinal de referência de deslocamento gerado pelo laser de referência. A velocidade de aquisição deve ser a mais elevada possível, pois devido a variações na velocidade de deslocamento da cunha, existem instantes em que a taxa de aquisição pode ser muito elevada. Também por esta razão é importante que o módulo possa detectar se houve perda de dados ou tentativa de sobre-aquisição. A taxa de aquisição proposta foi de 100 KHz, com uma precisão de 12 bits.

Estes requisitos impedem que o MAq seja controlado pelo hospedeiro, exigindo uma lógica de controle exclusiva e de alta velocidade. Como as tarefas de aquisição são simples e devem ser executadas no menor tempo possível não se justifica o emprego de um processador dedicado. Em seu lugar preferiu-se projetar uma máquina de estados, que preenche melhor os requisitos de desempenho. Graças ao pequeno número de estados exigidos para implementar o controle de aquisição, o tempo gasto nas tarefas de controle é da ordem de grandeza do tempo de acesso das memórias usadas no MAq.

O armazenamento dos dados deve interferir o mínimo com o hospedeiro para permitir que ele execute outras tarefas em paralelo. Segundo [EGG-90] a taxa máxima que se pode atingir num computador IBM-PC usando DMA é 476K palavras por segundo, com a redução da capacidade de processamento do hospedeiro pela metade. Esta interferência com o hospedeiro durante a aquisição dos dados não é aceitável. A solução adotada foi armazenar os dados temporariamente no próprio MAq e transferi-los ao hospedeiro ao fim do processo da aquisição. O MAq deve ser capaz de armazenar completamente um interferograma, que tem o tamanho proposto de 4K palavras de 16 bits. O módulo deve permitir ampliação da capacidade de armazenamento.

A transferência dos dados armazenados no MAq é feita sob o controle do hospedeiro, por meio de um programa otimizado. Para isso o hospedeiro deve poder acessar a memória do MAq. Para maior velocidade na aquisição e transferência dos dados, é interessante que este módulo possa realizar uma transferência de dados de 16 bits de uma só vez, o que implica no uso em hospedeiros com barramento ISA de 16 bits.

1.7.2. Módulo de Processamento

O MP_r adotado é uma placa de desenvolvimento para o VSP ZR34161, fornecida pela ZORAN. Esta placa deve ser conectada a barramentos ISA-16 bits e contém um VSP ligado a uma memória local, compartilhada com o hospedeiro, de 16K palavras de 16 bits. O *clock* do processador adotado na placa é de 20 MHz, logo um ciclo de máquina do VSP é de 100 ns.

Avaliações iniciais confirmaram que com este *clock* um desempenho de 10 FFT por segundo em vetores de 4096 elementos poderia ser atingido facilmente, e até superado. A transferência dos dados do/para o VSP é feita por programa otimizado. O DMA não é suportado pela placa de desenvolvimento.

2. Transformada Rápida de Fourier

2.1. Introdução

O emprego da Transformada Discreta de Fourier se generalizou após a publicação por Cooley e Tukey do artigo que apresentava um método de cálculo da DFT que exigia menos operações matemáticas e tornava prático o cálculo numérico da DFT [COO-66]. O algoritmo de DFT segundo a definição é de classe N^2 e o de Cooley e Tukey, de classe $M \log N$. Este foi o primeiro algoritmo de FFT. Não existe "o" algoritmo de FFT, mas toda uma classe deles, todos denominados coletivamente "algoritmos de FFT".

Não devem ser confundidos com FFT alguns métodos que minimizam o tempo de computação reduzindo o número de multiplicações, como o de Goertzel, mas que mantém a ordem N^2 do algoritmo de DFT [GOL-69].

A Transformada de Fourier [BRI-74] e o algoritmo de Cooley e Tukey já são bastante conhecidos, com farto material de referência [COO-66] [OPP-75], e por isso não serão discutidos em detalhes. Em lugar disso será apresentada uma abordagem diferenciada do algoritmo de Cooley e Tukey, introduzida por Gold e Bially [GOL-69]. Neste algoritmo a FFT é expressa quase exclusivamente em termos de FFT menores em linhas e colunas de matrizes, com um mínimo de adições e produtos. Isto é particularmente conveniente para este trabalho pois o VSP usado possui arquitetura otimizado para a execução de FFT.

O desenvolvimento do algoritmo de Gold e Bially apresentado a seguir foi elaborado a partir de [GOL-69] e de outras referências [STI-82] [ROB-87] [MYE-90].

2.2. FFT segundo Gold e Bially

Seja $\mathbf{x}$ um vetor complexo com N elementos:

$$(2.1) \quad \mathbf{x} = \{x_0, x_1, \dots, x_{N-1}\}$$

Segundo a definição, o vetor $\mathbf{X}$ complexo, que é a DFT de $\mathbf{x}$, é assim calculado:

$$(2.2) \quad X(n) = \sum_{k=0}^{N-1} W^{nk} x(k), \quad n = 0, 1, \dots, N-1$$

$$= e^{-j \frac{2\pi}{N} nk}$$

O algoritmo de Cooley e Tukey exige que N seja fatorável, da forma:

$$(2.3) \quad N = L \cdot M$$

Desse modo, os índices usados em (2.2) podem ser reescritos:

$$(2.4) \quad \begin{aligned} k &= r + Ls, & r &= 0, 1, \dots, L-1, & s &= 0, 1, \dots, M-1 \\ n &= Ml + m, & l &= 0, 1, \dots, L-1, & m &= 0, 1, \dots, M-1 \end{aligned}$$

Substituindo-se (2.4) em (2.2) pode-se representar o vetor $\mathbf{x}$ como uma matriz $\mathbf{a}$:

$$(2.5) \quad x(k) = x(r + Ls) = a(r, s) = \begin{bmatrix} x_0 & x_L & \cdots & x_{(M-1)L} \\ x_1 & x_{L+1} & \cdots & x_{(M-1)L+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{L-1} & x_{2L-1} & \cdots & x_{ML-1} \end{bmatrix}$$

Com a substituição de (2.4) em (2.2), a equação da DFT pode ser reescrita:

$$(2.6) \quad X(Ml + m) = \sum_{s=0}^{M-1} \sum_{r=0}^{L-1} W_N^{(r+Ls)(Ml+m)} x(r + Ls)$$

Desenvolvendo (2.6):

$$(2.7) \quad X(Ml + m) = \sum_{r=0}^{L-1} W_N^{r(Ml+m)} \sum_{s=0}^{M-1} W_N^{Lsm} x(r + Ls)$$

É fácil observar que a somatória mais interna não é dependente do índice l , apenas de m , r e s . Isso significa que uma vez calculada esta somatória de M termos para todos os valores de m e r possíveis, estes resultados podem ser usados para qualquer valor de l , sem novas computações. Pode-se encarar esta somatória como uma matriz de dimensões $L \times M$, onde cada elemento é obtido pela soma de M termos.

$$(2.8) \quad b(r, m) = \sum_{s=0}^{M-1} W_N^{Lsm} x(r + Ls)$$

Simplificando a somatória interna, tem-se:

$$(2.9) \quad W_N^{Lsm} = W_N^{Msm} = W_M^{sm}$$

$$(2.10) \quad b(r, m) = \sum_{s=0}^{M-1} W_M^{sm} x(r + Ls)$$

A equação (2.10) nada mais é que a equação (2.2) aplicada a r -ésima linha da matriz $\mathbf{a}$ definida em (2.5). Em outras palavras, a r -ésima linha da matriz $\mathbf{b}$ é obtida pelo cálculo da DFT sobre a r -ésima linha da matriz $\mathbf{a}$.

Usando (2.10) para reescrever (2.7):

$$(2.11) \quad X(Ml + m) = \sum_{r=0}^{L-1} W_N^{r(Ml+m)} b(r, m)$$

Esta equação demonstra que a FFT de N pontos pode ser calculada por meio de L operações de FFT básicas em M pontos, e N operações de soma-adição em L pontos. Neste ponto o desenvolvimento do método de Gold e Bially diverge do de Cooley e Tukey.

Desenvolvendo e simplificando (2.11) do mesmo modo que em (2.9):

$$(2.12) \quad X(Ml + m) = \sum_{r=0}^{L-1} W_L^{lr} W_N^{rm} b(r, m)$$

O produto mais interno da equação (2.12) apenas depende dos índices r e m . Assim pode-se interpretar este produto como uma matriz $\mathbf{c}$ assim definida:

$$(2.13) \quad c(r, m) = W_N^{rm} \cdot b(r, m)$$

Reescrevendo (2.12) usando (2.13):

$$(2.14) \quad X(Ml + m) = \sum_{r=0}^{L-1} W_L^{lr} c(r, m)$$

Fica evidente que (2.14) é a equação (2.2) aplicada a m -ésima coluna da matriz c . Se o vetor X for representado por uma matriz A , repetindo a visualização de (2.5):

$$(2.15) \quad A(l, m) = \sum_{r=0}^{L-1} W_L^{lr} c(r, m)$$

Pode-se dizer que a m -ésima coluna da matriz A é a DFT aplicada sobre a m -ésima coluna da matriz c . Para se obter X , finalmente, basta mapear em um vetor a matriz A segundo os índices definidos em (2.4).

O algoritmo de Gold e Bially pode ser representado esquematicamente na Fig 2.1.

Fig 2.1 - Esquema de FFT segundo Gold e Bially

2.3. Outros algoritmos de FFT

A idéia fundamental do algoritmo descrito anteriormente é transformar um vetor x em um arranjo bidimensional a para reduzir o número de cálculos na DFT. Para isso foram usadas as equações de mapeamento (2.4). Estas equações são um caso particular de uma classe de equações de mapeamento da forma:

$$(2.16) \quad \begin{aligned} k &= (K_1 r + K_2 s) \bmod N \\ n &= (K_3 l + K_4 m) \bmod N \end{aligned}$$

Os inteiros K_1 e K_2 devem ser escolhidos de modo que o mapeamento de a em x seja do tipo um para um, sem repetições. Idem para K_3 e K_4 . Mas se estes fatores forem escolhidos de modo a explorar as simetrias e periodicidade da DFT, pode ser alcançada uma grande redução nos cálculos da DFT. Segundo Good [GOO-71], uma boa seleção para estes parâmetros seria:

$$(2.17) \quad \begin{aligned} K_1 &= 1 & K_2 &= 1 \\ K_3 &= M & K_4 &= L \end{aligned}$$

Este mapeamento é válido apenas quando os fatores L e M que compõem N são primos entre si, pois só nesse caso o mapeamento é do tipo um para um. O grande benefício desta escolha é eliminar a ponderação pelos pesos W exigida em (2.13). A Fig 2.2 apresenta o diagrama esquemático deste algoritmo de FFT.

Fig 2.2 - Esquema de FFT segundo o Algoritmo de Fatores Primos

O algoritmo de Good pertence a uma subclasse de algoritmos de FFT denominada Algoritmos de FFT por Fatores Primos (PFA). Outro algoritmo dessa subclasse é o de Winograd, ou WFTA. O Dr. Shmuel Winograd descobriu uma decomposição da matriz de ponderação W que permite o cálculo da DFT de pequenos vetores (os mais comuns possuem 2, 3, 4, 5, 7, 8, 9 e 16 elementos [BLA-85]) com muito menos operações [SIL-77]. Compondo-se estas pequenas FFT para construir FFT maiores, segundo o algoritmo PFA, podem-se conseguir grandes reduções no número de operações.

2.4. FFT de Vetores Reais

Nas aplicações voltadas a espectroscopia, o vetor de dados (interferograma) a ser processado pelo algoritmo de FFT é puramente real. Os algoritmos descritos acima operam sobre um vetor complexo. Pode-se aproveitar algumas propriedades da DFT para conseguir a transformada de um vetor real de N elementos através de uma FFT em um vetor complexo de $N/2$ elementos.

Segundo [NUS-81] pode-se arranjar o vetor real x de N elementos em um vetor complexo y de $N/2$ elementos da seguinte forma:

$$\begin{aligned}
 h(k) &= x(2k) \\
 g(k) &= x(2k+1) \\
 y(k) &= h(k) + jg(k)
 \end{aligned}
 \quad k = 0, 1, \dots, \frac{N}{2} - 1$$

A transformada de Fourier de y pode ser escrita assim:

$$Y(k) = H(k) + jG(k) \quad (2.19)$$

Pode-se calcular H e G utilizando-se das propriedades da transformada de Fourier X de um vetor real.

$$\begin{aligned}
 \text{Re}\{X(k)\} &= \text{Re}\{X(-k)\} \\
 \text{Im}\{X(k)\} &= -\text{Im}\{X(-k)\}
 \end{aligned}
 \quad (2.20)$$

Sabendo-se isto, determina-se H e G .

$$\begin{aligned}
 (2.21) \quad \text{Re}\{H(k)\} &= \frac{\text{Re}\{Y(k)\} + \text{Re}\{Y(-k)\}}{2} & \text{Re}\{G(k)\} &= \frac{\text{Im}\{Y(k)\} + \text{Im}\{Y(-k)\}}{2} \\
 \text{Im}\{H(k)\} &= \frac{\text{Im}\{Y(k)\} - \text{Im}\{Y(-k)\}}{2} & \text{Im}\{G(k)\} &= \frac{\text{Re}\{Y(-k)\} - \text{Re}\{Y(k)\}}{2}
 \end{aligned}$$

Finalmente, segundo (2.11) pode-se calcular o vetor $\mathbf{X}$ a partir de $\mathbf{H}$ e $\mathbf{G}$, dessa forma:

$$(2.22) \quad X(n) = H(n) + W_N^n G(n) \quad n = 0, 1, \dots, N-1$$

2.5. Funções de Apodização

Uma das características da DFT é que ela é aplicada a um vetor finito, ao contrário da Transformada Contínua de Fourier. Este vetor finito é obtido limitando-se a amostragem do sinal $x(t)$ a uma região de interesse. Matematicamente isto é feito multiplicando-se $x(t)$ por uma função de "janela", ou função de apodização, $j(t)$. No domínio da frequência isso significa que os espectros dos dois sinais são convoluídos [BRI-75], como mostra a Fig 2.3.

Fig 2.3 - Influência do truncamento temporal no domínio da frequência

A função de janela deve ter seu espectro o mais próximo possível de uma função impulso, de modo que ele não altere o espectro do sinal, mas a função que possui este tipo de espectro é justamente a janela de tamanho infinito. Para substituí-la existem algumas funções que possuem espectros particularmente interessantes para serem usadas como funções de apodização [NUT-81]. Todas estas funções são limitadas no tempo dentro do intervalo T_0 e os seus espectros são apresentados normalizados pela frequência $f_0 = 1/(2T_0)$.

A função de apodização mais comum é a função retângulo, que é aplicada simplesmente truncando o sinal a ser lido. Matematicamente ela é assim definida:

$$j_r(t) = \begin{cases} 1, & t < T_0 \\ 0, & t \geq T_0 \end{cases}$$

Outras funções de apodização podem ser obtidas através da equação (2.18). São as funções de apodização harmônicas.

$$(2.18) \quad j_h(t) = \sum_k a_k (-1)^k \sin(2\pi f_0 t), \quad t < T_0$$

As constantes a_k e o número k de termos determinam o comportamento e as características da função de apodização. A seguir serão apresentadas algumas funções de apodização, juntamente com os seus espectros de potência característicos.

2.5.1. Função Retângulo

Fig 2.4 - Espectro de potência da função retângulo

O espectro da função retângulo tem o primeiro lóbulo estreito, mas o segundo possui uma amplitude elevada de -13.5 dB e os subseqüentes possuem uma taxa de decaimento reduzida, de 6 dB/oitava. Isso implica em interferências significativas na vizinhança do ponto de aplicação desta função.

2.5.2. Função de Hanning

Fig 2.5 - Espectro de potência da função de Hanning

O primeiro lóbulo da função de Hanning é mais largo que aquele da função retângulo, mas seu segundo lóbulo é bem menor, com -31.5 dB. Os lóbulos subsequentes decaem assintoticamente a uma taxa de 18 dB/oitava.

Esta função é definida com os valores:

$$a_0 = 0.5, a_1 = 0.5$$

2.5.3. Função de Blackman

Fig 2.6 - Espectro de potência da função de Blackman

O primeiro lóbulo da função de Blackman é mais largo que aquele da função de Hanning, mas seu segundo lóbulo é ainda menor, com -58.1 dB. Os lóbulos subsequentes decaem assintoticamente a uma taxa de 18 dB/oitava.

Esta função é definida com os valores:

$$a_0 = 0.42, a_1 = 0.50, a_2 = 0.08$$

A função de Blackman se mostra a mais interessante de todas pois possui lóbulos laterais de amplitude reduzida aliada a uma taxa de decaimento acelerada. Os lóbulos laterais reduzidos diminuem a influência da função em torno do ponto de aplicação e a taxa de decaimento elevada assegura que não há interferências nas frequências distantes do ponto central.

Estes conceitos serão retomados no capítulo 5, que trata da implementação do algoritmo de FFT no VSP.

3. Processador Vetorial de Sinais

3.1. Introdução

O processador usado para acelerar o cálculo da FFT é o ZR34161, um processador com arquitetura interna otimizada para operações de FFT, convolução e correlação em vetores de até 128 pontos [ZOR-86]. Além disso permite operações gerais em vetores, inclusive as operações de produto e acumulação, características dos Processadores Digitais de Sinal (DSP). Seus cálculos são efetuados sobre vetores inteiros de 19 bits, mas o uso do Ponto Flutuante em Bloco (BFP) permite maior precisão que a aritmética inteira [ZOR-86]. O processador ZR34161 é denominado Processador Vetorial de Sinais (VSP). Algumas características são:

- 23 instruções vetoriais de alto nível
- Aritmética inteira e de ponto flutuante em bloco
- Barramentos de dados e endereços de 16 bits
- Execução concorrente de instruções aritméticas e de I/O
- Ciclo de instrução de 100 ns
- Suporte a DMA
- Operações com vetores de até 128 pontos

3.2. Arquitetura

O VSP possui 4 blocos funcionais principais: a RAM de Dados, a Unidade de Execução, a Unidade de Acesso a Barramento e os Registradores. Sua arquitetura e módulos funcionais está representada na Fig 3.1.

Material sobre arquiteturas dedicadas a processamento de FFT pode ser encontrado em [FLA-78] [VET-86] e principalmente em [GUN-85].

Fig 3.1 - Arquitetura do VSP

3.2.1. RAM de Dados

O VSP possui uma memória RAM interna para armazenamento do vetor a ser processado. Esta memória é estruturada como um vetor com 128 palavras de 38 bits, como indicado na Fig 3.2. Esta palavra é dividida em parte real e imaginária, cada uma com 19 bits. Os 2 bits mais significativos não podem ser acessados e servem apenas como bits de guarda para prevenção de estouros (*overflow*) em operações de FFT. O bit menos significativo é usado em cálculos, e no caso de transferência do vetor para a memória externa, para arredondamento dos 16 bits restantes.

Fig 3.2 - Organização da RAM de Dados

Todas as instruções vetoriais operam sobre essa memória. Se a instrução exige mais um operando, ele será buscado na memória externa. Os resultados das instruções aritméticas também são aí armazenados. A memória de dados suporta um vetor de tamanho máximo de 128

elementos, não importando que eles sejam reais, imaginários ou complexos. Quando é necessário fazer operações sobre vetores maiores, as instruções vetoriais devem ser usadas repetidamente.

Esta memória pode ser dividida em duas seções de 64 elementos cada. Dessa forma instruções aritméticas que utilizem uma seção de memória podem ser executadas em paralelo com instruções de I/O que utilizem a outra seção. A decisão de executar em paralelo duas instruções é tomada pelo próprio processador. Como os barramentos de dados internos do VSP acomodam a parte real e a parte imaginária simultaneamente, e o barramento de dados externo apenas comporta uma parte do número complexo, durante uma transferência de dados complexos o barramento de dados interno do VSP fica metade do tempo inativo [ZOR-87]. O paralelismo é obtido utilizando-se estes ciclos inativos. Como existe a opção de se inserir ciclos de espera (*wait states*) nas transferências, para operação com memórias externas lentas, o tempo inativo do barramento interno pode ser ainda maior, tornando a execução paralela ainda mais vantajosa.

3.2.2. Unidade de Execução

A Unidade de Execução (EU) contém a Unidade Lógico Aritmética (ULA) e é responsável pelas operações de processamento. A ULA é estruturada para calcular eficientemente as *butterflies* do algoritmo de FFT radix-2 [BRI-75]. Para isso ela possui um multiplicador inteiro de 17 bits, um somador, um somador/subtrator e um acumulador complexo de 25 bits. O acumulador tem essa extensão para poder armazenar o resultado de uma FFT de 7 estágios (128 pontos) em números de 17 bits, que produz uma palavra extensa.

A EU, representada na Fig 3.3, pode executar FFT com aritmética inteira ou de BFP [OPP-75] [ZOR-87]. Com aritmética inteira o vetor é dividido por 2 a cada estágio da FFT. Usando BFP, as divisões são feitas apenas se há estouro (*overflow*), sendo registradas e acumuladas. Ao término da instrução de FFT, o número de divisões passa a ser o expoente de todo este bloco de dados e pode ser associado a ele. Posteriormente, ao fim de todas as FFT necessárias para o algoritmo, todos os blocos de dados devem ser normalizados em função do maior expoente. Isso aumenta a faixa dinâmica dos cálculos, juntamente com a precisão dos resultados. Um estudo em [OPP-75] demonstra que a relação sinal/ruído dos resultados da FFT usando BFP é 18 dB maior que a da FFT usando aritmética de ponto fixo, para um vetor de 2048 pontos.

Fig 3.3 - Arquitetura da Unidade de Execução

3.2.3. Unidade de Interface com Barramento

A Unidade de Interface com Barramento (BIU) é responsável pelas tarefas que exigem acesso a barramento, como a leitura de instruções, leitura e escrita de dados e DMA. É uma unidade independente da EU, e a arquitetura do VSP permite que operem em paralelo, se não se utilizarem da mesma seção de memória. Este recurso permite que a EU seja mantida constantemente em operação, sem intervalos de inatividade para leitura e escrita de dados.

3.2.4. Registradores

Utilizados pela EU e pela BIU, o VSP possui Registradores de Modo, Status, registros de escala, RAM de Escala e uma Tabela de Senos/Cosenos.

Os Registros de Modo e Status, de 16 bits, configuram e refletem o modo de operação do dispositivo.

A RAM de Escala é organizada como 16 palavras de 16 bits cada, palavras estas que contém 4 expoentes de 4 bits cada. Ao todo é capaz de armazenar 64 expoentes.

Os registros de escala são 3: o Registro de Escala propriamente dito, o Registro de Máximo Expoente e o Registro de Último Expoente.

O Registro de Escala é uma palavra com 16 bits de largura e capacidade para 4 expoentes. A cada FFT o expoente resultante é armazenado sequencialmente neste registro. Os expoentes armazenados no Registro de Escala devem ser salvos na memória RAM externa a cada 4 FFT, sob pena de perda de expoentes.

O Registro de Máximo Expoente é usado para manter o maior expoente de uma sequência de FFT, e ao fim da sequência de FFT estabelecida, normalizar todos os vetores de resultados baseado neste valor.

O Registro de Último Expoente armazena o último expoente gerado por uma FFT.

A Tabela de Senos e Cosenos é organizada como 256 palavras de 16 bits representando os valores dos senos de ângulos variando de 0 a $\pi/2$ em incrementos $\pi/512$. Uma lógica interna permite que se obtenha a partir dela 1024 valores de senos e cosenos nos 4 quadrantes do círculo trigonométrico, além de gerar um bit adicional de sinal. É utilizada para cálculos de FFT de até 1024 pontos.

3.3. Modos de Operação

O VSP pode operar de modo autônomo, buscando suas próprias instruções e dados, ou apenas executando as instruções escritas no seu Registro de Execução por um computador de controle. A estes dois modos de operação dá-se respectivamente o nome de Modo Mestre e Modo Escravo. O modo de operação é configurado através de um bit no Registro de Modo. O Modo Escravo permite que se acesse a memória e os registros internos do VSP. Neste trabalho o VSP operará apenas no Modo Mestre.

3.4. Interrupções

O VSP possui o sinal INT, de saída, que é ativado por determinado evento e permite que o VSP interrompa o computador hospedeiro. As classes de eventos que podem ativar o sinal INT são:

- Execução de uma instrução específica
- Estouro nos cálculos
- Fim do programa

A geração da interrupção pode ser mascarada individualmente para cada classe de eventos. A interrupção ao término de uma instrução específica é habilitada pelo campo EI, presente em todas as instruções. A interrupção nas outras classes de eventos é habilitada através do Registro de Modo. O hospedeiro pode determinar a classe de eventos que gerou a interrupção pelo conteúdo do Registro de Status.

3.5. Instruções

O VSP possui 23 instruções de alto nível divididas e 4 categorias: instruções de I/O, aritméticas de um operando, instruções aritméticas de dois operandos, e de controle. Serão tratadas aqui apenas as instruções utilizadas neste trabalho.

Devido ao alto nível das instruções do VSP, cada uma delas possui uma série de parâmetros que modificam a sua operação. Existem alguns parâmetros que são comuns a maioria delas. São eles:

- ADF: seleciona o formato (ADF:1, 2 ou 3 - imaginário, real ou complexo) do resultado que será armazenado na RAM interna. Pode-se escolher armazenar apenas a parte real ou imaginária do resultado.
- CN: indica que o operando na memória externa é uma constante (CN:1) em lugar de um vetor.
- EI: habilita geração de interrupção ao fim da instrução (EI:1).
- MBA: aponta para a posição de memória externa onde se inicia o vetor ser processado.
- MBS: define o tamanho do bloco sequencial de um vetor armazenado na memória externa, isto é, especifica o número de pontos consecutivos deste vetor.
- MDF: seleciona o formato (MDF:1, 2 ou 3 - imaginário, real ou complexo) do número será lido ou escrito na memória externa. Permite que apenas a parte real ou imaginária da RAM interna seja acessada. Útil para operar com vetores exclusivamente reais ou imaginários.
- MSS: define o número de pontos que separa o início de um bloco de pontos sequenciais, definido por MBS, do início do próximo bloco.
- NMPT: indica o número de pontos que vão ser afetados pela instrução.
- RS: especifica qual seção da RAM interna que será usada na instrução, caso ela esteja dividida em 2 seções.
- RV: indica se os dados serão lidos/escritos em sequência normal (RV:0) ou em sequência "bit-reversa" [BRI-75] (RV:1).
- SH: especifica se o resultado da operação aritmética corrente será dividido por dois para evitar *overflow* (SH:1).

Os parâmetros específicos de cada instrução serão discutidos juntamente com a instrução.

3.5.1. Instruções de I/O

LD move dados contidos na memória externa para a RAM de dados.

Usa: NMPT, RS, INTRP, EI, MBS, MSS, MDF e MBA.

ZR: especifica se, quando MDF seleciona um formato que não o complexo, a parte do número não envolvida na operação será zerada (ZR:1) ou não (ZR:0).

LDSM move dados da memória externa para a RAM de escala, ou para o Registro de Modo.

Usa: NMPT, RS, EI, MBA

MD: seleciona se os dados serão armazenados na RAM de Escala ou no Registro de Modo.

UP: se a instrução se refere a RAM de escala, especifica se os registros de escala são inicializados.

ST move dados contidos na RAM de dados para a memória externa.

Usa: NMPT, RS, EI, MBS, MSS, RV, MDF, MBA.

STB permite que os dados contidos na RAM de dados sejam armazenados na memória externa, mas com o ponteiro de endereços sendo decrementado a cada dado.

Usa: NMPT, RS, EI, MBA, MSS, RV, MDF

MBAB: aponta para o endereço na memória externa onde os dados vão começar a ser escritos.

3.5.2. Instruções Aritméticas de Um Operando

CMCN calcula o complexo conjugado dos dados contidos na RAM de dados.

Usa: NMPT, RS, ADF, EI

FFT calcula a FFT dos dados contidos na RAM de dados.

Usa: RS, ADF, EI

NMBT: especifica o número de operações de *butterfly* executada a cada passo da FFT.

R: especifica se a ordem dos dados é normal (R:0) ou bit-revertida (R:1). Em outras palavras, especifica se o algoritmo da FFT será de decimação em tempo ou em frequência.

FPS: define a separação dos pontos usados na *butterfly*. A cada passo da FFT este valor é multiplicado ou dividido por 2, dependendo do tipo de algoritmo escolhido.

LPS: determina o último passo da FFT através da separação dos pontos da *butterfly* neste passo.

RBA: define o endereço inicial da tabela de senos e cossenos a ser usado na FFT.

FSIZ: especifica o número de pontos contidos na FFT. Definindo-o com valor diferente do parâmetro NMBT permite-se que várias FFT sejam calculadas com uma única instrução.

AS: determina se haverá divisão automática por 2 a cada passo de FFT (AS:1) ou se será usada a aritmética de ponto flutuante por bloco (AS:0).

I: seleciona se a FFT será inversa (I:0) ou direta (I:1). Em outras palavras, especifica o sinal da parte imaginária da tabela de senos e cossenos.

SCL ajusta a escala do vetor armazenado na RAM de dados do VSP através de deslocamentos a esquerda ou a direita. O número de deslocamentos é determinado pelo conteúdo da RAM de Escala. Os parâmetros desta instrução permitem que blocos sucessivos do vetor sejam ajustados usando um fator de escala diferente.

Usa: NMPT RS, ADF, EI

SB: especifica se o fator de escala será o fator contido na RAM de Escala ou a diferença entre o Máximo Valor de Escala e o fator na RAM de Escala.

SCLVL: é o número de fatores de escala armazenados na RAM de Escala.

SCLBL: é o número de elementos sucessivos contidos na RAM de dados do VSP que serão ajustados pelo mesmo fator de escala.

3.5.3. Instruções Aritméticas de Dois Operandos

ADDC adiciona o vetor complexo armazenado na RAM de dados do VSP a um vetor complexo armazenado em determinado endereço da memória externa. O resultado é armazenado na RAM de dados.

Usa: NMPT, RS, EI, MBS, MSS, CN, SH, MBA

MLTC multiplica o vetor complexo armazenado na RAM de dados do VSP por um vetor complexo armazenado em determinado endereço da memória externa. O resultado é armazenado na RAM de dados. Calcula também a soma dos produtos e armazena-o no Acumulador do VSP.

Usa: NMPT, RS, EI, MBS, MSS, CN, SH, MBA

MLTR multiplica o vetor real armazenado na RAM de dados do VSP por um vetor real armazenado em determinado endereço da memória externa. O resultado é armazenado na RAM de dados. Calcula também a soma dos produtos e armazena-o no Acumulador do VSP.

Usa: NMPT, RS, EI, MBS, MSS, CN, SH, MBA

3.5.4. Instruções de Controle

HLT interrompe a busca de instruções pelo VSP, encerrando o programa.

NOP não executa operação alguma. Útil para gerar atrasos ou reservar espaço em um programa.

Usa: NMPT, RS, EI

4. Módulo de Aquisição

4.1. Introdução

O MAq é uma placa de interface analógica e digital para conexão a computadores da linha IBM-PC com barramento ISA de 16 bits (modelos AT em diante). Suas características principais são:

- Operação independente do hospedeiro
- Frequência máxima de aquisição de 200 KHz
- Memória local com capacidade para 8K palavras de 16 bits, permitindo expansão
- Polaridade selecionável por *strap*
- Aquisição comandada por evento externo ou por base de tempo interna
- Base de tempo para aquisição programada por software
- Início de aquisição disparado por sinal externo
- Detecção de perda de conversão
- Registro de até 4 sinais digitais durante a aquisição do sinal analógico
- Disponibilidade de 8 linhas digitais de entrada e 8 de saída
- Barramento de dados de 16 bits para maior velocidade de transferência de dados
- Mapeamento dos dispositivos em memória para acesso mais rápido

4.2. Arquitetura

A operação independente do MAq é possível graças a um Barramento de Dados Local, isolado do PC. Este barramento liga o conversor AD a memória local e cria um caminho exclusivo para os dados enquanto ocorre uma aquisição de blocos. Isso possibilita que o PC acesse os outros recursos do MAq mesmo durante um processo de aquisição. Quando se deseja ler a memória ou o conversor, este barramento é conectado ao barramento do PC. O Barramento de Endereços Local também contribui para a operação independente do módulo.

A arquitetura do MAq está na Fig 4.1. Por razões de clareza nele não estão representados os sinais de controle dos dispositivos.

Fig 4.1 - Arquitetura do Módulo de Aquisição

4.2.1. Barramento

O barramento ISA-16 bits do PC é fisicamente composto por dois conectores, o conector de 8 bits e o conector de expansão de 16 bits. Este acomoda os sinais adicionais de dados, endereços e controle acrescentados pelo uso de um microprocessador de 16 bits (Intel 80286), que substituiu um de 8 bits (Intel 8088) na arquitetura do PC-AT. O barramento de 8 bits manteve-se igual. A Fig 4.2 ilustra os conectores com os sinais dos barramentos.

Funcionalmente o barramento do PC é dividido em Barramento de Dados, de Endereços e de Controle. O Barramento de Dados possui 16 bits. O Barramento de Endereços possui 24 linhas de endereços, permitindo o acesso a até 16 MBytes de memória. O Barramento de Controle é composto por uma série de sinais que serão discutidos a seguir. Apenas os sinais usados neste trabalho serão abordados.

MEMR: sinal de saída ativo em baixo que sinaliza ciclo leitura de memória. Este sinal se encontra tanto no barramento de 8 bits quanto no de extensão de 16 bits e tem significados diferentes. No barramento de 8 bits ele só é ativo nos acessos a memória localizada em endereços abaixo de 1 MByte. Já no barramento de extensão ele é ativo em toda leitura de memória, independente do endereço. Foi usado o sinal do conector de 8 bits.

MEMW: sinal de saída ativo em baixo que sinaliza ciclo de escrita em memória. Como o sinal de MEMR, ele também se localiza nos dois conectores e tem o mesmo comportamento. Também foi usado o sinal do conector de 8 bits

- RESET:** sinal de saída ativo em alto que indica que o PC está em processo de inicialização. Pode ser usado para inicialização de dispositivos sequenciais na placa de expansão. Se encontra no conector de 8 bits.
- OSC:** sinal de saída presente no conector de 8 bits que disponibiliza um clock de 14.31818 MHz e ciclo de trabalho de 50%. Este sinal não é o clock do processador, e nem é sincronizado com ele. É usado pelo PC para geração de sinais de vídeo, e portanto, sua frequência é sempre fixa, não mudando de modelo para modelo de PC.
- MEM16:** sinal de entrada ativo em baixo que alerta o PC que o dispositivo que está sendo endereçado permite transferências de 16 bits. Estas transferências são realizadas com 1 *wait-state*. O sinal se encontra no conector de 16 bits, de modo que apenas uma placa de expansão de 16 bits pode ativá-lo. Sua ativação deve ser feita por uma saída *open-colector* ou *tri-state*.

Fig 4.2 - Sinais nos conectores do barramento ISA

4.2.2. Buffer de Barramento e Decodificação

As linhas de endereço, dados e controle do barramento usadas no MAq tiveram sua corrente de saída reforçada para aumentar o seu *fan-out*, que é de apenas uma porta TTL-LS. Sua decodificação foi projetada baseado no mapa de memória do PC. Ele indica que a memória livre do PC se estende de A0000h a EFFFFh, de forma não contínua, já que existem regiões em uso, como exibido na Fig 4.3.

Fig 4.3 - Utilização da memória RAM do IBM-PC

O MAq ocupa um espaço de 8 Kbytes na memória do PC, independentemente do tamanho da memória de aquisição. A decodificação gera 8 linhas de seleção, separadas por 1 KByte, para os dispositivos no MAq. Este espaço pode mapeado na memória do PC a partir de 80000h em intervalos de 8 Kbytes, até o endereço FFFFFh. A seleção do endereço base do MAq é feita por meio de *straps*.

A decodificação força que qualquer acesso aos dispositivos no MAq seja feito usando endereços pares. São possíveis acessos de bytes e words, ambos com duração de 1 ciclo de máquina. Se for usado um endereço ímpar (A0=1) o MAq não será habilitado. Isso foi feito para forçar o uso do modo de transferência de 16 bits mais rápido que o 80286 possui [INT-84].

4.2.3. Buffer de Memória

Este é um buffer bidirecional de 16 bits que faz o isolamento entre o barramento de dados do PC e o barramento de dados local.

4.2.4. Buffer do Conversor AD

Este é um buffer unidirecional de 16 bits que faz o isolamento do conversor AD e das 4 linhas digitais que são armazenadas durante aquisição automática dos dados.

4.2.5. Conversor AD

O conversor analógico-digital usado é o AD5240 de 12 bits. Faz a conversão pela técnica de aproximações sucessivas com tempo de conversão de 5 μ s [ANA-82]. Suas saídas não são multiplexadas, o que diminui o tempo de leitura do dispositivo e simplifica o projeto da máquina de estados de controle. Necessita de alimentação de +15V e -15V, fornecida por fonte de alimentação externa.

Este bloco possui apenas uma entrada analógica, visto que apenas um sinal será monitorado neste trabalho. Todos seus sinais digitais são plenamente compatíveis com as famílias TTL e DTL.

O circuito de *sample-hold* utilizado foi o AD582 [ANA-80]. Embora possua bom desempenho para aplicações gerais [TON-92], possui um tempo de aquisição (*acquisition time*) de 5 μ s, da mesma ordem de grandeza que o tempo de conversão do conversor AD, o que reduz a taxa de aquisição máxima do sistema. Foi adotado apenas para validar o módulo de aquisição. Um circuito de *sample-hold* com melhores características para alta velocidade é o HTS-0025 [ANA-80], que possui um tempo de aquisição de apenas 20 ns.

4.2.6. Memória Local

A memória usada foi a 6264 [NAT*-87], com estrutura interna de 8 Kbytes. É uma memória RAM estática com tempo de acesso de 70 ns. Suas linhas de dados e endereços não são multiplexadas. Foram utilizadas duas dessas memórias para formar um banco básico de 8K palavras de 16 bits. É possível expandir a memória local até 64K palavras, acrescentando mais blocos básicos de 8K palavras.

Todos os sinais de controle das memórias (RD, WR e CS) são gerados pela Lógica de Controle do MAq. As linhas de controle do PC não acessam as memórias diretamente. As linhas de endereço que vão ligadas às memórias são obtidas do Gerador de Endereços e permitem apenas o acesso seqüencial à memória. O PC enxerga a memória local como um único endereço, como indicado na Fig 4.4.

Fig 4.4 - Mapeamento da memória do MAq na memória do hospedeiro

4.2.7. Gerador de Endereços

O Gerador de Endereços é composto por 4 contadores síncronos de 4 bits (74LS163). Estes contadores possuem um tempo de estabilização de contagem de 30 ns [NAT-87], não importando a quantidade de contadores ligados em cascata. Eles possuem ainda 2 linhas de controle (INC e RST) que são acionadas pela Lógica de Controle. A cada leitura ou escrita nas memórias a Lógica de Controle ativa o sinal de incremento do Gerador de Endereços, e o endereço é incrementado, apontando para a próxima posição da memória local. O Gerador de Endereços é inicializado pelo sinal RST.

Dos 16 bits gerados pelos contadores, apenas os 13 bits menos significativos são oferecidos às memórias. Os 3 bits mais significativos são usados numa lógica de decodificação para gerarem 8 sinais de seleção. Cada sinal destes habilita um bloco de memória de 8K palavras.

4.2.8. Portas de Entrada e Saída (PIO)

Esta portas dispõe de 8 bits para leitura de sinais digitais externos e 8 bits para controle de dispositivos externos. Estas portas são implementadas usando o 8255 [INT-78], que possui 3 portas de 8 bits configuráveis como entrada ou saída. A porta A é configurada como saída, a porta B, como entrada. A metade menos significativa da porta C é configurada como saída e fornece os bits de controle do MAq. A outra metade pode ser programada como entrada ou saída, de acordo com a necessidade, uma vez que não está sendo utilizada pelo MAq.

4.2.9. Contadores (CTD)

É implementado usando-se um 8254 [INT-78] que possui 3 contadores independentes de 16 bits que operam com sinais de frequência de até 5 MHz. O contador 2 é programado como contador de eventos. Deve ser inicializado com o número de aquisições desejadas, para sinalizar o fim do processo de aquisição. Os dois outros contadores são programados como divisores,

usados para gerar a base de tempo de aquisição a partir da base de tempo obtida no barramento do PC.

4.2.10. Registro de Controle

São 4 bits de saída da porta C do 8255 que controlam alguns dispositivos do MAq. Os bits têm as seguintes funções:

Fig 4.5 - Registro de Controle

- BIT_MD0 bit menos significativo do controle de modo de operação.
- BIT_MD1 bit mais significativo do controle de modo de operação.
- RST_CTD ativo em baixo, inicializa o Gerador de Endereços.
- HAB_CTD ativo em alto, habilita a operação dos 3 contadores no bloco de Contadores.

4.2.11. Registro de Status

São 3 bits de uma porta de entrada independente que indicam o status de alguns dispositivos do MAq. Os bits têm as seguintes funções:

Fig 4.6 - Registro de Status

- FIM_CV ativo em alto, sinaliza o fim do processo de aquisição quando programado o Modo de Aquisição Automática.
- SOBRE_CV bit de sobreconversão, ativo em alto. Indica que houve um pedido de conversão no Modo de Aquisição Automática que não pode ser atendido. Útil quando a aquisição é feita comandada por um evento externo que não possua uma frequência definida. Este sinal é inicializado quando o modo de operação programado não é o Modo de Aquisição Automática.

STS_ADC bit de status do conversor, ativo em alto. Indica que uma conversão está em progresso. Usado no Modo de Aquisição Individual para este monitorar o estado do conversor AD.

4.3. Modos de Operação

O MAq possui 4 modos de operação que permitem aquisição individual ou automática, imediatamente ou após um evento. O modo de operação é selecionado no Registro de Controle.

4.3.1. Aquisição Individual

Este modo permite que o hospedeiro comande uma conversão diretamente. O resultado da conversão deve ser lido após decorrido o tempo de conversão, que pode ser determinado através da leitura do Registro de Status. Este modo não permite o acesso a memória local.

4.3.2. Acesso a Memória Local

Transfere o controle da memória local ao hospedeiro, permitindo sua leitura ou escrita sequencial. Antes de cada leitura/escrita de um bloco de dados deve-se inicializar o apontador da memória local. Este modo não permite o controle ou acesso ao conversor AD.

4.3.3. Aquisição Automática

Este modo faz a aquisição independente de um vetor de dados. O tamanho do vetor deve ser pré-programado, e a taxa de aquisição, pré-definida. A aquisição se faz automaticamente, sem controle do hospedeiro, e os seus resultados são armazenados na memória local sequencialmente. Em paralelo com a aquisição do sinal analógico é feito um registro de 4 eventos digitais. O PC pode acompanhar o processo de aquisição pela leitura do contador de aquisições efetuadas ou pelo sinal de Fim de Conversão no Registro de Status. A aquisição se encerra quando for lido o número de pontos programado. Neste modo o PC não pode controlar nem o conversor AD nem acessar a memória local.

4.3.4. Aquisição Automática Disparada por Evento

Neste modo o MAq aguarda um evento para iniciar a aquisição automática. Após a chegada deste evento o comportamento do MAq é idêntico ao Modo de Aquisição Automática.

4.4. Lógica de Controle

A máquina seqüencial que faz o controle dos dispositivos do MAq é uma máquina de Moore [BOO-67] síncrona e pode ser representada pela Fig 4.7. Esta representação difere da representação clássica de uma máquina seqüencial por possuir um bloco de multiplex, mas ainda é uma máquina de Moore, já que os sinais de saída são função do estado da máquina e dos sinais de entrada.

Fig 4.7 - Diagrama de blocos da máquina seqüencial

4.4.1. Sinais de Saída

A lógica de controle deve coordenar as atividades do MAq, em todos os modos de operação, através dos sinais de controle listados na Tab 4.1.

Tab. 4.1 - Sinais de controle

Nome	Relativo ao Bloco	Ativo	Descrição
INCC	Gerador de Endereço		Incrementa o endereço da Memória Local
CSBM	Buffer de Memória	Baixo	Habilita o Buffer de Memória
CSBA	Buffer do Conversor AD	Baixo	Habilita o Buffer do Conversor AD
CSMEM	Memória Local	Baixo	Habilita um dos conjuntos da Memória Local
CV	Conversor AD		Inicia uma conversão
SOBRCV	Registro de Status	Alto	Indica uma tentativa de sobre-conversão
WEMEM	Memória Local	Baixo	Seleciona a a escrita de dados nas memórias

4.4.2. Sinais de Entrada

A lógica de controle gera os sinais de saída baseado em alguns sinais de entrada. Parte destes sinais é usada na Lógica Combinacional, e outra parte no Multiplex. Os sinais relativos às variáveis de estado não serão incluídos:

4.4.2.1. Multiplex

Tab. 4.2 - Sinais de saída do bloco Multiplex

Nome	Relativo ao Bloco	Ativo	Descrição
STSA	Conversor AD	Alto	Indica que o conversor está em conversão
RD	Buffer e Decodificação	Baixo	Indica que o PC está em um ciclo de leitura
WR	Buffer e Decodificação	Baixo	Indica que o PC está em um ciclo de escrita
REQCV	Externo	$\overline{1}$	Solicita uma conversão
CSAM	Buffer e Decodificação	Baixo	Requer acesso ao conversor ou à memória.

4.4.2.2. Lógica Combinacional

Tab. 4.3 - Sinais de saída do bloco Combinacional

Nome	Relativo ao Bloco	Ativo	Descrição
STSA	Conversor AD	Alto	Indica que o conversor está em conversão
FIMC	Contadores	Alto	Sinaliza o fim do processo de aquisição de dados
MD1	Registro de Controle	-	Bit mais significativo do modo de operação
MD0	Registro de Controle	-	Bit menos significativo do modo de operação
RST	Buffer e Decodificação	Alto	Coloca a máquina de estados no estado inicial
TG	Externo	Alto	Dispara o processo de aquisição de dados

4.4.3. Modos de Operação

O comportamento dos sinais de saída varia de acordo com o modo de operação do MAq. Logo tais sinais devem ser analisados em função do modo de operação.

4.4.3.1. Modo 0

O PC assume o controle do conversor. Os sinais de controle do conversor devem estar sincronizados com os do PC. A conversão é disparada pela escrita no endereço do conversor e o seu resultado é lido neste mesmo endereço. Neste modo não há pedidos de conversão externos, nem é possível a leitura da memória. Possui apenas um estado.

4.4.3.2. Modo 1

O PC assume o controle da memória local. Os sinais de controle da memória local devem estar sincronizados com os do PC. A escrita ou leitura no endereço que a memória local está mapeada, que é o mesmo do conversor AD, deve escrever ou ler a posição corrente da memória

local. O Gerador de Endereços deve ser incrementado, apontando para a próxima posição da memória. Neste modo o conversor não pode ser acessado e estão desabilitados todos os sinais relativos a ele. Possui apenas um estado.

4.4.3.3. Modo 2

O MAq passa a executar o processo de aquisição de dados. Os sinais de controle do conversor, memória e gerador de endereços não são mais sincronizados com o PC, mas passam a ser gerados por meio de uma máquina seqüencial. Neste modo a aquisição pode ser imediata ou condicional. A aquisição condicional ignora todas as solicitações de conversão enquanto não ocorrer o disparo da aquisição por um sinal externo, enquanto a imediata inicia a aquisição logo na primeira solicitação de conversão. Esta máquina implementa o algoritmo de aquisição ilustrado na Fig 4.8, onde as tarefas já estão dividida e associadas aos estados da máquina. Estes estados serão tratados adiante

4.4.3.4. Modo 3

Especifica que a aquisição no modo de operação 2 será condicional, disparada por sinal externo. Possui apenas 1 estado.

4.4.4. Variáveis de Estados

Para os modos de operação 0, 1, e 3 existe um estado associado. Para o modo 2 existem 5 estados. Ao todo são 8 estados, o que exige 3 variáveis de estado, VE2, VE1 e VE0. A Tab 4.4 apresenta os valores das variáveis de estado.

Tab. 4.4 - Estados associados aos modos de operação

Modo	Estado	VE2 VE1 VE0
0	S00	000
1	S10	001
2	S20	011
	S21	110
	S22	010
	S23	100
	S24	111
3	S30	101

Os valores das variáveis de estado foram escolhidos de modo que apenas uma variável se modifique em uma transição de estado válida. Isso reduz indeterminações na lógica combinacional e evita o aparecimento de transientes nos sinais de saída.

Fig 4.8 - Algoritmo de aquisição dos dados

4.4.5. Diagrama de Estados

O diagrama de estados representa os estados, as transições válidas entre estados e as condições para a transição. Ele está na Fig 4.9.

Fig 4.9 - Diagrama de estados da máquina sequencial do MAq

4.4.6. Implementação

4.4.6.1. Lógica Sequencial e Combinacional

Foi implementada através de uma PAL 16R6 [BIR-81], que possui um bloco combinacional para a determinação do próximo estado, e 6 flip-flops tipo D, sincronizados com um sinal de clock único. Esta PAL determina apenas o próximo estado, não gerando nenhum sinal de controle propriamente dito. O sinal de clock que sincroniza este dispositivo tem um período de aproximadamente 140 ns e é gerado pela divisão do sinal OSC do barramento do PC. Este período foi escolhido em função do tempo de acesso às memórias, com uma margem de segurança.

4.4.6.2. Multiplex

Foi implementado através de uma PAL 16L8 [BIR-81], que possui apenas circuitos combinacionais. Este bloco é usado para selecionar uma determinada função booleana para um sinal de saída em função do estado corrente do circuito. Isto permite que os sinais que controlam as memórias e o conversor AD possam ser sincronizados com os sinais de controle do PC quando necessário (modos de operação 0 e 1). Quando estes sinais de controle dependem apenas das variáveis de estado (modo 2 e 3), as saídas do multiplex passam a ser fixas para cada estado, e não dependem mais dos sinais do PC. A Fig 4.10 ilustra o explicado.

Fig 4.10 - Exemplo da utilização do bloco de multiplex

4.5. Detalhes Operacionais

Por ser mapeado em memória, este módulo exige que algumas condições sejam satisfeitas pelo hospedeiro. Como acessos à memória, são feitos mais rapidamente que acessos a dispositivos de IO, a velocidade de operação do hospedeiro deve ser observada. Se ele possuir

um clock de barramento maior que 12 Mhz, ele deve ser configurado de modo a não ultrapassar este limite. O clock do barramento pode ser definido pelo *setup* na maioria dos PCs. Aqueles que não possuem este parâmetro definível provavelmente não são velozes o suficiente para prejudicar o acesso ao Maq. Deve-se ressaltar que o que causa problema é a velocidade de acesso ao barramento, e não a velocidade do processador, dois parâmetros independentes nos Pcs.

Outro detalhe é a existência da *shadow-ram* mapeada nos mesmos endereços do Maq. Se isto acontecer, haverá conflito de endereçamento e a operação do módulo não será correta. Pelo *setup* do hospedeiro é possível desabilitar estas memórias nos endereços ocupados pelo MAq.

Finalmente, deve-se excluir a região de memória usada pelo MAq do gerenciamento da memória estendida/expandida, geralmente feito pelo programa EMM386.EXE do DOS. Isto se faz usando a opção /X na linha que invoca este programa no arquivo de configuração CONFIG.SYS. Supondo que o MAq esteja ocupando a região 0xD000:0x0000 a 0xD000:0xFFFF, esta linha deve ficar assim:

```
DEVICE=C:\DOS\EMM386.EXE /X=D000-E000
```

5. Módulo de Processamento

5.1. Introdução

Não foi considerado interessante desenvolver o hardware do módulo de processamento, pois ele já foi desenvolvido pela mesma empresa que produz o VSP, visando o teste de algoritmos em um ambiente de operação real, e também porque já existem bons trabalhos nesta área [COS-86] [RIE-85].

Esta placa é compatível com o barramento ISA de 16 bits. Possui um processador ZORAN ZR34161, uma memória local de 8K palavras de 16 bits de largura, que é usada para armazenar tanto programas quanto os dados. Possui também a lógica de controle que permite ao PC compartilhar a memória da placa e os registros do VSP, e supervisionar a operação do sistema.

Neste módulo apenas foi desenvolvido o algoritmo de processamento para o VSP e rotinas para o hospedeiro controlar a placa de desenvolvimento.

5.2. Algoritmo de FFT

5.2.1. Introdução

A implementação de algoritmos de processamento de sinais usando apenas as instruções do VSP é uma tarefa complexa. Os programas devem ser escritos "por extenso" já que, como visto no capítulo 3, o VSP não possui quaisquer instruções de controle de fluxo de programa. Apesar da natureza vetorial das instruções aritméticas, um algoritmo que processe um vetor de tamanho maior que 128 pontos exigiria várias instruções vetoriais. Outra dificuldade apresentada pela programação linear é a necessidade de se fazer vários programas para um mesmo algoritmo, um para cada tamanho de vetor desejado.

Uma solução para este problema é usar a técnica de geração automática de programas, também conhecida como *autogen* [MOR-77]. Esta técnica foi proposta para tornar certos programas de processamento de sinais mais eficientes em termos de tempo de execução, usando funções básicas (de *kernel*) executadas sequencialmente. Isto torna um programa mais eficiente pois não são necessárias avaliações de condição ou desvios no fluxo durante a execução do programa. Em contrapartida aumenta o tamanho do programa, que passa a depender do tamanho do vetor a ser processado.

A técnica *autogen* apenas pode ser usada em algoritmos cuja execução seja independente dos valores do sinal de entrada. Os algoritmos de FFT satisfazem essa condição.

As instruções vetoriais do VSP se prestam muito bem para o papel de funções de *kernel* que a técnica *autogen* exige. O programa para o VSP é gerado por meio de outro programa (o

programa gerador), escrito em linguagem de alto nível (PASCAL, neste trabalho) que supre as estruturas de controle e repetição que a linguagem do VSP não possui, além de suportar variáveis e instruções aritméticas escalares em tempo de compilação. O programa gerador pode ser parametrizado para produzir programas que processem vetores de vários tamanhos.

O programa gerador produz como saída um arquivo texto com o programa para o VSP usando as instruções da linguagem montadora do VSP, contendo inclusive comentários. Este programa deve ser compilado com o programa montador (*assembler*) [ZOR*-87] e então pode ser executado pelo VSP. Esta solução em duas etapas foi adotada visando maior flexibilidade e facilidade para a compreensão do algoritmo, já que o arquivo de texto torna possível efetuar alterações no mesmo e simplifica a interpretação e correção do algoritmo.

5.2.2. Escolha do Algoritmo

Para os algoritmos analisados no capítulo 2 deve ser feita uma análise de eficiência na sua implementação usando os comandos do VSP. O VSP possui uma arquitetura otimizada para o cálculo de *butterflies* radix-2, logo, quanto maior for o uso das instruções de FFT, maior será a eficiência da implementação. Por outro lado, alguns algoritmos de FFT relativamente mais eficientes, como o de Fatores Primos e o de Winograd não podem ser usados por envolverem operações de *butterfly* em vetores com tamanhos primos.

Dos algoritmos tratados anteriormente, o de Cooley e Tukey e o de Gold e Bially são os que melhor podem ser implementados no VSP.

5.2.2.1. Eficiência de Implementação

Supondo um algoritmo para processar um vetor com N pontos, onde N satisfaz a equação (2.4). A eficiência da implementação pode ser determinada pelo tempo gasto no processamento deste vetor, e este tempo pode ser calculado considerando o número de operações de FFT e multiplicações complexas, que são as operações que exigem maior esforço computacional no algoritmo. O número de ciclos de máquina exigidos para a execução destas instruções depende do tamanho do vetor que é processado. O número de ciclos exigidos para algumas operações [ZOR-86] está na Tab 5.1.

Tab. 5.1 - Duração das instruções MLTC e FFT

N	MLTC	FFT
128	527	1850
64	271	818
32	143	362
16	79	162
8	47	74
4	31	34
2	23	14

Sabendo-se que a duração de um ciclo de máquina é de 100 ns, e o número de operações exigida por cada algoritmo, pode-se estimar o tempo de execução dos dois algoritmos para vários valores de N .

$$\text{Cooley e Tukey: } O_{CT}(N) = M.\text{FFT}(L) + N.\text{MUL}(L)$$

$$\text{Gold e Bially: } O_{GB}(N) = M.\text{FFT}(L) + M.\text{MUL}(L) + L.\text{FFT}(M)$$

Tab. 5.2 - Velocidade de processamento dos algoritmos de CT e de GB

N	M	L	CT (ms)	GB (ms)
256	16	16	2,28	0,64
512	16	32	4,62	1,33
1024	32	32	15,80	2,77
2048	32	64	31,90	5,80
4096	64	64	116,24	12,20

O gráfico apresentado na Fig 5.1 torna mais clara a diferença de velocidade entre os dois algoritmos.

Fig 5.1 - Velocidade de processamento dos algoritmos de CT e de GB

Esta diferença mostra claramente que o algoritmo de Gold e Bially pode ser implementado mais eficientemente no VSP do que o de Cooley e Tukey. Os tempos de execução calculados são o limite mínimo teórico para cada algoritmo, já que não foram consideradas as transferências de dados do/para o VSP, nem operações aritméticas menos relevantes.

5.2.3. Implementação

5.2.3.1. Introdução

O programa implementado no VSP efetua outras tarefas além da FFT em um vetor de pontos. Inicialmente o programa faz a apodização do vetor de entrada. Depois é feita a FFT propriamente dita. Em seguida se faz a composição da FFT complexa para obter a FFT do vetor de entrada. Finalmente é feito o cálculo do módulo do resultado.

A seguir serão descritas estas etapas de cálculo. A representação gráfica que acompanha cada etapa ilustra a sequência de instruções, os operandos usados por elas e a RAM interna do VSP. Os resultados das operações são armazenados nesta memória. Os operandos utilizados são hachurados. Os parâmetros mais comuns de cada instrução são agrupados a margem do desenho, e quando uma instrução se utiliza de parâmetros diferentes, estes acompanham as instruções. Os operandos estão armazenados na memória externa do VSP e são considerados variáveis.

5.2.3.2. Mapeamento de memória do VSP

O uso da memória do VSP é apresentado na Tab 5.3. Esta tabela adota as seguintes convenções:

- \$ significa o primeiro endereço livre depois daqueles ocupados pela variável definida anteriormente
- @ acompanhado de um identificador de variável significa o endereço de início daquela variável

Tab. 5.3 - Variáveis utilizadas pelo programa de FFT

Variável	Estrutura	Tipo	Início	Tamanho Estrut.	Espaço Ocup.
Dado	matriz	real	0x0000	2L linhas × 64 colunas	2L * 64
TabApod	matriz	real	\$	2L linhas × 64 colunas	2L * 64
H	matriz	complexo	@DadoIni	L linhas × 64 colunas	L * 64 * 2
G	matriz	complexo	\$+2	L linhas × 64 colunas	L * 64 * 2
Trans	matriz	complexo	\$+2	L linhas × 64 colunas	L * 64 * 2
Área de Programa	---	---	0x300C	---	ver Tab 6.1
ModosOp	---	---	\$	---	6
CteNeg	escalar	complexo	\$	---	2
CteMulC	escalar	complexo	\$	---	2
RamEscH	vetor	real	\$	16 elementos	16
RamEscV	vetor	real	\$	16 elementos	16
TabResC	vetor	complexo	\$	64 elementos	16 * 2
TabMulC	vetor	complexo	\$	64 elementos	16 * 2
TabResF	vetor	complexo	\$	64 elementos	16 * 2
TabReff	vetor	complexo	\$	64 elementos	16 * 2
TabMulF	vetor	complexo	\$	64 elementos	16 * 2

5.2.3.3. Apodização

A apodização é feita multiplicando-se o vetor real de entrada pela Tabela de Apodização. Esta tabela é carregada juntamente com o vetor, permitindo o uso de várias funções de apodização sem alteração do programa de FFT. Esta etapa opera com vetores de 128 elementos, ao invés dos normais de 64 elementos, para reduzir o número de instruções e diminuir seu tamanho.

Fig 5.2 - Apodização

5.2.3.4. FFT horizontal

Esta etapa efetua os cálculos de FFT sobre os L vetores horizontais da matriz gerada pelo mapeamento do vetor de dados inicial, como descrito no capítulo 2. A cada 4 instruções de FFT os 4 valores de escala são armazenados numa região de memória externa denominado **RamEscH**. Os resultados da FFT são armazenados na mesma posição dos vetores lidos.

Fig 5.3 - FFT horizontal

5.2.3.5. Multiplicação, ajuste de escala e FFT vertical

Esta etapa efetua 3 operações. Estas operações foram agrupadas para maior eficiência, pois desta forma se evita as várias transferências de dados entre a memória externa e a RAM do VSP. Inicialmente se faz a multiplicação das colunas da matriz **Dado** pelo vetor **TabMulC**, como descrito no capítulo 2. Em seguida será feita o ajuste de escala dos dados usando os fatores armazenados em **RamEscH**. Após isso será feita a FFT nos vetores verticais, cujos valores de escala são armazenados desta vez em **RamEscV**. Se os vetores verticais possuírem menos que 64 elementos, mais de um vetor será carregado na RAM do VSP e a instrução de FFT calculará a FFT destes vetores de uma só vez. Finalmente o vetor de multiplicação **TabMulF** será atualizado, através da sua multiplicação pelo vetor **TabReffF**. Estas atualizações sucessivas do vetor **TabMulC** dispensam o armazenamento de uma matriz completa com os valores de ponderação na memória externa do VSP.

Fig 5.4 - Multiplicação, ajuste de escala e FFT vertical

5.2.3.6. Ajuste de escala final

Esta etapa ajusta a escala dos vetores verticais após o cálculo de FFT sobre eles. Na verdade os vetores carregados são os horizontais, e os parâmetros da instrução SCL asseguram o ajuste de escala coerente. Os vetores já ajustados são armazenados na matriz **Trans**, sendo que a metade inicial é armazenada na sequência normal, e a metade final, na sequência invertida. Isto é feito para simplificar os cálculos posteriores.

Fig 5.5 - Ajuste de escala final

5.2.3.7. Determinação dos vetores H e G - 1ª parte

Nesta etapa se determina a parte real de **H** e a parte real de **G** segundo (2.21). Nestes cálculos as operações são feitas apenas para metade dos vetores que compõe a matriz **Trans**. A outra metade é obtida usando as propriedades de simetria da FFT sobre vetores reais e imaginários apresentadas em (2.20).

Fig 5.6 - Determinação dos vetores H e G - 1ª parte

5.2.3.8. Determinação dos vetores H e G - 2ª parte

Nesta etapa determinam-se as partes imaginárias dos vetores **G** e **H**. Usa-se o aproveitamento de simetria da FFT para economia de cálculos.

Fig 5.7 - Determinação dos vetores H e G - 2ª parte

5.2.3.9. Composição dos vetores H e G

Após determinar-se a FFT dos vetores **H** e **G**, deve-se compô-los segundo (2.22) para obter o resultado final. A composição é feita multiplicando-se o vetor **G** pelo vetor **TabMulC**. Este resultado é somado ao vetor **H**, e o resultado é armazenado em **H**. Após isto o vetor **TabMulC** é atualizado pela sua multiplicação com a constante **CteMulC**. Isto evitou que se tivesse que armazenar na memória externa do VSP toda a matriz de composição.

Fig 5.8 - Composição dos vetores H e G

5.2.3.10. Cálculo do módulo do vetor

Finalmente é feito o cálculo do quadrado do módulo do resultado, sendo armazenado em outra área. Dessa maneira pode-se acessar o resultado de duas formas diferentes.

Fig 5.9 - Cálculo do módulo do vetor

5.3. Sistema de Desenvolvimento

5.3.1. Comunicação Hospedeiro-VSP

A placa de desenvolvimento do VSP possui uma memória local, que é compartilhada pelo VSP e pelo hospedeiro. Esta memória armazena os dados a serem processados e o programa de processamento. Possui 16K palavras, uma largura de palavra de 16 bits, e seu mapeamento no hospedeiro é determinado por meio de *straps* na placa [ZOR-88]. As tarefas básicas necessárias para o uso do VSP são:

- Colocar a memória do VSP disponível para o hospedeiro
- Escrever os dados na memória do VSP
- Carregar o programa na memória do VSP
- Passar o endereço de execução para o VSP e liberar a sua memória
- Esperar o fim do processamento
- Colocar a memória do VSP disponível para o hospedeiro
- Ler o resultado na memória do VSP

Estas tarefas foram encapsuladas em funções e procedimentos apresentados no Apêndice D. Os subsídios para a implementação destas funções foram obtidos em [ZOR-88].

6. Resultados

Todos os teste a seguir foram feitos usando como hospedeiro um computador 486 DX2 de 50 MHz. Ele foi configurado de acordo com o descrito no capítulo 4.

6.1. Módulo de Aquisição

O teste mais completo que pode ser feito no MAq é uma aquisição de um sinal analógico, feita na frequência especificada no projeto (100 KHz) e usando toda sua memória (8K palavras). Este teste envolve a operação do conversor AD, a leitura da memória interna, a aquisição automática de um sinal analógico e o disparo da conversão por meio de um sinal externo. Ou seja, todos os modos de operação do módulo e todos os blocos funcionais.

O sinal lido pelo MAq foi um sinal senoidal de 20 Hz, com uma amplitude de 4 V. Para este teste o MAq estava configurado para aceitar sinais bipolares e com fundo de escala de 5 V.

O sinal de disparo de conversão foi providenciado por um circuito adicional, montado separadamente e apresentado no Apêndice H. A aquisição é iniciada no momento que o sinal a ser digitalizado atinge determinado valor, podendo-se inclusive selecionar se o disparo será feito na subida ou na descida do sinal. Para o teste o nível da tensão de disparo selecionado foi de 0 V, e o disparo feito na descida do sinal. Este circuito deve ser aproveitado no interferômetro para cumprir a função de determinar o momento de início de leitura do interferograma, como foi discutido no Capítulo 1.

O resultado da aquisição se encontra na Fig 6.1, e o programa que implementou este teste está no Apêndice I.

Fig. 6.1 - Resultado da aquisição de senoide de 20 Hz e 4 V de amplitude

6.1.1. Precisão na Conversão

A precisão na conversão pode ser comprovada pelos valores do eixo Y do gráfico da Fig 6.1. Os valores de pico do sinal lido são de 3672 e 449 para os valores máximo e mínimo, o que representa respectivamente 4 V negativos e positivos [ANA-82].

6.1.2. Análise da Taxa de Aquisição

A taxa de aquisição pode ser comprovada pela análise do eixo X do gráfico da Fig 6.1. Sabendo-se que o sinal lido tem um período de 50 ms e que o intervalo de aquisição é de 10 μ s, um período do sinal deve estar se completando por volta do ponto número 5000, como comprova o gráfico. As imprecisões são devidas a erros de calibração do gerador de sinais e a uma taxa de aquisição (99,4 KHz) ligeiramente diferente da desejada, pois a frequência da base de tempo de aquisição não é um múltiplo exato da frequência de aquisição desejada.

6.2. Módulo de Processamento

Os resultados aqui apresentados são de natureza qualitativa e quantitativa. Para análise qualitativa é apresentado o gráfico do resultado da FFT de um pulso, cuja transformada de Fourier é bastante conhecida, calculada pelo VSP usando o algoritmo descrito anteriormente. Para a análise quantitativa da precisão deste algoritmo em diversas classes de sinais, é apresentado um estudo dos erros de cálculo da FFT em sinais aleatórios.

6.2.1. Desempenho

A medida de desempenho foi feita no algoritmo como um todo e nas suas tarefas básicas. Foi medido também o tamanho de cada programa, excluindo as estruturas de dados auxiliares, apresentadas na Tab 5.3. Os tempos estão expressos em milisegundos e os tamanhos em número de palavras.

A medida exata dos tempos foi feita no programa simulador do VSP [ZOR*-88], uma vez que a sua execução concorrente de instruções dificulta o cálculo destes tempos. Para validar este método de medida foi usado o MAq executando o programa mais demorado (2048 pontos, algoritmo completo), onde o tempo de execução foi aferido por meio do relógio do hospedeiro. Como sua precisão é limitada a 55 ms, os resultados destas medidas, apesar de não serem exatamente iguais aos obtidos pelo simulador, comprovaram-no. O tamanho do programa é fornecido pelo programa montador do VSP.

Foram medidos os tempo de processamento de 3 etapas do algoritmo: apenas do cálculo da FFT de um vetor complexo de N pontos; do cálculo da FFT no vetor real com $2N$ pontos, que

envolve a FFT complexa, a determinação dos vetores **H** e **G** e a composição dos mesmos (ver capítulo 5); e o tempo de processamento total do algoritmo. Os resultados estão na Tab 6.1.

Tab. 6.1 - Desempenho e tamanho dos módulos do programa de FFT

Pontos	FFT complexa		FFT real (2N)		Algoritmo (2N)	
	Tempo	Tamanho	Tempo	Tamanho	Tempo	Tamanho
2048	9,92	1393	16,86	2778	19,64	3190
1024	4,76	717	8,22	1414	9,62	1626
512	2,29	379	4,02	732	4,72	844

6.2.2. FFT de um Pulso

Foi escolhido fazer a FFT de um pulso graças ao seu espectro rico em harmônicas e bastante conhecido. Para não se alterar este espectro foi escolhida a função retângulo como função de apodização. A figura Fig 6.2 apresenta o resultado, e o programa que implementou este teste está no Apêndice J.

O pulso escolhido possui uma largura de 8 pontos, com a máxima amplitude aceita pelo VSP (0x7FFF). O vetor é real e possui um tamanho de 4096 pontos, e a sua transformada é um vetor complexo com 2048 pontos.

Fig. 6.2 - Transformada de um pulso de 8 pontos de largura

6.2.3. Análise de Erros

Este teste avalia a precisão do algoritmo implementado no VSP comparando seu resultado com o resultado produzido por um programa equivalente implementado no hospedeiro, como mostra a Fig 6.3.

A apodização usada é a função retângulo, pois as outras funções aqui discutidas podem alterar substancialmente os resultados se os sinais a serem processados têm a maior parte de sua informação nos extremos da janela, e quando se usa a aritmética de ponto fixo.

Fig. 6.3 - Análise de erros do algoritmo de FFT

O algoritmo de FFT implementado no hospedeiro é o de Cooley e Tukey radix-2 com decimação no tempo [BRI-74] e usa aritmética de ponto flutuante nos seus cálculos. A representação do tipo numérico usado é de 5 bytes para mantissa e 1 para expoente. Deve-se ressaltar que além do algoritmo de FFT no hospedeiro ser diferente, ele também processa um vetor complexo como entrada, ao invés de um vetor real com a metade do tamanho.

Para se abranger todos os tipo de funções neste teste, foi escolhido aplicar ao VSP sinais aleatórios, cuja amplitude varia dentro dos extremos permitidos pelo VSP. O processo de teste é automatizado para a geração de um número significativo de dados para análise. Os resultados são armazenados em arquivo de registro.

Foram feitas análises para 50 vetores de teste de 1024 elementos e o maior erro encontrado foi de 0,1% do valor máximo. Isto comprova a precisão do algoritmo e da aritmética do ponto flutuante em bloco.

7. Conclusões

Este trabalho produziu um sistema computacional que, embora fosse orientado para automação de um espectrômetro de varredura refrativa, é flexível o bastante para ser empregado em uma ampla gama de aplicações. Esta preocupação foi um fator dominante em todo o processo de desenvolvimento do sistema.

O módulo de aquisição é simples e de baixo custo. Seu desempenho é, em média, 5 vezes superior ao dos sistemas de aquisição tradicionais, controlados pelo hospedeiro, cuja frequência de aquisição é de, no máximo, 40 KHz. Sua operação independente torna-o interessante para a análise de sinais transientes com tempos de relaxação da ordem de milisegundos, que exigem sistemas de aquisição muito mais caros. Entre os sinais que estão nesta categoria estão sinais de áudio e vibrações mecânicas, o que garante ampla utilização deste módulo. A utilização deste módulo se torna simples devido à biblioteca de rotinas de controle desenvolvida neste trabalho. Estas rotinas, escritas em Pascal, escondem os detalhes de hardware e simplificam a sua operação, como pode ser comprovado nos programas de teste apresentados nos Apêndices.

O módulo de processamento aproveitou uma placa de desenvolvimento para o VSP e deu-lhe uma aplicação específica. Neste módulo o trabalho se concentrou na programação do VSP. Para tanto foram analisados vários algoritmos de FFT, e estudados em profundidade os dois que melhor se prestavam para a implementação vetorial. Foi apresentada uma técnica de programação que simplifica o desenvolvimento de programas para o VSP.

O conjunto formado pelo módulo de aquisição, módulo de processamento e computador hospedeiro pode ser interpretado como um *pipe-line* vetorial, já que todos os elementos podem operar concorrentemente. As tarefas de aquisição, processamento e interpretação, normalmente executadas nesta sequência, podem ser executadas com máxima eficiência por cada um dos elementos do *pipe-line*.

Como proposta de desenvolvimento futuro, é recomendada a substituição do circuito de *sample-hold* utilizado no módulo de aquisição por um com tempo de aquisição (T_{SH}) menor, de modo a aproximar frequência de aquisição (F_{AQ}) do seu valor máximo teórico de 200 KHz. O uso de um conversor AD com menor tempo de conversão (T_{CV}) também traria grande melhora no desempenho, pois neste sistema de aquisição o tempo gasto nas tarefas de controle da aquisição (T_{CTRL}) é desprezível. Se for necessário, o uso de memórias de menor tempo de acesso (T_{MEM}) pode reduzir ainda mais o tempo gasto no controle de aquisição. As vantagens advindas da substituição dos componentes do MAq pode ser avaliada pela seguinte expressão:

$$T_{AQ} = T_{SH} + T_{CV} + T_{CTRL}$$

$$F_{AQ} = \frac{1}{T_{AQ}}$$

Os tempos dos componentes utilizados são:

$$T_{CTRL} = 3 \cdot T_{MEM}$$

$$T_{CV} = 5\mu s$$

$$T_{SH} = 5\mu s$$

$$T_{MEM} = 70 ns$$

Quanto ao módulo de processamento, sugere-se o aumento da memória instalada através do acréscimo de um banco de memórias em uma placa-filha substituindo as memórias já instaladas. Como o controle do sistema se baseia em PAL, é simples alterar os mapeamentos das memórias. Outra sugestão seria a elaboração de um algoritmo simples de extração de paralelismo que analisasse o programa do VSP e alterasse a ordem das suas instruções, quando possível, de modo a aproveitar a capacidade de execução paralela da BIU e da EU. Atualmente esta tarefa fica a cargo do programador.

Referências Bibliográficas

- [ANA-80] ANALOG DEVICES, INC., **Data Acquisition Components and Subsystems Catalog**, Massachusetts, 1980
- [ANA-82] ANALOG DEVICES, INC., **Data Acquisition Databook - Integrated Circuits**, Massachusetts, 1982
- [BEL-72] BELL, R. J., **Introductory Fourier Transform Spectroscopy**, New York, Academic Press, 1972
- [BIR-81] BIRKNER, J. & COLI, V., **PAL Handbook**, Monolithic Memories Inc., 1981
- [BLA-85] BLAHUT, R. E., **Fast Algorithms for Digital Signal Processing**, Addison-Wesley, Reading, 1985
- [BOO-67] BOOTH, T. L., **Sequential Machines and Automata Theory**, New York, John Wiley and Sons, 1967
- [BRI-74] BRIGHAM, E. O., **The Fast Fourier Transform**, New Jersey, Prentice-Hall, 1974
- [COO-66] COOLEY, J. W. & TUKEY, J. W. An Algorithm for the Machine Calculation of Complex Fourier Series, **Mathematics of Computation**, Vol. 19, No. 90, pp. 297-301, 1965
- [COS-86] COSTA, L. F., **Arquitetura Dedicada Para o Cálculo da Transformada de Fourier, Tese de Mestrado**, Instituto de Física e Química de São Carlos, 1986
- [DYE-88] DYER, S. A. & MORRIS, L. R., A New Era for DSP Systems Design?, **IEEE Micro**, Dez. 1988, pp. 10-12.
- [EGG-90] EGGBRECHT, L. W., **Interfacing to the IBM Personal Computer**, Indiana, Howard Sams, 1990
- [FLA-78] FLADUNG, R. L. & MERGER, H. W., High Performance Fast Fourier Transformer, **IEEE Trans. on Industrial Electronics and Control Instrumentation**, Vol. IECI-25, No. 4, pp. 322-326, Nov. 1978
- [GOL-69] GOLD, B. & RADER, C. M., **Digital Processing of Signals**, New York, McGraw Hill, 1969
- [GOL-73] GOLD, B. & BIALLY, T., Paralellism in Fast Fourier Transform Hardware, **IEEE Trans. Audio Electroacoustics**, Vol. AU-21, pp. 5-16, Feb. 1973

- [GOO-71] GOOD, I. J., The relationship between two fast Fourier transforms, **IEEE Trans. Comput.**, Vol C-20, pp. 310-317, 1971
- [GUN-85] GÜNDEL, L., A Novel High-Speed Fourier Vector Processor, **Signal Processing**, Vol. 9, No. 2, Set. 1985, pp. 107-120.
- [HWA-84] HWANG, K. & BRIGGS, F. A., **Computer Architecture and Parallel Processing**, New York, McGraw-Hill, 1984
- [INT-78] INTEL CORPORATION, **Component Data Catalog**, Santa Clara, 1978
- [INT-84] INTEL CORPORATION, **IAPX 286 Hardware Reference Manual**, Santa Clara, 1984
- [KOR-79] KORN, D. G. & LAMBIOTTE, J. J., Coomputing the Fast Fourier Transform on a Vector Computer, **Mathematics of Computation**, Vol. 33, No. 147, pp. 977-992, Jul. 1979
- [MOR-77] MORRIS, L. R. Automatic Generation of Time Efficient Digital Signal Processing Software, **IEEE Trans. on Acoustics, Speech and Signal Processing**, Vol. ASSP-25, No. 1, pp. 74-79, 1977
- [MYE-90] MYERS, D. G., **Digital Signal Processing - Efficient Convolution and Fourier Transform Techniques**, New York, Prentice Hall, 1990, 355p.
- [NAT-87] NATIONAL SEMICONDUCTOR CORPORATION, **LS/S/TTL Logic Databook**, Santa Clara, 1987
- [NAT*-87] NATIONAL SEMICONDUCTOR CORPORATION, **Ramdom Access Memory Databook**, Santa Clara, 1987
- [NUS-81] NUSSBAUMER, HENRI J., **Fast Fourier Transform and Convolution Algorithms**, Berlim, Springer Verlag, 1981
- [NUT-81] NUTTALL, A. H., Some Windows with Very Good Sidelobe Behavior, **IEEE Trans. on Acoustics, Speech and Signal Processing**, Vol. ASSP-29, No. 1, pp. 84-91, Fev. 1981
- [OPP-75] OPPENHEIM, A. V. & SCHAFFER, R. W., **Digital Signal Processing**, New Jersey, Prentice Hall International Editions, 1975
- [RIE-85] RIEDEL, N. K et al, A Signal Processing Implementation for an IBM-PC-based Workstation, **IEEE Micro**, pp. 52-67, Out. 1985

- [ROB-87] ROBERTS, R. A. & MULLIS, C. T., **Digital Signal Processing**, Reading, Addison-Wesley, 1987
- [ROS-90] ROSCH, W. L., **Desvendando o Hardware do PC**, Rio de Janeiro, Ed. Campus, 1990
- [SCH-92] SCHMALZEL, J. L. & PATEL, P., A., Transforming the World of Digital Signal Processing, **IEEE Micro**, pp. 8-9, Dez. 1992.
- [SIL-77] SILVERMAN, H. S., An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA), **IEEE Trans. on Acoustics, Speech and Signal Processing**, Vol ASSP-25, No. 2, pp. 152-165, 1977
- [STI-82] STIGALL, P. D.; ZIEMER, R. E. & HUDEC, L., A Performance Study of 16 bit Microcomputer-implemented FFT Algorithms, **IEEE Micro**, pp. 61-66, Nov. 1982
- [TON-92] TONISSI, S. A., Instrumentação Eletrônica Para Pesquisas na Área de Ótica, **Tese de Mestrado**, Instituto de Física e Química de São Carlos, 1992
- [VET-86] VETTERLI, M. & LIGTENBERG, A., A Discrete Fourier-Cosine Transform Chip, **IEEE Journal on Selected Areas in Communications**, Vol. SAC-4, No. 1, pp. 49-61, Jan. 1986
- [ZOR-86] ZORAN CORPORATION, **ZR34161 Vector Signal Processor - Engineering Data**, Santa Clara, 1986
- [ZOR-87] ZORAN CORPORATION, **ZR34161 Vector Signal Processor - User Manual**, Santa Clara, 1987
- [ZOR*-87] ZORAN CORPORATION, **VSPA-161 Vector Signal Processor - Assembler Reference Manual**, Santa Clara, 1987
- [ZOR-88] ZORAN CORPORATION, **VSPE-161 and VSPX-161 Evaluation Package**, Santa Clara, 1988
- [ZOR*-88] ZORAN CORPORATION, **ZR34161 Vector Signal Processor Simulator - User's Manual**, Santa Clara, 1988

Apêndices

A. Listagem do Algoritmo de FFT

Este programa processa um vetor real com 1024 elementos. Ele foi usado no lugar do programa original para 4096 elementos por ser menor e mais facilmente compreendido.

```

equ      TamVet=1024;
equ      Dados=0, H=Dados, G=Dados+1026;
equ      Trans=G+1026, TabApod=1024;
equ      IniProg=12300;

org      IniProg;

DEFAULT  INTRP:0, EI:0, RV:0, MDF:3, ZR:0, ZP:0;
DEFAULT  RBA:0, FSIZ:64, AS:0, I:0;

/*
*/      Resetando o VSP

LDASM    NMPT:1, UP:0, MD:1, MBA:ModoIni;

/*
*/      Definindo 2 secoes de memoria RAM

LDASM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

/*
*/      Salva valor inicial da tabela de multiplicacoes da FFT

LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabResF+0;
ST        NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF+0;

/*
*/      Salva valor inicial da tabela de multiplicacoes da composicao da FFT

LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabResC+0;
ST        NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulC+0;

/*
*/      Multiplicando vetores

/*
*/      Define apenas 1 secao de memoria

LDASM    NMPT:1, UP:0, MD:1, MBA:ModoUni;

/*
*/      Multiplica os vetores Dados * TabApod = Dados

LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+0;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+0;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+0;
LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+128;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+128;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+128;
LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+256;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+256;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+256;
LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+384;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+384;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+384;
LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+512;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+512;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+512;
LD        NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+640;
MLTR      NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+640;
ST        NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+640;

```

```

LD      NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+768;
MLTR    NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+768;
ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+768;
LD      NMPT:128, RS:0, RV:0, MDF:2, ZR:1, MBS:2, MSS:2, MBA:Dados+896;
MLTR    NMPT:128, RS:0, ADF:2, MBS:2, MSS:2, MBA:TabApod+896;
ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:Dados+896;

/*
  Retorn com 2 secoes de memoria
*/

LDSM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

/*
  FFT horizontal
*/

DEFAULT MBS:1, MSS:8, RV:0, FPS:32, LPS:1, AS:0;

LD      NMPT:64, RS:0, MBA:Dados+0;
FFT     NMBT:64, RS:0;
LD      NMPT:64, RS:1, MBA:Dados+2;
ST      NMPT:64, RS:0, MBA:Dados+0, RV:1;
FFT     NMBT:64, RS:1;
LD      NMPT:64, RS:0, MBA:Dados+4;
ST      NMPT:64, RS:1, MBA:Dados+2, RV:1;
FFT     NMBT:64, RS:0;
LD      NMPT:64, RS:1, MBA:Dados+6;
ST      NMPT:64, RS:0, MBA:Dados+4, RV:1;
FFT     NMBT:64, RS:1;
LD      NMPT:64, RS:0, MBA:Dados+8;
ST      NMPT:64, RS:1, MBA:Dados+6, RV:1;

/*
  Salvando o registro de escala em RamEscH
*/

LDSM    NMPT:1, UP:0, MD:1, MBA:ModoScl;
STI     NMPT:1, STR:5, MBA:RamEscH+0;
LDSM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

FFT     NMBT:64, RS:0;
LD      NMPT:64, RS:1, MBA:Dados+10;
ST      NMPT:64, RS:0, MBA:Dados+8, RV:1;
FFT     NMBT:64, RS:1;
LD      NMPT:64, RS:0, MBA:Dados+12;
ST      NMPT:64, RS:1, MBA:Dados+10, RV:1;
FFT     NMBT:64, RS:0;
LD      NMPT:64, RS:1, MBA:Dados+14;
ST      NMPT:64, RS:0, MBA:Dados+12, RV:1;
FFT     NMBT:64, RS:1;
ST      NMPT:64, RS:1, MBA:Dados+14, RV:1;

/*
  Salvando o registro de escala em RamEscH
*/

LDSM    NMPT:1, UP:0, MD:1, MBA:ModoScl;
STI     NMPT:1, STR:5, MBA:RamEscH+1;
LDSM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

/*
  Multiplicacao pelos pesos, escalamento e fft vertical
*/

/*
  Define o maximo scale register e carrega fator de escalento
*/

LDSM    NMPT:1, UP:0, MD:1, MBA:ModoScl;
LDSM    NMPT:2, UP:1, MD:0, MBA:RamEscH;
LDSM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

DEFAULT MBS:8, MSS:8, RV:0, SB:1, LN:1;
DEFAULT FPS:4, LPS:1, CN:0;

LD      NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
MLTC    NMPT:64, RS:0, SH:1, MBA:Dados+0;
LD      NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;

```

```

SCL      NMPT:64, RS:0, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:0, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:0, RV:3, MBA:Dados+0;
MLTC     NMPT:64, RS:1, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:1, SH:1, MBA:Dados+128;
LD       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:1, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:1, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:1, RV:3, MBA:Dados+128;
MLTC     NMPT:64, RS:0, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:0, SH:1, MBA:Dados+256;
LD       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:0, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:0, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:0, RV:3, MBA:Dados+256;
MLTC     NMPT:64, RS:1, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:1, SH:1, MBA:Dados+384;
LD       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:1, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:1, R:0, FSIZ:8, RBA:0;
ST       NMPT:1, RS:1, MBA:VarSinc;

```

/*

Salvando o registro de escala em RamEscV

*/

```

LDSM     NMPT:1, UP:0, MD:1, MBA:ModoScl;
STI      NMPT:1, STR:5, MBA:RamEscV+0;
LDSM     NMPT:1, UP:0, MD:1, MBA:ModoNrm;

ST       NMPT:64, RS:1, RV:3, MBA:Dados+384;
MLTC     NMPT:64, RS:0, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:0, SH:1, MBA:Dados+512;
LD       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:0, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:0, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:0, RV:3, MBA:Dados+512;
MLTC     NMPT:64, RS:1, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:1, SH:1, MBA:Dados+640;
LD       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:1, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:1, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:1, RV:3, MBA:Dados+640;
MLTC     NMPT:64, RS:0, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:0, SH:1, MBA:Dados+768;
LD       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:0, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:0, R:0, FSIZ:8, RBA:0;
ST       NMPT:64, RS:0, RV:3, MBA:Dados+768;
MLTC     NMPT:64, RS:1, MBS:64, MSS:64, SH:0, MBA:TabRefF;
ST       NMPT:64, RS:1, MBS:64, MSS:64, MBA:TabMulF;
MLTC     NMPT:64, RS:1, SH:1, MBA:Dados+896;
LD       NMPT:64, RS:0, MBS:64, MSS:64, MBA:TabMulF;
SCL      NMPT:64, RS:1, SCLBL:1, SCLVL:8;
FFT      NMBT:64, RS:1, R:0, FSIZ:8, RBA:0;
ST       NMPT:1, RS:1, MBA:VarSinc;

```

/*

Salvando o registro de escala em RamEscV

*/

```

LDSM     NMPT:1, UP:0, MD:1, MBA:ModoScl;
STI      NMPT:1, STR:5, MBA:RamEscV+1;
LDSM     NMPT:1, UP:0, MD:1, MBA:ModoNrm;

ST       NMPT:64, RS:1, RV:3, MBA:Dados+896;

```

/*

Escalamento vertical, bit-reversao e mapeamento

*/

/*

Define o maximo scale register e carrega fator de escalento

*/

```

LDSM      NMPT:1, UP:0, MD:1, MBA:ModoScl;
LDSM      NMPT:2, UP:1, MD:0, MBA:RamEscV;
LDSM      NMPT:1, UP:0, MD:1, MBA:ModoNrm;

DEFAULT   MBS:1, MSS:8, RV:0, SB:1, LN:1;

LD        NMPT:64, RS:0, MBA:Dados+0;
SCL       NMPT:64, RS:0, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:1, MBA:Dados+2;
ST        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+0;
SCL       NMPT:64, RS:1, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:0, MBA:Dados+4;
ST        NMPT:64, RS:1, MBS:64, MSS:64, MBA:Trans+128;
SCL       NMPT:64, RS:0, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:1, MBA:Dados+6;
ST        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+256;
SCL       NMPT:64, RS:1, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:0, MBA:Dados+8;
ST        NMPT:64, RS:1, MBS:64, MSS:64, MBA:Trans+384;
SCL       NMPT:64, RS:0, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:1, MBA:Dados+10;
STB       NMPT:64, RS:0, MBS:64, MSS:64, MBAB:Trans+1025;
SCL       NMPT:64, RS:1, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:0, MBA:Dados+12;
STB       NMPT:64, RS:1, MBS:64, MSS:64, MBAB:Trans+897;
SCL       NMPT:64, RS:0, SCLBL:8, SCLVL:8;
LD        NMPT:64, RS:1, MBA:Dados+14;
STB       NMPT:64, RS:0, MBS:64, MSS:64, MBAB:Trans+769;
SCL       NMPT:64, RS:1, SCLBL:8, SCLVL:8;
STB       NMPT:64, RS:1, MBS:64, MSS:64, MBAB:Trans+641;
LD        NMPT:1, RS:0, MBA:Trans;
ST        NMPT:1, RS:0, MBA:Trans+512;

/*
Montando o vetor H e G
*/

LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+0;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+512;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:H+0;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBAB:H+1024;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:G+0;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:G+1024;
LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+128;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+640;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:H+128;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBAB:H+896;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:G+128;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:G+896;
LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+256;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+768;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:H+256;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBAB:H+768;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:G+256;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:G+768;
LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+384;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+896;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:H+384;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBAB:H+640;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:G+384;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:G+640;
LD        NMPT:1, RS:0, MDF:2, ZR:1, MBA:Trans+1024;
ST        NMPT:1, RS:0, MBA:H+512;
LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+0;
MLTC      NMPT:64, RS:0, MBS:64, MSS:64, CN:1, SH:0, MBA:CteNeg;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+512;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:G+1;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:H+1025;
CMCN      NMPT:64, RS:0;
LD        NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBA:G+1;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:H+1;
CMCN      NMPT:64, RS:1;
STB       NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBAB:G+1025;
LD        NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+128;
MLTC      NMPT:64, RS:0, MBS:64, MSS:64, CN:1, SH:0, MBA:CteNeg;
ADDC      NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+640;
ST        NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:G+129;
STB       NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:H+897;
CMCN      NMPT:64, RS:0;
LD        NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBA:G+129;

```

```

ST      NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:H+129;
CMCN    NMPT:64, RS:1;
STB     NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBAB:G+897;
LD      NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+256;
MLTC    NMPT:64, RS:0, MBS:64, MSS:64, CN:1, SH:0, MBA:CteNeg;
ADDC    NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+768;
ST      NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:G+257;
STB     NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:H+769;
CMCN    NMPT:64, RS:0;
LD      NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBA:G+257;
ST      NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:H+257;
CMCN    NMPT:64, RS:1;
STB     NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBAB:G+769;
LD      NMPT:64, RS:0, MBS:64, MSS:64, MBA:Trans+384;
MLTC    NMPT:64, RS:0, MBS:64, MSS:64, CN:1, SH:0, MBA:CteNeg;
ADDC    NMPT:64, RS:0, MBS:64, MSS:64, SH:1, MBA:Trans+896;
ST      NMPT:64, RS:0, MBS:1, MSS:2, MDF:2, MBA:G+385;
STB     NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBAB:H+641;
CMCN    NMPT:64, RS:0;
LD      NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBA:G+385;
ST      NMPT:64, RS:0, MBS:1, MSS:2, MDF:1, MBA:H+385;
CMCN    NMPT:64, RS:1;
STB     NMPT:64, RS:1, MBS:1, MSS:2, MDF:1, MBAB:G+641;
LD      NMPT:1, RS:0, MDF:2, ZR:1, MBA:Trans+1025;
ST      NMPT:1, RS:0, MBA:G+512;

```

```

/*
Compondo a FFT com G e H
*/

```

```

DEFAULT MBS:64, MSS:64, RV:0, CN:0, SH:0;

LD      NMPT:64, RS:0, MBA:G+0;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+0;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+0;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+128;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+128;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+128;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+256;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+256;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+256;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+384;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+384;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+384;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+512;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+512;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+512;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+640;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+640;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+640;
ST      NMPT:64, RS:1, MBA:TabMulC;
LD      NMPT:64, RS:0, MBA:G+768;
MLTC    NMPT:64, RS:0, MBA:TabMulC;
LD      NMPT:64, RS:1, MBA:TabMulC;
ADDC    NMPT:64, RS:0, SH:1, MBA:H+768;
MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
ST      NMPT:64, RS:0, MBA:H+768;
ST      NMPT:64, RS:1, MBA:TabMulC;

```

```

        LD      NMPT:64, RS:0, MBA:G+896;
        MLTC    NMPT:64, RS:0, MBA:TabMulC;
        LD      NMPT:64, RS:1, MBA:TabMulC;
        ADDC    NMPT:64, RS:0, SH:1, MBA:H+896;
        MLTC    NMPT:64, RS:1, CN:1, MBA:CteMulC;
        ST      NMPT:64, RS:0, MBA:H+896;

/*
Calculando o modulo do resultado
*/

/*
Define apenas 1 secao de memoria
*/

        LDSM    NMPT:1, UP:0, MD:1, MBA:ModoUni;

/*
Calcula o modulo de  $H^2 = \text{TabApod}$ 
*/

        LD      NMPT:128, RS:0, RV:0, MDF:3, MBS:2, MSS:2, MBA:H+0;
        MGSQ    NMPT:128, RS:0, ADF:2;
        ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:TabApod+0;
        LD      NMPT:128, RS:0, RV:0, MDF:3, MBS:2, MSS:2, MBA:H+256;
        MGSQ    NMPT:128, RS:0, ADF:2;
        ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:TabApod+128;
        LD      NMPT:128, RS:0, RV:0, MDF:3, MBS:2, MSS:2, MBA:H+512;
        MGSQ    NMPT:128, RS:0, ADF:2;
        ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:TabApod+256;
        LD      NMPT:128, RS:0, RV:0, MDF:3, MBS:2, MSS:2, MBA:H+768;
        MGSQ    NMPT:128, RS:0, ADF:2;
        ST      NMPT:128, RS:0, RV:0, MDF:2, MBS:2, MSS:2, MBA:TabApod+384;

/*
Retorn com 2 secoes de memoria
*/

        LDSM    NMPT:1, UP:0, MD:1, MBA:ModoNrm;

        HLT     ;

/*
Modos de operacao
*/

/*
Gera um reset no VSP
*/

ModoIni:
        dw      0x507B;

/*
Operacao normal, 2 secoes de RAM, 1 ciclo/leitura e as 3 int. ativas
*/

ModoNrm:
        dw      0x107B;

/*
Define 2 ciclos/leitura, exigencia da instrucao STI
*/

ModoScl:
        dw      0x007B;

/*
Operacoes de apodizacao e modulo operam em RAM com 1 secao
*/

ModoUni:
        dw      0x187B;

/*
Espaco reservado para sincronismo
*/

```



```

VarSinc:      dw      0x0000;
              dw      0x0000;

/*
 * Constante valendo -1 para usada subtracao
 */

CteNeg:       dw      32769;
              dw      0;

/*
 * Constante de peso para a compor FFT ( W(64,NLIN*2*64) )
 */

CteMulC:      dw      30273;
              dw      52997;

/*
 * RAM de escala horizontais e verticais
 */

RamEscH:      dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;

RamEscV:      dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;
              dw      0x0000;

/*
 * Tabela de reserva para compor FFT
 */

TabResC:

... Seguem 64 números complexos, omitidos para simplificação ...;

/*
 * Tabela de multiplicacao para compor FFT
 */

TabMulC:

... Seguem 64 números complexos, omitidos para simplificação ...;

/*
 * Tabela de peso de reserva

```

```
*/
```

```
TabResF:
```

```
    ... Seguem 64 números complexos, omitidos para simplificação ...;
```

```
/*
```

```
   Tabela de peso de referencia
```

```
*/
```

```
TabRefF:
```

```
    ... Seguem 64 números complexos, omitidos para simplificação ...;
```

```
/*
```

```
   Espaco livre para tabela de peso de multiplicacao
```

```
*/
```

```
TabMulF:
```

```
    dw      0;
```

B. Listagem do Programa de Montagem

O tamanho do vetor a ser processado é definido pela constante NLIN. Este exemplo processa um vetor com 8*64 elementos complexos, ou 8*64*2 elementos reais.

```
{ $DEFINE PREPOS }           { Ativa pre e pos processamento - apod, modulo }
{ $DEFINE REAL }             { Ativa composicao da FFT complexa p/ FFT real }

{ Modulo para geracao do programa assembler baseado no metodo "autogen",
  e usando algoritmo da fft de linhas e colunas, otimizado.
  Faz a transformada de 2N pontos reais com apenas uma transformada de N ptos
  complexos. Faz a apodizacao antes usando uma tabela transferida para o VSP
  junto com os dados. Faz os pre e pos processamentos com a memoria interna
  nao dividida em 2 secoes, para diminuir o numero de instrucoes e melhor
  utilizar a memoria, pois senao o programa nao cabe em 16 K }
```

program gerafft;

uses crt, dos;

const

{ Define o numero de linhas de 64 pontos cada o vetor de dados
 complexo possui. O total de pontos eh NLIN*64 ptos complexos ou
 o dobro de pontos reais. }

NLIN = 8;

TAB = #9; {Codigo de tabulacao }

ABREC = '/*'; { Simbolo de abertura de comentario }

FECHAC = '*/'; { Simbolo de fechamento de coment }

FIMC = ';;'; { Simbolo de encerramento de comando }

DC = ' '; { Espacos para separar comandos }

{ Definicao dos comandos do VSP }

ABS = 'ABS';

ACCI = 'ACCI';

ACCR = 'ACCR';

ADDC = 'ADDC';

ADDR = 'ADDR';

CMCN = 'CMCN';

CMLT = 'CMLT';

DEMO = 'DEMO';

FFT = 'FFT';

HLT = 'HLT';

JMPI = 'JMPI';

LD = 'LD';

LDSM = 'LDSM';

MGSQ = 'MGSQ';

MLTC = 'MLTC';

MLTR = 'MLTR';

MODLT = 'MODLT';

NOP = 'NOP';

SCL = 'SCL';

SCLT = 'SCLT';

ST = 'ST';

STB = 'STB';

STI = 'STI';

DEFAULT = 'DEFAULT';

{ Definicao dos parametros das instrucoes }

ADF = 'ADF';

AS = 'AS';

CN = 'CN';

EI = 'EI';

FPS = 'FPS';

FSIZ = 'FSIZ';

PIF = 'I'; { Modificado de I }

INTRP = 'INTRP';

LN = 'LN';

```

LPS          = 'LPS';
MBA          = 'MBA';
MBAB         = 'MBAB';
MBS          = 'MBS';
MD           = 'MD';
MDF          = 'MDF';
MSS          = 'MSS';
NMBT         = 'NMBT';
NMPT         = 'NMPT';
POR          = 'OR';           { Modificado de OR }
R            = 'R';
RBA          = 'RBA';
RDA          = 'RDA';
RIA          = 'RIA';
RS           = 'RS';
RV           = 'RV';
SB           = 'SB';
SCLBL        = 'SCLBL';
SCLVL        = 'SCLVL';
SH           = 'SH';
SHF          = 'SHF';
PSTR         = 'STR';         { Modificado de STR }
UP           = 'UP';
VSIZ         = 'VSIZ';
ZP           = 'ZP';
ZR           = 'ZR';

var
  scom        : string;       { Buffer da linha de comando }
  cpar        : word;         { Contador de parametros de comando }
  alg         : text;         { Arquivo de geracao do algoritmo }
  ram         : word;         { Num da secao da memor. RAM do VSP }

{ Transforma inteiro em string }

function stri (ni: integer): string;
var
  s            : string;
begin
  str (ni, s);
  stri := s;
end;

{ Transforma word em string }

function strw (nw: word): string;
var
  s            : string;
begin
  str (nw, s);
  strw := s;
end;

{ ****
  Funcoes de geracao dos comandos do VSP
  **** }

{ Acrescenta uma linha no arquivo de programa do VSP }

procedure v (l: string);
begin
  writeln (alg, l);
end;

{ Descarrega uma linha de comando no arquivo de programa do VSP }

procedure flushc;
begin
  if scom <> ''
  then begin
    v (scom);
    scom := '';
    cpar := 0;
  end;

```

```

        end;
end;

{ Acrescenta uma linha em branco no arquivo de saida do VSP }

procedure vl;
begin
flushc;
v (' ');
end;

{ Acrescenta uma linha de comentario no arquivo de saida do VSP }

procedure vc (l: string);
begin
vl;
v (ABREC);
v (DC + l);
v (FECHAC);
vl;
end;

{ Acrescenta um "equate" no arquivo de saida }

procedure equ (nome, sval: string);
begin
nome := TAB + 'equ' + TAB + nome + '=' + sval + ';';
v (nome);
end;

{ Acrescenta um "org" no arquivo de saida }

procedure org (nome: string);
begin
nome := TAB + 'org' + TAB + nome + ';';
v (nome);
end;

{ Acrescenta um "define word" no arquivo de saida }

procedure dw (nome, sval: string);
begin
if nome <> ''
then v (nome + ':');
if sval <> ''
then begin
nome := TAB + 'dw' + TAB + sval + ';';
v (nome);
end;
end;

{ Acrescenta uma virgula para separar paramentros nos comandos do VSP }

function virg: string;
var
p          : byte;
begin
if cpar > 0
then virg := ', '
else virg := '';
end;

{ Inicia o buffer de comando com o nome do comando }

procedure c (com: string);
begin
flushc;
scom := TAB + com + TAB + FIMC;
end;

{ Acrescenta um parametro tipo word no buffer de comando }

procedure p (par: string; vlr: word);

```

```

var
    sv
        : string;
begin
    str (vlr, sv);
    par := virg + par + ':' + sv;
    insert (par, scom, length (scom));
    inc (cpar);
end;

{ Acrescenta um parametro do tipo deslocamento em relacao a um endereco base
  no buffer de comando }

procedure q (par, bas: string; desl: word);
var
    sv
        : string;
begin
    str (desl, sv);
    par := virg + par + ':' + bas + '+' + sv;
    insert (par, scom, length (scom));
    inc (cpar);
end;

{ Acrescenta um parametro do tipo string no buffer de comando }

procedure s (par, bas: string);
begin
    par := virg + par + ':' + bas;
    insert (par, scom, length (scom));
    inc (cpar);
end;

{ Acrescenta uma string no buffer de comando }

procedure t (par: string);
begin
    par := virg + par;
    insert (par, scom, length (scom));
    inc (cpar);
end;

{ Acrescenta uma atribuicao de um endereco base e de um deslocamento no
  buffer de comando. Se o endereco base for vazio, faz a atribuicao apenas
  do deslocamento. }

procedure u (par, bas: string; desl: word);
var
    sv, px
        : string;
begin
    str (desl, sv);
    if bas = ''
    then px := ''
    else px := bas;
    if desl = 0
    then px := px
    else if px = ''
        then px := px + sv
        else px := px + '+' + sv;
    par := par + '=' + px;
    t (par);
end;

{ *****

  Funcoes auxiliares para a FFT

  ***** }

{ Inicializa o arquivo de programa do VSP, e das variaveis relacionadas }

procedure inig;
var
    i
        : word;
begin
    assign (alg, 'PROG.VSP');
    rewrite (alg);

```

```

scom := '';
cpar := 0;
end;

{ Faz o "toggle" de uma variavel de dois estados }

function inv (n: word): word;
begin
if n = 1
then inv := 0
else inv := 1;
end;

{ Retorna o numero da outra secao de RAM do VSP que nao a secao corrente }

function ira: word;
begin
ira := inv (ram);
end;

{ Salva o registro com as 4 escalas das 4 ultimas FFTs na memoria RAM externa }

procedure salvasc1 (modsc, endsc, modnr: string; n: word);
var
    smba          : string;
begin
vc ('Salvando o registro de escala em ' + endsc);
c(LDSM); p(NMPT,1); p(UP,0); p(MD,1); s(MBA,modsc);
c(STI); p(NMPT,1); p(PSTR,5); q(MBA,endsc,n);
c(LDSM); p(NMPT,1); p(UP,0); p(MD,1); s(MBA,modnr);
vl;
end;

{ Carrega na memoria RAM de expoentes do VSP os expoentes das ultimas FFTs
  realizadas. Determina tambem o maior expoente entre eles. }

procedure defomsr (modsc, endsc, modnr: string; nnib: word);
var
    nw          : word;
begin
if (nnib mod 4) = 0
then nw := nnib div 4
else nw := nnib div 4 + 1;
vc ('Define o maximo scale register e carrega fator de escalento');
c(LDSM); p(NMPT,1); p(UP,0); p(MD,1); s(MBA,modsc);
c(LDSM); p(NMPT,nw); p(UP,1); p(MD,0); s(MBA,endsc);
c(LDSM); p(NMPT,1); p(UP,0); p(MD,1); s(MBA,modnr);
vl;
end;

{ Carrega na memoria RAM de expoentes do VSP os 4 ultimos expoentes das 4
  ultimas FFTs realizadas. Usado no lugar de DEFOMSR quando o numero de
  expoentes eh menor que 4. }

procedure carrsc1 (modsc, endsc, modnr: string; nw: word);
begin
vc ('Carrega 4 nibbles para escala');
c(LDSM); p(NMPT,1); p(UP,0); p(MD,0); q(MBA,endsc,nw);
vl;
end;

{ Sincroniza o processamento, garantindo que a secao de RAM corrente do VSP
  nao esta sendo usada em paralelo pela unidade aritmetica }

procedure sincro;
begin
c(ST); p(NMPT,1); p(RS,ram); s(MBA,'VarSinc');
end;

{ *****

Funcoes para a apodizacao de um vetor real e calculo do modulo de um
vetor complexo

```

```

*****}

{ Multiplica dois vetores reais de tamanho l*64 pontos }

procedure mulvetr (endv1, endv2, endrs: string; l: word);
var i
  : word;
begin
  l := l;
  ram := 0;
  vc ('Define apenas 1 secao de memoria');
  c(LDSM);    p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoUni');
  vc ('Multiplica os vetores ' + endv1 + ' * ' + endv2 + ' = ' + endrs);
  for i := 0 to l-1 do
    begin
      c(LD);    p(NMPT,128); p(RS,ram); p(RV,0); p(MDF,2); p(ZR,1); p(MBS,2); p(MSS,2);
      q(MBA,endv1,i*128);
      c(MLTR);  p(NMPT,128); p(RS,ram); p(ADF,2); p(MBS,2); p(MSS,2); q(MBA,endv2,i*128);
      c(ST);    p(NMPT,128); p(RS,ram); p(RV,0); p(MDF,2); p(MBS,2); p(MSS,2); q(MBA,endrs,i*128);
    end;
  vc ('Retorn com 2 secoes de memoria');
  c(LDSM);    p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoNrm');
end;

{ Calcula o quadrado do modulo de um vetor de tamanho l*64 pontos }

procedure modvet (endv1, endrs: string; l: word);
var i
  : word;
begin
  l := l div 2;
  ram := 0;
  vc ('Define apenas 1 secao de memoria');
  c(LDSM);    p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoUni');
  vc ('Calcula o modulo de ' + endv1 + ' ^ 2 = ' + endrs);
  for i := 0 to l-1 do
    begin
      c(LD);    p(NMPT,128); p(RS,ram); p(RV,0); p(MDF,3); p(MBS,2); p(MSS,2);
      q(MBA,endv1,i*128*2);
      c(MGSQ);  p(NMPT,128); p(RS,ram); p(ADF,2);
      c(ST);    p(NMPT,128); p(RS,ram); p(RV,0); p(MDF,2); p(MBS,2); p(MSS,2); q(MBA,endrs,i*128);
    end;
  vc ('Retorn com 2 secoes de memoria');
  c(LDSM);    p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoNrm');
end;

{ *****

  Funcoes para a FFT em um vetor complexo

  *****}

{ Faz a FFT de l linhas horizontais, salvando-as na sequencia natural nas
  mesmas posicoes }

procedure ffth (endin: string; l: word; modsc, endsc, modnr: string);
var
  i, nd
    : word;
begin
  ram := 0;
  nd := 0;
  c(DEFAULT); p(MBS,1); p(MSS,1); p(RV,0); p(FPS,32); p(LPS,1); p(AS,0);
  vl;
  c(LD);    p(NMPT,64); p(RS,ram); q(MBA,endin,0);
  for i := 1 to l do
    begin
      c(FFT);  p(NMBT,64); p(RS,ram);
      if i < l
      then begin
        c(LD);    p(NMPT,64); p(RS,ira); q(MBA,endin,i*2);
        end;
      c(ST);    p(NMPT,64); p(RS,ram); q(MBA,endin,i*2-2); p(RV,1);
      ram := ira;
      if (i mod 4) = 0
      then begin
        salvascl (modsc, endsc, modnr, nd);
        inc (nd);
        end;
    end;
end;

```



```

if (l mod 4) <> 0
then salvascl (modsc, endsc, modnr, nd);
end;

{ Multiplica o resultado da FFT horizontal pelas constantes W, escala este
  resultado e as FFTs horizontais, e faz as FFTs verticais. Estas operacoes
  sao feitas em 1 vetores de 64 pontos }

procedure multsclfft (endin, tabmul, tabref, modsc, endsh, endsv, modnr: string; l: word);
var
  i, d          : word;
begin
  defomsr (modsc, endsh, modnr, 1);
  c(DEFAULT); p(MBS,1); p(MSS,1); p(RV,0); p(SB,1); p(LN,1);
  c(DEFAULT); p(FPS,1 div 2); p(LPS,1); p(CN,0);
  vl;
  d := 0;
  ram := 0;
  c(LD);      p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); s(MBA,tabmul);
  for i := 1 to l do
    begin
      c(MLTC);      p(NMPT,64); p(RS,ram); p(SH,1); q(MBA,endin,(i-1)*128);
      c(LD);        p(NMPT,64); p(RS,ira); p(MBS,64); p(MSS,64); s(MBA,tabmul);
      c(SCL);       p(NMPT,64); p(RS,ram); p(SCLBL,1); p(SCLVL,1);
      c(FFT);       p(NMBT,64); p(RS,ram); p(R,0); p(FSIZ,1); p(RBA,0);
      if (i mod 4) = 0
      then begin
        sincro;
        salvascl (modsc, endsv, modnr, d);
        inc (d);
      end;
      c(ST);        p(NMPT,64); p(RS,ram); p(RV,3); q(MBA,endin,(i-1)*128);
      if i < l
      then begin
        c(MLTC);      p(NMPT,64); p(RS,ira); p(MBS,64); p(MSS,64); p(SH,0); s(MBA,tabref);
        c(ST);        p(NMPT,64); p(RS,ira); p(MBS,64); p(MSS,64); s(MBA,tabmul);
      end;
      ram := ira;
    end;
  end;

{ Escala os resultados da FFT vertical e salva-os em ordem natural em outra
  posicao. Salva tambem em ordem reversa. Esta operacoes sao feitas em 1
  vetores de 64 pontos }

procedure sclvpos (endin, endrv: string; l: word; modsc, endsc, modnr: string);
var
  i, nr, ir, j      : word;
  cl, l2, f          : word;
begin
  j := 0;
  ir := 1;
  ram := 0;
  cl := 64 div l;
  l2 := l div 2;
  f := l2*128 + 128 + 2 - 1;
  defomsr (modsc, endsc, modnr, 1);
  c(DEFAULT); p(MBS,1); p(MSS,1); p(RV,0); p(SB,1); p(LN,1);
  vl;
  c(LD);      p(NMPT,64); p(RS,0); q(MBA,endin,0);
  for i := 1 to l do
    begin
      c(SCL);      p(NMPT,64); p(RS,ram); p(SCLBL,cl); p(SCLVL,1);
      if i < l
      then begin
        c(LD);      p(NMPT,64); p(RS,ira); q(MBA,endin,i*2);
      end;
      if i > l2
      then begin
        c(STB);      p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); q(MBAB,endrv,(l-i)*128+f);
      end
      else begin
        c(ST);       p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); q(MBA,endrv,(i-1)*128);
      end;
      ram := ira;
    end;
  c(LD);      p(NMPT,1); p(RS,0); s(MBA,endrv);
  c(ST);      p(NMPT,1); p(RS,0); q(MBA,endrv,l2*128);
end;

```

```

{*****

    Funcoes para calculo da FFT de um vetor real a partir das FFTs de
    vetores pares e impares

*****}

{ Calcula os dois vetores que sao a FFT da parte par e impar do vetor de
  dados real de entrada de tamanho 2*1*64 pontos }

procedure calcpr (trnin, vetr, veti, ctneg: string; l: word);
var
    m, i, f, d, l2 : word;
begin
    ram := 0;
    m := 0;
    l2 := 1 div 2;
    d := 12 * 128;
    f := 12*128 + 2 - 1;
    for i := 0 to l2-1 do
        begin
            c(LD); p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); q(MBA,trnin,i*128);
            c(ADDC); p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); p(SH,1); q(MBA,trnin,d+i*128);
            c(ST); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,2); q(MBA,vetr,i*128);
            c(STB); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,2); q(MBAB,vetr,f+(12-i)*128-1);
            c(ST); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,1); q(MBA,veti,i*128);
            c(STB); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,1); q(MBAB,veti,f+(12-i)*128-1);
        end;
    c(LD); p(NMPT,1); p(RS,ram); p(MDF,2); p(ZR,1); q(MBA,trnin,l*128);
    c(ST); p(NMPT,1); p(RS,ram); q(MBA,vetr,d);
    for i := 0 to l2-1 do
        begin
            c(LD); p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); q(MBA,trnin,i*128);
            c(MLTC); p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); p(CN,1); p(SH,0); s(MBA,ctneg);
            c(ADDC); p(NMPT,64); p(RS,ram); p(MBS,64); p(MSS,64); p(SH,1); q(MBA,trnin,d+i*128);
            c(ST); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,2); q(MBA,veti,i*128+1);
            c(STB); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,1); q(MBAB,vetr,f+(12-i)*128);
            c(CMCN); p(NMPT,64); p(RS,ram);
            c(LD); p(NMPT,64); p(RS,ira); p(MBS,1); p(MSS,2); p(MDF,1); q(MBA,veti,i*128+1);
            c(ST); p(NMPT,64); p(RS,ram); p(MBS,1); p(MSS,2); p(MDF,1); q(MBA,vetr,i*128+1);
            c(CMCN); p(NMPT,64); p(RS,ira);
            c(STB); p(NMPT,64); p(RS,ira); p(MBS,1); p(MSS,2); p(MDF,1); q(MBAB,veti,f+(12-i)*128);
        end;
    c(LD); p(NMPT,1); p(RS,ram); p(MDF,2); p(ZR,1); q(MBA,trnin,l*128+1);
    c(ST); p(NMPT,1); p(RS,ram); q(MBA,veti,d);
end;

{ Compoe a FFT do vetor real de 2*1*64 pontos usando as FFTs do vetor par e
  impar do vetor real }

procedure compfft (hini, gini, xini, tabmul, ctcom: string; l: word);
var
    i, l2, d : word;
begin
    ram := 0;
    l2 := 1 div 2;
    d := 12 * 128;
    c(DEFAULT); p(MBS,64); p(MSS,64); p(RV,0); p(CN,0); p(SH,0);
    vl;
    for i := 0 to l-1 do
        begin
            c(LD); p(NMPT,64); p(RS,ram); q(MBA,gini,i*128);
            c(MLTC); p(NMPT,64); p(RS,ram); s(MBA,tabmul);
            c(LD); p(NMPT,64); p(RS,ira); s(MBA,tabmul);
            c(ADDC); p(NMPT,64); p(RS,ram); p(SH,1); q(MBA,hini,i*128);
            c(MLTC); p(NMPT,64); p(RS,ira); p(CN,1); s(MBA,ctcom);
            c(ST); p(NMPT,64); p(RS,ram); q(MBA,xini,i*128);
            if i < (l-1)
            then begin
                c(ST); p(NMPT,64); p(RS,ira); s(MBA,tabmul);
            end;
        end;
    end;
end;

{*****}

```

Funcoes para geracao e manipulacao de tabelas de pesos W

```

*****}

{ Copia uma tabela (vetor) de 64 pontos }

procedure coptab (org, dst: string);
begin
  c(LD);      p(NMPT,64); p(RS,0); p(MBS,64); p(MSS,64); q(MBA,org,0);
  c(ST);      p(NMPT,64); p(RS,0); p(MBS,64); p(MSS,64); q(MBA,dst,0);
end;

{ Gera a tabela de pesos W para a multiplicacao intermediaria na FFT complexa.
  Esta tabela possui 1 coeficientes. }

procedure tabpesof (p, l: word);
var
  i, n      : word;
  ci, si    : integer;
begin
  n := l*64;
  for i := 0 to l-1 do
    begin
      ci := Round (cos(-i*p*2*pi / n) * $7FFF);
      si := Round (sin(-i*p*2*pi / n) * $7FFF);
      dw ('', strw (ci) + TAB + TAB + '/* ' + stri (p) + ' * ' + stri (i) + TAB + '*/');
      dw ('', strw (word (si)));
    end;
end;

{ Gera a tabela de pesos W para a composicao da FFT da parte par e impar do
  vetor de dados. Esta tabela possui 64 coeficientes. }

procedure tabpesoc (d, p, l: word);
var
  i, n      : word;
  ci, si    : integer;
  ang       : real;
begin
  n := l*64;
  for i := 0 to 63 do
    begin
      ang := 2*p*pi * (-i-d) / n;
      ci := Round (cos(ang) * $7FFF);
      si := Round (sin(ang) * $7FFF);
      dw ('', strw (ci) + TAB + TAB + '/* ' + stri (i+d) + TAB + '*/');
      dw ('', strw (word (si)));
    end;
end;

var
  m, i, nw      : word;

begin
  inig;
  nw := NLIN*64*2;
  vl;
  c('equ');      u('TamVet','',nw);
  c('equ');      t('Dados=0'); u('H','Dados',0); u('G','Dados',nw+2);
  c('equ');      u('Trans','G',nw+2); u('TabApod','',nw);
  c('equ');      t('IniProg=12300');
  vl;
  c('org');      t('IniProg');
  vl;
  c(DEFAULT); p(INTRP,0); p(EI,0); p(RV,0); p(MDF,3); p(ZR,0); p(ZP,0);
  c(DEFAULT); p(RBA,0); p(FSIZ,64); p(AS,0); p(PIF,0);
  vc ('Resetando o VSP');
  c(LDSM);      p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoIni');
  vc ('Definindo 2 secoes de memoria RAM');
  c(LDSM);      p(NMPT,1); p(UP,0); p(MD,1); s(MBA,'ModoNrm');
  vc ('Salva valor inicial da tabela de multiplicacoes da FFT');
  coptab ('TabResF', 'TabMulF');
  vc ('Salva valor inicial da tabela de multiplicacoes da composicao da FFT');
  coptab ('TabResC', 'TabMulC');
  {$IFDEF PREPOS}
  vc ('Multiplicando vetores');
  mulvetr ('Dados', 'TabApod', 'Dados', NLIN);

```

[illegible]

```
        tabpesof (i, NLIN);
vc ('Tabela de peso de referencia');
dw ('TabRefF', '');
for i := 0 to m-1 do
    tabpesof (m, NLIN);
vc ('Espaco livre para tabela de peso de multiplicacao');
dw ('TabMulF', '0');
close (alg);
end.
```

C. Rotinas de Controle para o MAq

```

unit placa2;

interface

const
    SEG_BASE      = $D000;      { Segmento da memoria a ser mapeado }
    BASE_ADC      = $0000;      { Offset da memoria a ser mapeado }
    BASE_MEM      = BASE_ADC;   { Offset da memoria local }
    BASE_PIO      = $0400;      { Offset da porta IO }
    BASE_CTD      = $0800;      { Offset do contador/divisor }
    BASE_STS      = $0C00;      { Offset do registro de status }

    { Offsets das portas IO }

    PORT0         = BASE_PIO + 0;
    PORT1         = BASE_PIO + 2;
    PORT2         = BASE_PIO + 4;
    PORTC         = BASE_PIO + 6;

    MODO_PIO      = $8A;        { Programa portas A e C1 saidas }

    { Offsets dos contadores/divisores }

    TIMER0        = BASE_CTD + 0;
    TIMER1        = BASE_CTD + 2;
    TIMER2        = BASE_CTD + 4;
    TIMERC        = BASE_CTD + 6;

    { Definicoes de frequencias usadas na placa (Hz) }

    CK_PC         = 14318180;
    CK_TIMER      = CK_PC / 4;

    { Offset do Registro de Controle do MAq, que e' a Porta 2 do PIO }

    REG_CTL       = PORT2;

    { Palavras de mascaramento dos bits no Reg. Ctrl }

    BITS_MD       = $03;
    RST_CTD       = $04;
    HAB_CTD       = $08;

    { Offset do Refistro de Status do MAq }

    REG_STS       = BASE_STS;

    { Palavras de mascaramento dos bits no Reg. Status }

    FIM_CTD       = $01;
    SOBR_CV       = $02;
    STS_ADC       = $04;

    { Modo default de operacao para o MAq }

    MODO_DEF      = $00;

{ Escreve uma palavra na memoria local }
procedure escMem (n: word);

{ Le uma palavra na memoria local }
function leMem: word;

{ Reseta o contador de enderecos da memoria local }
procedure rstMem;

{ Escreve byte na porta especificada }
procedure escPort (n: byte; p: word);

{ Le um byte da porta especificada }
function lePort (p: word): byte;

{ Le o modo de operacao corrente - 0, 1, 2, ou 3 }
function leModo: byte;

```

```

{ Seleciona o modo de operacao corrente - 0, 1, 2, ou 3 }
procedure poeModo (md: word);

{ Programa o modo de operacao do contador }
procedure iniCtd (md, ctd: byte);

{ Programa o valor de contagem/divisao do contador }
procedure carrCtd (ctg: word; ectd: word);

{ Le o valor do contador }
function leCtd (ectd: word): word;

{ Programa o numero de pontos para serem lidos }
procedure progNPtos (n: word);

{ Calcula a frequencia de aquisicao baseado no clock e nos divisores }
function freqAq: real;

{ Programa uma frequencia de aquisicao }
procedure progFreqAq (hz: real);

{ Le o Reg. de Status do MAq }
function leSts: byte;

implementation

var
    div0, div1      : word;

procedure escMem (n: word);
begin
    memw [SEG_BASE:BASE_MEM] := n;
end;

function leMem: word;
begin
    leMem := memw [SEG_BASE:BASE_MEM];
end;

procedure escPort (n: byte; p: word);
begin
    mem [SEG_BASE:p] := n;
end;

function lePort (p: word): byte;
begin
    lePort := mem [SEG_BASE:p];
end;

function leModo: byte;
begin
    leModo := lePort (REG_CTL) and BITS_MD;
end;

procedure poeModo (md: word);
begin
    escPort ((lePort (REG_CTL) and (not BITS_MD)) or md, REG_CTL);
end;

procedure rstMem;
var w      : word;
begin
    escPort (lePort (REG_CTL) and (not RST_CTD), REG_CTL);
    w := leMem;
    escPort (lePort (REG_CTL) or RST_CTD, REG_CTL);
end;

function leSts: byte;
begin
    leSts := mem [SEG_BASE:REG_STS] and $07;
end;

```

```

procedure iniCtd (md, ctd: byte);
var
  p          : byte;
begin
  p := (ctd shl 6) + (3 shl 4) + (md shl 1);
  mem [SEG_BASE:TIMERC] := p;
end;

procedure carrCtd (ctg: word; ectd: word);
begin
  mem [SEG_BASE:ectd] := lo (ctg);
  mem [SEG_BASE:ectd] := hi (ctg);
  if ectd = TIMER0
  then div0 := ctg
  else if ectd = TIMER1
  then div1 := ctg;
end;

function leCtd (ectd: word): word;
var c          : word;
begin
  c := mem [SEG_BASE:ectd];
  c := c + mem [SEG_BASE:ectd] * $100;
  leCtd := c;
end;

procedure progNPtos (n: word);
begin
  iniCtd (0, 2);           { Inicia ctd 2 no modo de contagem }
  carrCtd (n, TIMER2);     { Carrego o numero de pontos no ctd 2 }
end;

function freqAq: real;
begin
  freqAq := CK_TIMER / (div0 * div1);
end;

procedure progFreqAq (hz: real);
var div0, div1      : word;
    divt            : real;
begin
  divt := CK_TIMER / hz;
  if divt < $20000
  then begin
    div1 := 2;
    div0 := round (divt / 2);
  end
  else begin
    div0 := $FFFF;
    div1 := round (divt / div0);
  end;
  iniCtd (2, 0);
  carrCtd (div0, TIMER0);
  iniCtd (2, 1);
  carrCtd (div1, TIMER1);
end;

begin
  mem [SEG_BASE:PORTC] := MODO_PIO;
  mem [SEG_BASE:REG_CTL] := MODO_DEF;   { Modo op 0, reset contador ender }
end.

```


D. Rotinas de Controle para o MPr

```

unit vspelib;

interface

uses crt, vetdin;           { Usa unidade gerenciadora de vetores dinamicos }

const

    VSPMEMSEG      = $E000;      { Segmento de memoria mapeada no MPr }
    VSEREGSEG      = $E800;      { Segmento com os registros do MPr }
    INSTBASEMAP    = $306;      { Offset do Registro de Instrucoes }
    NEXTFETCH      = $300;      { Offset do Registro de Fetch }
    MODEREGMAP     = $300;      { Offset do Registro de Modo }
    STATUSREGMAP   = $301;      { Offset do Registro de Status }

    { Enderecos dos ports }

    VSPMODEPORT    = $300;
    VSPESTATPORT   = $300;

    { Mascaras de status }

    VSPEBDRESET    = $10;
    VSPEBDEXT      = $05;
    VSPEBDINT      = $02;

    { Padroes de controle }

    CARDRES        = $0;
    CARDREQ        = $31;
    RELEASEVSP     = $30;
    CARDDIS        = $B0;
    VSPECLRINT     = $11;

    { Reseta o VSP por software }
    function resetAll: boolean;

    { GetVSPmem: le conteudo da memoria do VSP }
    function GetVSPmem (add: word): word;

    { PutVSPmem: escreve em determinado endereco do VSP o valor desejado }
    procedure PutVSPmem (add: word; n: word);

    { GetVSPreg: le conteudo dos registros do VSP }
    function GetVSPreg (add: word): word;

    { PutVSPreg: escreve em determinado registro do VSP o valor desejado }
    procedure PutVSPreg (add: word; n: word);

    { CheckIntStat: verdadeiro se existe interrupcao pendente no PC }
    function CheckIntStat: boolean;

    { TakeBus: solicita barramento ao VSP e espera entrega, retornando estado }
    function TakeBus: boolean;

    { enableCard: torna o VSP visivel ao PC }
    procedure enableCard;

    { disableCard: retira o VSP do barramento do PC }
    procedure disableCard;

    { WaitForInt: aguarda pedido de interrupcao do VSP, qdo entrega-o ao PC }
    function WaitForInt: boolean;

    { releaseBus: permite que o VSP acesse a sua memoria, retirando-a do PC }
    procedure ReleaseBus;

    { startVSP: inicia o programa no endereco especificado, retornando status }
    function startVSP (add: word): boolean;

    { hardfini: reseta e desabilita o VSP }
    procedure hardfini;

    { readVSPsts: le o status do VSP e reseta as interrupcoes }

```

```

function readVSPsts: integer;

{ writeVSPmode: escreve no registro de modo do VSP }
procedure writeVSPmode (mode: integer);

{ leSinal: le sequencia de words da memoria do VSP e armazena em vetor }
procedure leSinal (psig: apt_vet_word; add, n: word);

{ escSinal: escreve sequencia de words na memoria do VSP }
procedure escSinal (psig: apt_vet_word; add, n: word);

implementation

{*****}
*
*   GetVSPmem: le conteudo da memoria do VSP
*
*   *****}
function GetVSPmem (add: word): word;
var a      : ^word;
begin
a := ptr (VSPMEMSEG, add*2);
GetVSPmem := a^;
end;

{*****}
*
*   PutVSPmem: escreve em determinado endereco do VSP o valor desejado
*
*   *****}
procedure PutVSPmem (add: word; n: word);
var a      : ^word;
begin
a := ptr (VSPMEMSEG, add*2);
a^ := n;
end;

{*****}
*
*   GetVSPreg: le conteudo dos registros do VSP
*
*   *****}
function GetVSPreg (add: word): word;
var a      : ^word;
begin
a := ptr (VSPMEMSEG, $8000 + add*2);
GetVSPreg := a^;
end;

{*****}
*
*   PutVSPreg: escreve em determinado registro do VSP o valor desejado
*
*   *****}
procedure PutVSPreg (add: word; n: word);
var a      : ^word;
begin
a := ptr (VSPMEMSEG, $8000 + add*2);
a^ := n;
end;

{*****}
*
*   resetAll: reseta VSP e coloca-o como escravo do PC
*
*   *****}
function resetAll: boolean;
begin
port [VSPMODEPORT] := CARDRES;
if (port [VSPESTATPORT] and $10) = $10
then
    resetall := false
else begin
    port [VSPMODEPORT] := CARDREQ;
    resetall := true;
end;
end;

```

```

end;

{*****
 *
 * CheckIntStat: verdadeiro se existe interrupcao pendente no PC
 *
 *****/}
function CheckIntStat: boolean;
begin
if (port [VSPESTATPORT] and $02) = $02
then
    CheckIntStat := true
else
    CheckIntStat := false;
end;

{*****
 *
 * TakeBus: solicita barramento ao VSP e espera entrega, retornando estado
 *
 *****/}
function TakeBus: boolean;
begin
port [VSPEMODEPORT] := CARDREQ;
delay (1);
if (port [VSPESTATPORT] and VSPEBDEXT) = $01
then
    TakeBus := true
else
    TakeBus := false;
end;

{*****
 *
 * enableCard: torna o VSP visivel ao PC
 *
 *****/}
procedure enableCard;
begin
port [VSPEMODEPORT] := RELEASEVSP;
end;

{*****
 *
 * disableCard: retira o VSP do barramento do PC
 *
 *****/}
procedure disableCard;
begin
port [VSPEMODEPORT] := CARDDIS;
end;

{*****
 *
 * WaitForInt: aguarda pedido de interrupcao do VSP, qdo entrega-o ao PC
 *
 *****/}
function WaitForInt: boolean;
var i
    : word;
begin
for i := 1 to 50000 do
    if CheckIntStat
    then begin
        if TakeBus
        then
            WaitForInt := true
        else
            WaitForInt := false;
        exit;
    end;
end;

end;

{*****
 *
 * releaseBus: permite que o VSP acesse a sua memoria, retirando-a do PC
 *
 *****/}

```

```

*
*****}
procedure ReleaseBus;
begin
port [VSPEMODEPORT] := RELEASEVSP;
end;

{*****}
*
* startVSP: inicia o programa no endereco especificado, retornando status *
*
*****}
function startVSP (add: word): boolean;
begin
if TakeBus
then
if (port [VSPESTATPORT] and VSPEBDINT) = 0
then begin
PutVSPreg (INSTBASEMAP, add);
ReleaseBus;
StartVSP := true;
end
else
StartVSP := false
else
StartVSP := false;
end;

{*****}
*
* hardfini: reseta e desabilita o VSP
*
*****}
procedure hardfini;
begin
resetAll;
disableCard;
end;

{*****}
*
* readVSPsts: le o status do VSP e reseta as interrupcoes
*
*****}
function readVSPsts: integer;
begin
TakeBus;
ReadVSPsts := GetVSPmem (STATUSREGMAP);
port [VSPEMODEPORT] := VSPECLRINT;
port [VSPEMODEPORT] := RELEASEVSP;
end;

{*****}
*
* writeVSPmode: escreve no registro de modo do VSP
*
*****}
procedure writeVSPmode (mode: integer);
begin
TakeBus;
PutVSPreg (MODEREGMAP, mode);
port [VSPEMODEPORT] := RELEASEVSP;
end;

{*****}
*
* leSinal: le sequencia de words da memoria do VSP e armazena em vetor
*
*****}
procedure leSinal (psig: apt_vet_word; add, n: word);
var i : word;
begin
takeBus;
for i := 0 to n-1 do
psig[i] := getVSPmem (add+i);
end;

```

```

{*****}
*
*   escSinal: escreve sequencia de words na memoria do VSP
*
*   *****}
procedure escSinal (psig: apt_vet_word; add, n: word);
var i
    : word;
begin
takeBus;
for i := 0 to n-1 do
    putVSPmem (add+i, psig[i]);
end;

end.

```

E. Esquemas do MAq

FOSC			
Título			
PLACA DE AQUISIÇÃO DE ALTA VELOCIDADE			
Size	Document Number	REV	
B	DIAGRAMA DE BLOCOS		
Date:	February 21, 1995	Sheet	1 of 8

IFOSC			
Title			
PLACA DE AQUISICAO DE ALTA VELOCIDADE			
Size	Document Number	REV	
A	BANCO DE MEMORIAS		
Date:	January 4, 1995	Sheet	4 of 8

IFOSC			
Title: PLACA DE AQUISIÇÃO DE ALTA VELOCIDADE			
Size: A	Document Number	LOGICA DE CONTROLE	REV
Date: February 21, 1995	Sheet	8	of 8

F. Equações da PAL Combinacional

TITLE PLACA AQUIS ALTA VEL
 PATTERN LOGICA COMBINACIONAL
 REVISION P1.03
 AUTHOR WALTER SOTO
 COMPANY IFQSC - OTICA
 DATE 2/07/94

; ESTE CIRCUITO IMPLEMENTA UMA LOGICA COMBINACIONAL PARA O CONTROLE DE
 ; BUFFERS, MEMORIAS E CONTADORES NA PLACA DE AQUISICAO DE ALTA VELOCIDADE

CHIP LOGCOMB PAL16L8

; PINS

;1	2	3	4	5	6	7	8	9	10
STSA	RD	WR	REQCV	CSAM	VE0	VE1	VE2	NC	GND
;11	12	13	14	15	16	17	18	19	20
NC	INCC	CSBA	CSBM	CSMEM	CV	SOBRCV	NC	WEMEM	VCC

; A LOGICA E' A DE UM MULTIPLEX, SENDO QUE PARA CADA ESTADO (INDICADO PELAS
 ; VE) UMA SAIDA DEVE SER SELECIONADA. ABAIXO ESTA O ESQUELETO DESTE MUX.

```
;
; /CV = /VE2 * /VE1 * /VE0 ; MD AQ PC S00
; + /VE2 * /VE1 * VE0 ; MD LE/ESC MEM S10
; + /VE2 * VE1 * VE0 ; MD AQ AUTO S20
; + VE2 * VE1 * /VE0 ; MD AQ AUTO S21
; + /VE2 * VE1 * /VE0 ; MD AQ AUTO S22
; + VE2 * /VE1 * /VE0 ; MD AQ AUTO S23
```

EQUATIONS

```
/CV = /VE2 * /VE1 * /VE0 * /CSAM * /WR ; MD AQ PC
+ /VE2 * VE1 * VE0 * /REQCV ; MD AQ AUTO 0

/CSBA = /VE2 * /VE1 * /VE0 * /CSAM * /RD ; MD AQ PC
+ /VE2 * VE1 * /VE0 ; MD AQ AUTO 2

/CSBM = /VE2 * /VE1 * /VE0 * /CSAM * /RD ; MD AQ PC
+ /VE2 * /VE1 * VE0 * /CSAM ; MD LE/ESC MEM

/CSMEM = /VE2 * /VE1 * VE0 * /CSAM ; MD LE/ESC MEM
+ /VE2 * VE1 * /VE0 ; MD AQ AUTO 2

/WEMEM = /VE2 * /VE1 * VE0 * /WR ; MD LE/ESC MEM
+ /VE2 * VE1 * /VE0 ; MD AQ AUTO 2

/INCC = /VE2 * /VE1 * VE0 * /CSAM ; MD LE/ESC MEM
+ /VE2 * VE1 * /VE0 ; MD AQ AUTO 2

/SOBRCV = /VE1 * /SOBRCV ; FLIP FLOP RS
+ VE0 * /SOBRCV ; S=/REQCV * VE1 * /VE0
+ REQCV * /SOBRCV ; R=/CSAM * /RD
+ /VE1 * /CSAM * /RD
+ VE0 * /CSAM * /RD
+ REQCV * /CSAM * /RD
```

```
; . . . . .
; . . . . .
```

SIMULATION

TRACE_ON CV CSBA CSBM CSMEM WEMEM INCC SOBRCV

; MODO AQUISICAO CONTROLADA PELO PC (SEM ACESSO A MEMORIA)

SETF /VE2 /VE1 /VE0

SETF /STSA RD WR REQCV CSAM

SETF /CSAM /RD ; RESETA FF DO SINAL EMCV
 SETF CSAM RD

SETF /RD ; LEITURA EM OUTRO ENDEREÇO
 SETF RD

```

SETF /WR                ; ESCRITA EM OUTRO ENDEREÇO
SETF WR

SETF /REQCV              ; DISPARO EXTERNO DO AD
SETF REQCV

SETF /CSAM /WR           ; DISPARO DO AD
SETF CSAM WR

SETF STSA                ; SIMULA CONVERSAO
SETF /RD                 ; LEITURAS EM OUTRO END
SETF RD
SETF /WR                 ; ESCRITAS EM OUTRO END
SETF WR
SETF /STSA

SETF /CSAM /RD           ; LEITURA DO AD
SETF CSAM RD

; MODO LEITURA / ESCRITA NA MEMORIA EXTERNA

SETF /VE2 /VE1 VEO      ; MD LE/ESC MEM
SETF /STSA RD WR REQCV CSAM

SETF /CSAM /RD           ; RESETA FF DO SINAL EMCV
SETF CSAM RD

SETF /RD                 ; LEITURA EM OUTRO ENDEREÇO
SETF RD

SETF /WR                 ; ESCRITA EM OUTRO ENDEREÇO
SETF WR

SETF /REQCV              ; TENTA DISPARO EXTERNO DO AD
SETF REQCV

SETF /CSAM /WR           ; ESCRITA EM MEMORIA
SETF CSAM WR

SETF STSA                ; SIMULA CONVERSAO
SETF /RD                 ; LEITURAS EM OUTRO END
SETF RD
SETF /WR                 ; ESCRITAS EM OUTRO END
SETF WR
SETF /STSA

SETF /CSAM /RD           ; LEITURA DA MEMORIA
SETF CSAM RD

; MODO AQUISICAO AUTOMATICA

;          ESTADO S20 - ESPERA STATUS = 1

SETF /VE2 VE1 VEO      ; MD S20
SETF /STSA RD WR REQCV CSAM

SETF /RD                 ; LEITURA EM OUTRO ENDEREÇO
SETF RD

SETF /WR                 ; ESCRITA EM OUTRO ENDEREÇO
SETF WR

SETF /REQCV              ; DISPARO EXTERNO DO AD
SETF REQCV

SETF STSA                ; SIMULA CONVERSAO

;          ESTADO S21 - ESPERA STATUS = 0

SETF VE2 VE1 /VEO

SETF /RD                 ; LEITURAS EM OUTRO END
SETF RD

SETF /WR                 ; ESCRITAS EM OUTRO END
SETF WR

SETF /STSA

```



```

;          ESTADO S22 - GRAVA NA MEMORIA
SETF /VE2 VE1 /VE0

SETF /REQCV          ; DISPARO EXTERNO DO AD - ERRO
SETF REQCV

;          ESTADO S20 - ESPERA STSA = 1
SETF /VE2 VE1 VE0

SETF /CSAM /RD          ; TENTA LEITURA DA MEMORIA
SETF CSAM RD

SETF /WR          ; ESCRITA EM OUTRO ENDEREÇO
SETF WR

;          ESTADO S30 - ESPERA O TRIGGER
SETF VE2 /VE1 VE0

SETF /CSAM /RD
SETF CSAM RD

SETF /REQCV
SETF REQCV

;          ESTADO S24 - ESPERA O TRIGGER
SETF VE2 VE1 VE0

SETF /CSAM /RD
SETF CSAM RD

SETF /REQCV
SETF REQCV

TRACE_OFF

```

G. Equações da PAL Sequencial

```
TITLE    PLACA_AQUIS_ALTA_VEL
PATTERN  LOGICA_SEQUENCIAL
REVISION P1.03
AUTHOR   WALTER_SOTO
COMPANY  IFQSC - OTICA
DATE     4/07/94
```

```
; ESTE CIRCUITO IMPLEMENTA UMA LOGICA SEQUENCIAL PARA A MAQUINA DE ESTADOS
; QUE CONTROLA OS MODOS DE AQUISICAO. PODE TAMBEM IMPLEMENTAR UMA LOGICA
; COMBINACIONAL POIS TEM 2 SAIDAS COMBINACIONAIS NAO USADAS.
```

```
CHIP LOGSEQ PAL16R6
```

```
;PINS
;1      2      3      4      5      6      7      8      9      10
CLK      STSA    FIMC    MD1     MD0     RST     TG      NC      NC      GND
;11     12     13     14     15     16     17     18     19     20
CSPAL    NC      VE2     VE1     VE0     NC      NC      NC      NC      VCC
```

```
; AS ENTRADAS MD1 E MD0 SELECIONAM O MODO DE OPERACAO DA PLACA. SAO 4 MODOS,
; A SABER:
```

```
;
;      MD1 MD0 MODO      DESCRICAO
;      0  0  0        MODO AQUISICAO INDIVIDUAL
;      0  1  1        MODO ACESSO A MEMORIA LOCAL
;      1  0  2        MODO AQUISICAO AUTOMATICA
;      1  1  3        MODO AQUISICAO AUTOMATICA DISPARADO POR EVENTO
```

```
; CADA MODO TEM O SEU DIAGRAMA DE ESTADOS PROPRIO. OS MODOS 0, 1 E 3 POSSUEM
; APENAS 1 ESTADO. O MODO 2 POSSUI 5 ESTADOS. CADA ESTADO E' REPRESENTADO
; PELAS VARIAVEIS DE ESTADO, CUJO VALOR E' ESCOLHIDO PARA MINIMIZAR AS
; TRANSICOES ESPURIAS DE BITS QUNADO DA MUDANCA DE ESTADOS (CODIGO GRAY).
```

```
;
;      MODO      ESTADO  VEST      DESCRICAO
;      0          S00     000      ESCRITA INICIA CONVERSAO, LEITURA LE AD
;      1          S10     001      ESCRITA/LEITURA MEMORIA, RST ENDER E INC ENDER
;      2          S20     011      ESPERA INICIO CONVERSAO
;          S21     110      ESPERA FIM CONVERSAO E TESTE FIM DE AQUIS
;          S22     010      LE AD, ESCRIVE NA MEMORIA E INC ENDER
;          S23     100      FIM DA AQUISICAO
;          S24     111      ESPERA TRIGGER PARA INICIO DA CONVERSAO AUTOM
;      3          S30     101      RESETA O FF DE ESPERA DE TRIGGER
```

```
EQUATIONS
```

```
/VE0      := /MD1 * /MD0                                ; MD0
+ MD1 * /MD0 * /VE2 * VE1 * VE0 * STSA ; MD2 S20 STSA
+ MD1 * /MD0 * VE2 * VE1 * /VE0          ; S21
+ MD1 * /MD0 * VE2 * /VE1 * /VE0          ; S23
+ RST

/VE1      := /MD1 * /MD0                                ; MD0
+ /MD1 * MD0                                           ; MD1
+ MD1 * /MD0 * VE2 * VE1 * /VE0 * /STSA *             ; MD2 S21 FIMC
+ FIMC
+ MD1 * /MD0 * VE2 * /VE1 * /VE0                    ; MD2 S23
+ MD1 * MD0
+ RST

/VE2      := /MD1 * /MD0                                ; MD0
+ /MD1 * MD0                                           ; MD1
+ MD1 * /MD0 * /VE2 * /VE1 * VE0                    ; MD2 S10
+ MD1 * /MD0 * /VE2 * VE1 * VE0 * /STSA ; MD2 S20
+ MD1 * /MD0 * VE2 * VE1 * /VE0 * /STSA *             ; MD2 S21 /STSA /FIMC
+ /FIMC
+ MD1 * /MD0 * /VE2 * VE1 * /VE0                    ; MD2 S22
+ MD1 * /MD0 * VE2 * VE1 * VE0 * TG ; MD2 S24
+ RST
```

```
; . . . . .
; . . . . .
```

```
SIMULATION
```

```

TRACE_ON VE2 VE1 VE0 GATE

SETF /CLK /STSA /FIMC /RST /CSPAL /TG          ; CONDICAO INICIAL
CLOCKF CLK

SETF RST
CLOCKF CLK
SETF /RST

SETF /MD1 /MD0                                ; POE MODO 0
CLOCKF CLK

SETF /MD1 MD0                                ; POE MODO 1
CLOCKF CLK

SETF MD1 /MD0 /STSA                          ; POE MODO 2
CLOCKF CLK

; FAZ UMA AQUISICAO, ARMAZENANDO NA MEMORIA E CONTINUANDO O PROCESSO

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK                            ; ESPERANDO CONVERSAO
    END

SETF STSA                                    ; INICIO CONVERSAO

FOR I := 1 TO 6 DO
    BEGIN
        CLOCKF CLK                            ; ESPERA FIM DE CONVERSAO
    END

SETF /STSA                                    ; FIM DA COVERSAO

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK                            ; ESCRIVE NA MEMORIA E
        ; VOLTA A ESPERAR COVERSAO
    END

; FAZ A ULTIMA AQUISICAO, TERMINANDO O PROCESSO

SETF FIMC                                    ; ATIVA INDICADOR DE FIM

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK                            ; ESPERANDO CONVERSAO
    END

SETF STSA                                    ; INICIO CONVERSAO

FOR I := 1 TO 6 DO
    BEGIN
        CLOCKF CLK                            ; ESPERA FIM DE CONVERSAO
    END

SETF /STSA                                    ; FIM DA COVERSAO

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK                            ; VAI PARA O ESTADO FINAL
    END

SETF /FIMC

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK
    END

SETF /MD1 MD0                                ; APOS O FIM, RETORNA A MD1

FOR I := 1 TO 4 DO
    BEGIN
        CLOCKF CLK
    END

; FAZ UMA AQUISICAO, MUDANDO DE ESTADO NO MEIO DELA E NAO NO FIM

SETF /FIMC MD1 /MD0 /STSA                    ; DESATIVA INDICADOR DE FIM

```

```

FOR I := 1 TO 4 DO
  BEGIN
    CLOCKF CLK                ; ESPERANDO CONVERSAO
  END

SETF /MD1 /MD0                ; FORCANDO VOLTA A MD0

FOR I := 1 TO 6 DO
  BEGIN
    CLOCKF CLK
  END

; PASSAGEM DO MODO MD1 PARA O MODO MD2 (01 -> 10) NOS SEUS 2 CAMINHOS

SETF /MD1 MD0                ; 01 -> 11 -> 10

CLOCKF CLK

SETF MD1 MD0

CLOCKF CLK

SETF MD1 /MD0

CLOCKF CLK

SETF /MD1 MD0                ; 01 -> 00 -> 10

CLOCKF CLK

SETF /MD1 /MD0

CLOCKF CLK

SETF MD1 /MD0

CLOCKF CLK

; TESTA O MODO DE AQUISICAO AUTOMATICA COM DISPARO INICIAL POR TRIGGER

SETF MD1 MD0                ; RESETANDO GATE

CLOCKF CLK

SETF TG                      ; IGNORA TRIGGER

CLOCKF CLK

SETF /TG

CLOCKF CLK

SETF MD1 /MD0                ; ESPERA TRIGGER

FOR I := 1 TO 4 DO
  BEGIN
    CLOCKF CLK
  END

SETF TG                      ; OCORRE TRIGGER

FOR I := 1 TO 4 DO
  BEGIN
    CLOCKF CLK
  END

SETF /TG

CLOCKF CLK

TRACE_OFF

```

H. Esquema do Circuito de Disparo de Aquisição

I. Programa de Teste do MAq

```

program modo3;

uses
  crt, graph,
  jangraf,      { Funcoes basicas de controle de janelas graf. }
  placa2,       { Rotinas de controle do MAq }
  vetdin,       { Rotinas de gerenciamento de vetores dinamicos }
  arqbin;       { Rotinas para manipular arquivos binarios }

const
  N_PTOS        = 8*1024;
  EXT_ARQ       = '.TXT';
  NOME_ARQ      = 'MOD03';

  { Controle do disparo da aquisicao no modo 3 }
  RST_TG        = 0;
  HAB_TG        = 1;

var
  i, w          : word;
  sts           : byte;
  pv            : apt_vet_word;

begin
  alloc_word (pv, N_PTOS);      { Aloca vetor na heap }
  escPort (RST_TG, PORT0);     { Desabilita o disparo externo }

  progFreqAq (100000);         { Freq. aquis. de 100 KHz }
  progNPTos (N_PTOS);         { Programa numero de ptos p/ leitura }

  poeModo (1);                 { Poe a placa no modo acesso a memor. }
  rstMem;                      { Zera o contador de endereco da mem. }

  poeModo (3);                 { Espera o sinal de trigger de inicio }
  poeModo (2);                 { Prepara a aquisicao automatica }
  escPort (HAB_TG, PORT0);     { Habilita o disparo de inicio de aq. }
  repeat until (leSts and FIM_CTD) <> 0; { Testa fim da aquisicao automatica }
  sts := leSts;                { Le o status pois o modo 1 zera sts }

  iniciaGrafico ('c:\tp\bgi'); { Poe PC no modo grafico }
  defineMundo (0, 4095, N_PTOS, 0); { Define o mundo para a janela }
  ativaJanGraf (0, 0, getMaxX, getMaxY, true); { Define as coordenadas }

  poeModo (1);                 { Poe no modo de leitura de memoria }
  rstMem;                      { Inicia o contador de enderecos }
  for i := 0 to N_PTOS-1 do    { Le e plota resultados da aquisicao }
  begin
    pv^[i] := leMem div 16;     { Elimina os 4 sinais digitais lidos }
    plotaPonto (i, pv^[i]);
  end;

  escAscIntg (NOME_ARQ+EXT_ARQ, pv, N_PTOS); { Salva resultado aquisi. }
  repeat until keypressed;

  poeModoTexto;
  id (sts and SOBR_CV) <> 0    { Testa se perdeu alguma conversao }
  then writeln ('Houve perda de conversao !');
  libera;                     { Libera memoria alocada p/ vetores }
end.

```

J. Programa de Teste do MPr

```

program procVSP;

uses
  crt, graph,
  jangraf,           { Tratamento de janelas graficas           }
  eixosleg,          { Plota eixos e legendas                     }
  vetdin,            { Gerencia vetores dinamicos                 }
  arqbin,            { Manipula arquivos binarios e de texto      }
  arqhex,            { Le e traduz arquivos no formato hexa      }
  vspelib,           { Rotinas de controle do VSP                 }
  apodiz;            { Funcoes de apodizacao pre-definidas       }

const
  ARQ_PROG           = 'D:\TATUI\VSPA\PROG.HEX';
  ARQ_RES            = 'D:\TATUI\SW\FINAL\RESFFT.TXT';
  N_PTOS             = 4096;
  LARG_PULSO         = 8;

  INI_INTERF         = 0;
  INI_APOD           = N_PTOS;
  INI_ESP_CPLX       = 0;
  INI_ESP_POT        = N_PTOS;
  INI_PROG           = 12500;

{ Simula a aquisicao de um sinal tipo pulso }
procedure aqSinal (pv: apt_vet_intg; n: word);
var i
  f
    : word;
    : real;
begin
  for i := 0 to n-1 do
    begin
      if i < LARG_PULSO
      then pv[i] := $7FFF
      else pv[i] := 0;
    end;
  end;

{ Plota um vetor de numeros inteiros }
procedure plotaInt (pv: apt_vet_intg; n: word);
var i
    : word;
begin
  poeModoGraf;
  definemundo (0, $7FFF, n-1, -$7FFF);
  ativajangraf (0, 0, getmaxx, getmaxy, false);
  desenhabordas;
  plotaeixos (10, 2, '', '', linnormal, linnormal);
  for i := 0 to n-1 do
    plotaPonto (i, pv[i]);
  end;

{ Plota um vetor de numeros complexos }
procedure plotaCplx (pv: apt_vet_intg; n: word);
var i
    : word;
begin
  n := n div 2;
  poeModoGraf;
  definemundo (0, $7FFF, n-1, -$7FFF);
  ativajangraf (0, 0, getmaxx, getmaxy, false);
  desenhabordas;
  plotaeixos (10, 2, '', '', linnormal, linnormal);
  for i := 0 to n-1 do
    plotaPonto (i, pv[i*2]);
  end;

var
  sig,
  apd,
  esp,
  pot
    : array [0..N_PTOS-1] of integer;

begin
  iniGraf;
  { Poe PC no modo grafico }

```



```

resetAll;                                { Reseta VSP }
takeBus;                                  { Captura a memoria do VSP }
progVSP (ARQ_PROG);                       { Programa o VSP com o arquivo hexa }

pesosApd (RETANG, @apd, N_PTOS);          { Monta tabela de pesos de apodiz }

resetAll;
aqSinal (@sig, N_PTOS);                   { Cria o sinal a ser processado }
escSinal (@sig, INI_INTERF, N_PTOS);      { Escreve o sinal a ser proces. }
escSinal (@apd, INI_APOD, N_PTOS);        { Escreve os pesos de apodizacao }

startVSP (INI_PROG);                      { Inicia a execucao do programa }
if waitForInt                             { Espera fim de execucao }
then begin
    leSinal (@esp, INI_ESP_CPLX, N_PTOS); { Le o espectro }
    leSinal (@pot, INI_ESP_POT, N_PTOS);   { Le o espectro de potencia }
end;

plotaCplx (@esp, N_PTOS);                  { Plota o espectro complexo }
delay (1000);
plotaInt (@pot, N_PTOS div 2);             { Plota o espectro de potencia }
delay (1000);

escAscIntg (ARQ_RES, @esp, N_PTOS);       { Exporta resultado p/ arquivo ASCII }
end.

```