

UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA
Departamento de Engenharia Electrotécnica e computadores

**Análise comparativa de sistemas informáticos de
instrumentação virtual.**

**Desenvolvimento e controlo de instrumentos virtuais
(Relatório de estágio PRODEP)**

**Isabel Cecilia Praça
João carlos Pinto de Almeida**

Prodep 92/93



1621.3 (C423) / LEEC / 392 / PAA / Vol. 1
06 / 10 / 09

UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA

Departamento de Engenharia Electrotécnica e computadores

**Análise comparativa de sistemas informáticos de
instrumentação virtual.
Desenvolvimento e controlo de instrumentos
virtuais
(Relatório de estágio PRODEP)**

**Isabel Cecilia Praça
João carlos Pinto de Almeida**

**Orientados pelo
Prof. Artur Cardoso**

Prodep 92/93

Índice:

1-Introdução	1
1.1-Descrição do "LabWindows"	1
1.2-Interface GPIB	2
1.3-Desenvolvimento do software	3
1.4-Módulo do instrumento	3
1.5-Módulo gráfico	4
1.6-Programa final	4
2-Campos de aplicação	5
3-Programa desenvolvido para o multímetro PM2519/51 da Philips	6
3.1-Descrição do instrumento	6
3.1.1- Programação da interface	6
3.1.2-Programação do multímetro	6
3.1.3-Leitura das medidas do multímetro	6
3.2-Descrição do programa	7
4-Programa desenvolvido para um osciloscópio digital DL1200A da Yokogawa	11
4.1-Descrição do instrumento	11
4.2-Descrição do programa	11
5-Conclusões	17
Bibliografia	18
Anexos	19
Anexo I	19
Anexo II	21
Anexo III	27
Anexo IV	38
Anexo V	51

1-Introdução

O objectivo deste trabalho foi o desenvolvimento de software que permita o controlo de instrumentação com interface GPIB através de um IBM PC (ou compatível). Os instrumentos usados foram um multímetro da Philips PM2519, e um osciloscópio Yokogawa DL1200. Os programas foram desenvolvidos em ambiente de programação "LabWindows".

1.1-Descrição do "LabWindows"

Trata-se de um programa especialmente concebido para concepção de aplicações de instrumentação, testes e aquisição de dados. Permite o desenvolvimento de programas quer em linguagem C quer em QuickBasic.

Permite: comunicação por GPIB e RS-232, aquisição e armazenamento de dados, formatação de dados, análise e aquisição de dados, manipulação e criação de gráficos. Para tal possui as seguintes livrarias: livraria gráfica, de análise, de formatação e entradas saídas, de GPIB, de GPIB-488.2, de RS-232, de interface com o utilizador ("User Interface"), e do sistema. De seguida descrevem-se mais especificamente as funções de algumas destas livrarias.

A livraria "User Interface" permite operar os painéis criados com o auxílio do "User Interface Editor", assim como apresentar gráficos. O "User Interface Editor" possibilita a criação de painéis, menus, caixas de diálogo, etc.. De seguida apresenta-se um exemplo de um painel criado através desta interface.

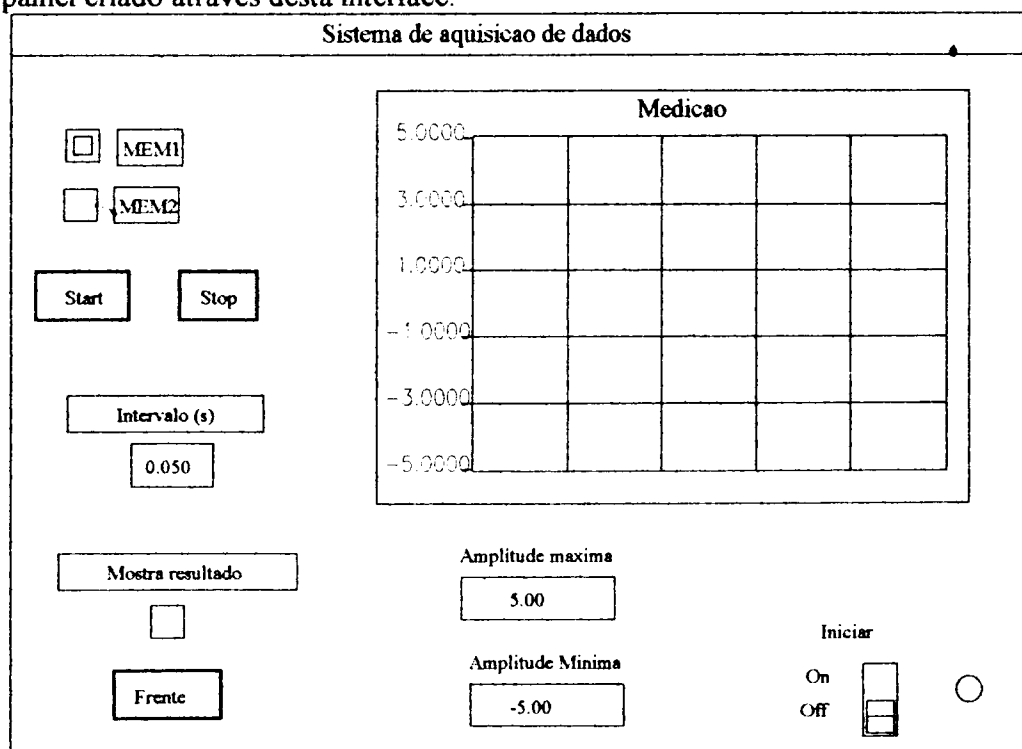


Fig 1

A livreria de formatação e entrada/saída possui funções que permitem formatar buffers de acordo com o especificado (por exemplo a passagem de números inteiros para strings ou vice-versa), e funções de manipulação de strings, e operação de ficheiros.

Para apresentar gráficos sem recorrer ao uso de painéis existem funções específicas na livreria gráfica, que apresentam um código mais simples, mais pequeno e mais rápido relativamente aos desenvolvidos juntamente com painéis.

É possível realizar operações com números complexos, matrizes, cálculos estatístico, etc..., com o auxílio das funções da livreria de análise.

A livreria GPIB permite configurar a interface e o aparelho, ler ou enviar dados, e controlar o aparelho, o barramento e a placa GPIB.

Existem ainda livrerias com funções já desenvolvidas para aplicar a instrumentos específicos.

1.2-Interface GPIB

A interface GPIB é definida pelas normas IEEE 488.1 e 488.2.

A norma 488.1 é de 1974 e apenas define os aspectos mecânicos, eléctricos e funcionais da interface por forma a que seja possível a compatibilidade entre os aparelhos dos diferentes fabricantes.

Os dois aparelhos usados neste trabalho, seguem a norma 488.1. De seguida são feitas algumas considerações gerais sobre esta norma.

O standard 488.1 aplica-se a sistemas de interface que permitem a interligação de instrumentos num sistema de instrumentação, segundo as seguintes características: os dados devem ser exclusivamente digitais, o número máximo de dispositivos não pode ser superior a 15, o comprimento máximo nos cabos de ligação deve ser inferior a 20m e o débito máximo de informação não poderá exceder o 1 Mb/s.

Neste tipo de interface, existem dois tipos de mensagens: mensagens usadas para o controlo da interface e mensagens destinadas aos aparelhos do sistema (este tipo de mensagens designam-se por dependentes do dispositivo). As funções específicas a cada aparelho não são definidas por este standard.

Para que as mensagens sejam trocadas de forma não ambigua é necessário que os aparelhos possuam as capacidades de actuar como "listener", "talker" e "controller".

Um aparelho com a capacidade de "listener" pode ser endereçado pela interface por forma a que possa receber comandos ou mensagens dependentes do dispositivo. Ao funcionar como "talker", uma vez endereçado pela interface pode enviar mensagens dependentes do dispositivo para outro instrumento. Um dispositivo a actuar como controlador pode endereçar outros aparelhos por forma a que estes possam receber ou enviar mensagens (um dispositivo apenas com esta característica não pode receber nem enviar mensagens dependentes do dispositivo).

São também possíveis os seguintes procedimentos:

1) "Serial Poll" - quando um dispositivo (com função de "talker") necessita de alguma acção do controlador envia uma mensagem de pedido de serviço. O controlador interroga todos os dispositivos para saber qual deles necessita de serviço.

- 2) "Parallel Poll" - o controlador interroga simultaneamente todos os dispositivos, os quais lhe envia no barramento de dados um bit a informar se necessitam de serviço ou não. Esta função só permite que oito dispositivos em simultâneo respondam à interrogação.
- 3) A inicialização e disparo do dispositivo.
- 4) Programação do dispositivo quer pelo painel frontal quer através do barramento (Remote / Local).

A configuração do sistema (aparelho e controlador) foi efectuada usando o package "PC-GPIB" e recorrendo-se ao uso do ficheiro executável "Ibconf". Esta configuração é feita ao nível de endereços de "talker" e "listener", do tempo para efectuar operações de entrada/saída ou de ocorrência de um determinado evento (timeout), do byte de fim de string usado, e o que ocorrerá na sua recepção ou quando será enviado.

1.3-Desenvolvimento do software

O desenvolvimento do software de controlo do dispositivo necessita de fundamentalmente duas fases, o módulo do instrumento e o programa propriamente dito, com painéis de apresentação que fazem uso das funções elaboradas no módulo do instrumento, e que irá ser manobrado pelo utilizador. Iremos de seguida expor o método adoptado para a sua realização.

1.4-Módulo do instrumento

O módulo do instrumento consiste de funções que estabelecem e encerram a comunicação, bem como funções que permitam extrair informação, configurar, e operar o aparelho. Estas funções existem no programa `core_gpb`, que é a estrutura em torno da qual se irão desenvolver as rotinas necessárias a cada dispositivo em concreto.

O `core_gpb` é constituído por uma rotina básica de início e outra de fecho da comunicação com o dispositivo, bem como de outras rotinas utilitárias descritas a seguir.

Para inicializar a comunicação, é usada a função `"ibfind ( )"`, que estabelece a comunicação e retorna um inteiro como descritor do aparelho. O encerramento da comunicação é feito através da função `"ibonl ( )"`.

Outras rotinas usadas são as de leitura e escrita do dispositivo, que permitem trocar a informação através de um buffer de caracteres, ou directamente para um ficheiro.

Existem ainda duas rotinas para avaliar se um número inteiro ou real está situado entre os limites especificados, e que se torna bastante útil nomeadamente quando existem gamas limitadas de variação de certos parâmetros.

É possível verificar se o dispositivo já foi fechado ou não, usando para tal uma destas rotinas utilitárias já desenvolvida.

Todas as rotinas encontram-se organizadas hierárquicamente numa árvore, podendo estar ou não agrupadas, e incluir ou não funções secundárias.

Cada rotina tem um painel associado, que permite operá-la e escolher, se for caso disso, os parâmetros a enviar.

1.5-Módulo gráfico

Com o auxílio do "User Interface Editor" é possível criar painéis com botões, caixas de selecção e gráficos, que permitem ao utilizador seleccionar as aplicações pretendidas de um modo transparente.

1.6-Programa final

Faz a interligação do módulo gráfico com o módulo do instrumento, resultando num programa que irá ser usado pelo utilizador para trabalhar sobre o instrumento.

A estrutura do programa é constituída por uma parte de verificação das opções seleccionadas pelo utilizador e do desencadeamento das funções respectivas. E outra parte constituída pelo desenvolvimento dessas funções que apresentam caixas para diálogo com o utilizador, e recorrem às funções do módulo do instrumento para desencadear as acções desejadas no dispositivo.

2-Campos de aplicação

Hoje em dia torna-se cada vez mais usual a implementação de sistemas de instrumentação baseados na interface GPIB. Uma das suas maiores aplicações é nos sistemas de teste automático.

Isto deve-se às vantagens que este tipo de interface oferece, quanto à realização automática de medidas. As vantagens são:

- maior qualidade nas medidas, a probabilidade de erro será bastante reduzida devido à utilização de meios automáticos.

- redução dos custos, uma vez que estes sistemas poderão ser manobrados por técnicos menos qualificados; além disso a automação introduzida permite a redução do tempo necessário à realização de uma determinada tarefa.

Programas como o LabWindows facilitam e reduzem o tempo de desenvolvimento dos programas de controlo destes sistemas.

Um sistema deste tipo, é na generalidade, constituído por, um computador com interface GPIB, ligado a vários instrumentos que controlam interruptores e dispositivos de entrada/saída que efectuem medidas sobre o dispositivo em teste.

Alguns exemplos deste tipo de sistemas são o teste do comportamento de semi-condutores de potência em relação à temperatura, teste de duração dos interruptores, teste de stress das lâmpadas de halógeneo, etc...

Existem vários tipos de instrumentos que usam este tipo de interface, nomeadamente, osciloscópios, multímetros, sistemas de aquisição de dados, geradores de forma de onda arbitrárias, sistemas modulares de interruptores e de entrada e saída, etc...

3-Programa desenvolvido para o multímetro PM2519/51 da Philips

3.1-Descrição do instrumento

O multímetro PM2519 é um instrumento já antigo, cuja interface GPIB segue a norma IEEE 488.1. A sua programação inclui funções que permitem programar a interface e o dispositivo.

São feitas abaixo algumas considerações gerais acerca dos comandos e informação disponibilizada pelo dispositivo.

3.1.1- Programação da interface

A estrutura das mensagens a enviar deverá incluir um cabeçalho, corpo e separador.



Fig. 2

Um dos comandos existentes é o "ID?", que pede ao aparelho a sua identificação, e que no programa desenvolvido foi usado para, após se ter estabelecido comunicação com o dispositivo verificar se esta se estava estabelecer correctamente.

É também possível pedir um serviço ao dispositivo através do comando "MSR n [n] [n]", em que n [n] [n] é o equivalente decimal da máscara.

Usando "SPR nn [,nn]" , poder-se-á estabelecer qual o carácter a usar como separador, o qual é seleccionado pelo equivalente decimal de nn na tabela de caracteres ISO. O carácter usado como separador permite separar vários comandos, enviados em série, sendo atendidos pelo multímetro por ordem de chegada.

Outro comando é o "TSI" que permite fazer um teste à interface.

3.1.2-Programação do multímetro

A mensagem de programação consiste em uma ou mais unidades, sendo as unidades independentes e constituídas por cabeçalho, corpo e separador. A execução da mensagem é de acordo com a sequência de entrada, excepto no caso do comando de "execute" que é executado no fim da mensagem.

As mensagens possíveis de enviar ao dispositivo relacionam-se com comandos de início de medidas ("X1"), velocidade ("V0", "V1"), gama de variação ("R0", "R1", ..., "R5"), modos de trigger ("T1", "T2"), e referência de zero ("Z1", ..., "Z5").

As mensagens mencionadas estão incluídas numa opção do menu do programa desenvolvido, sendo assim possível ao utilizador configurar o multímetro.

A selecção do tipo de medida a efectuar (Volts, Amperes, etc...) é efectuada manualmente através do comando do painel do multímetro.

3.1.3-Leitura das medidas do multímetro

A informação de saída do multímetro segue a estrutura já mencionada de cabeçalho, corpo e separador.

O cabeçalho é constituído por duas partes : função (discrimina qual o tipo de medida efectuada) e informação adicional (indica se a medida foi efectuada com referência de zero relativa e se excedeu os limites ou factor de crista).

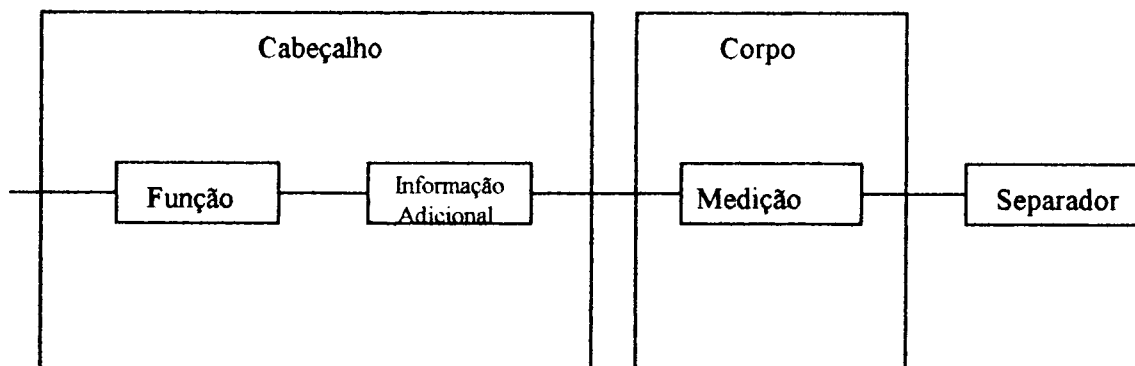


Fig. 3

3.2-Descrição do programa

O objectivo deste programa denominado "AQUIDAT" foi simular um sistema de aquisição de dados provenientes do multímetro, e mostrá-los sequencialmente num gráfico elaborado para o efeito. Para isso foi desenvolvido um módulo do instrumento que contém as funções a serem utilizadas em programas que usem este multímetro como parte de um sistema de instrumentação, e que recorrem a painéis gráficos associados a essas funções. O módulo do instrumento foi desenvolvido com base no programa `coré_gpb`, em volta do qual foram construídas as funções específicas relativas ao dispositivo usado.

A árvore das funções, construídas para permitir operar o aparelho, é neste caso constituída por apenas quatro funções.

- Inicialização
- Configurar
- Leitura
- Fechar

A primeira é a de abertura de comunicação com o dispositivo, e inclui um pedido de identificação ao aparelho. O código desta função é apresentado de seguida.

```
void PM2519_init ()
{
    bd = 0;
    bd = ibfind("PM2519");

    /*         pede identificação         */

    if (PM2519_write_data ("ID?", 3) != 0)
        return;
    if (PM2519_read_data (cmd, 50) != 0)
```

```
        return;
    Scan (cmd, "PM2519C:S1");
    if (NumFmtdBytes () != 10)
    {
        PM2519_err = 223;
        return;
    }
} /* fim de init */
```

Seguindo a sua ordem na árvore, apresentamos agora a função que permite configurar o multímetro quanto ao modo de trigger, velocidade, referência de zero e gama das medidas.

```
void PM2519_config(velocidade, range, trig, zref)
char *velocidade;
char *range;
char *trig;
char *zref;
{
    if (PM2519_device_closed()) return;
    if ( PM2519_write_data(velocidade,2) ) return;
    if ( PM2519_write_data(range,2) ) return;
    if ( PM2519_write_data(trig,2) ) return;
    if ( PM2519_write_data(zref,2) ) return;
    PM2519_err = 0;
}
```

A função apresentada a seguir é a de leitura, com a qual, após testar que o dispositivo não está fechado, lê o valor presente no display do multímetro, separando a parte de identificação do tipo de leitura efectuado do valor da leitura em si, guardando-os em variáveis distintas.

```
void PM2519_leitura(tipo, valor)
char *tipo;
double *valor;
{
    if (PM2519_device_closed()) return;
    if ( PM2519_read_data(cmd,19) ) return;
    Scan(cmd,"%s>%s%f",tipo,valor);
}
```

Por fim, apresenta-se a rotina de encerramento de comunicação, em que se verifica se o dispositivo já está fechado. Em caso da resposta ser negativa ele é de facto encerrado.

```
void PM2519_close ()
{
    /* verifica se o dispositivo já está fechado */

    if (PM2519_device_closed())
        return;

    /* Fecha o dispositivo */
}
```

```
if ( !bon1 (bd,0) < 0)
    PM2519_err = 221;
else
    PM2519_err = 0;

bd = 0;
}
```

As funções, acima definidas são utilizadas através do painel de interface com o utilizador, de modo a que a operação com o multímetro se torne de certo modo transparente.

Para criar a interface gráfica com o utilizador, foi construído um painel com o auxílio da ferramenta "User Interface Editor", descrita no manual com o mesmo nome.

Foi elaborado um painel com um botão de inicialização, que permite o início da comunicação com o multímetro, usando para tal a função acima descrita "PM2519_init()"; colocando este botão no estado Off, permite ainda encerra-se a comunicação recorrendo à função "PM2519_close()".

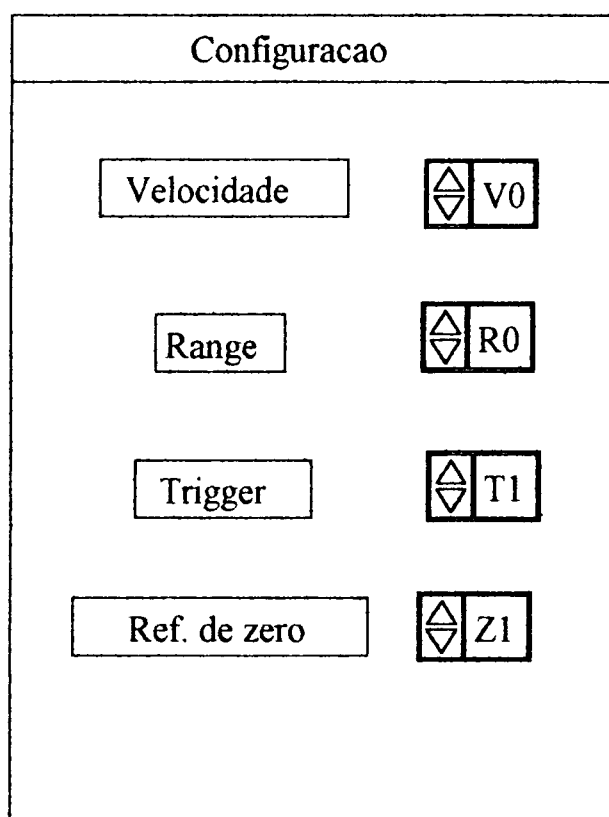
Este sistema permite a aquisição periódica de dados com um pedido pré definido, bem como seleccionar o instante de início o de fim.

Ao accionar o botão de start, o programa começa a adquirir dados com um intervalo constante, o que é garantido através da função "syncwait ()" e "timer ()". A função de "timer ()" devolve o tempo nesse instante e a de "syncwait ()" permite esperar um determinado intervalo de tempo a partir desse instante. A aquisição de dados é feita com o auxílio da função "PM2519_leitura ()", que devolve a leitura do multímetro sob a forma de um número real (double), como um dos seus parâmetros, assim como o tipo de leitura efectuado.

O gráfico utilizado é do tipo "StripChart", permitindo a simulação da aquisição de dados em tempo real, pois permite ir rodando o gráfico à medida que novos dados são acrescentados.

O menu apresentado pelo programa permite a configuração do multímetro de acordo com a escolha do utilizador, recorrendo à função "PM2519_config ()".

A introdução dos parâmetros de configuração, é feita, com o auxílio de um "PopUpPanel" concebido para o efeito.



No anexo I, é apresentada uma listagem do programa, devidamente comentada.
O funcionamento do programa encontra -se descrito no manual respectivo.

4-Programa desenvolvido para um osciloscópio digital DL1200A da Yokogawa

4.1-Descrição do instrumento

Trata-se de um osciloscópio que permite a aquisição de quatro canais, se necessário em simultâneo, com possibilidade de adquirir sinais até 100MHz.

É possível para cada canal, ajustar o ganho, a atenuação de entrada, o tipo de acoplamento, ajustar a sua posição no ecrã, etc...

Possui vários modos de trigger, sendo possível ter como fonte qualquer um dos canais, ou trigger externo. O seu acoplamento pode ser do tipo AC, DC, HF-REJ, ou TV. É possível ajustar o nível de trigger, o delay, e também o Hold-Off.

Dispõe de uma tecla (MEASURE), que permite a visualização de várias medidas, nomeadamente período, tempo de subida, overshoot, valor pico a pico, tensão máxima e mínima etc...

Dispõe de oito memórias para gravar formas de onda, e quatro para gravar painéis completos.

Tem a possibilidade de configurar os canais automaticamente através de um "autosetup".

É permitido iniciar e interromper a aquisição do sinal recorrendo à tecla de START/STOP.

4.2-Descrição do programa

O presente trabalho tem como objectivo a aquisição de sinal e o controlo remoto de um osciloscópio utilizando um computador pessoal equipado de interface GPIB.

O programa construído para o efeito permite alterar a configuração do osciloscópio, assim como adquirir os vários canais do mesmo, e de seguida converter as ondas adquiridas para um formato compatível com programas como o WORD ou o Corel Draw.

O programa "Osci" usa a mesma estrutura básica do programa "Aquadat", conjunto de painéis, módulo do instrumento e programa principal.

Para auxiliar a descrição do desenvolvimento do módulo do instrumento, ilustra-se de seguida a sua árvore de funções.

Inicializar

- Abrir

- Mostra canal

- Inicializar

- Auto setup

Start

Stop

Gotolocal

Gravar em memória

- Save data

- Save panel

- Load data

- Load panel

Configurar

- Trigger
 - Mode
 - Level
 - Source
 - Delay
 - Coupling
 - Hold off
- Time/div
- Acquisition
- Configura canal
 - Input
 - V/Div
 - Position

Aquisição

- Adquire onda
- Measure

Fechar

- Fechar canal
- Fechar osciloscópio

No primeiro bloco, bloco de inicialização, encontram-se as funções de inicialização de comunicação com o dispositivo (Abrir), selecção do canal (Mostra canal), inicialização do osciloscópio num estado conhecido (Inicializar), e configuração automática dos canais (Auto setup). A listagem comentada de todas estas funções, encontra-se no Anexo III.

Para inicializar ou parar a aquisição de uma onda, existem os comandos de Start e Stop, respectivamente, enviados ao osciloscópio através das rotinas com o mesmo nome. Existe também uma função específica que executa o comando GPIB "go to local".

O bloco intitulado "Gravar em memória", tal como o nome indica, contem as funções que permitem gravar para memória, os dados ou o painel na totalidade. Inclui também, as funções que permitem recarregar os dados ou painéis gravados na memória do osciloscópio.

O bloco com maior número de funções, é o de configuração. Inclui um sub-bloco, que reúne as funções de configuração do trigger, quanto ao modo, nível, fonte, tempo de atraso, acoplamento, e hold off. Fazem também parte deste bloco, as funções de selecção do tempo por divisão (Time/div), de selecção do modo de aquisição (Acquisition), e o sub grupo de funções de configuração individual de cada canal (Configura canal). Dentro do sub grupo atrás referido, encontram-se as rotinas de Input (permite escolher o tipo de acoplamento, atenuação da ponta, e se se deseja inverter ou não a forma de onda), V/div (ajuste da escala vertical), Position (permite deslocar a forma de onda ao longo do eixo vertical).

As rotinas constituintes do bloco de aquisição, são Adquire onda, e Measure (em modo auto, permite a visualização do período, tensão máxima e mínima, tensão pico a pico, overshoot, etc.). Quanto à função Adquire onda, é feita a recolha de uma forma de onda previamente gravada numa das memórias internas do osciloscópio, e mais tarde poderá ser

visualizada ou tratada. Como se trata de uma função específica, apresenta-se de seguida o seu código.

```
/*-----*/
/* Esta função permite adquirir uma onda do osciloscópio */
/*-----*/
void DL1200_Adquire_onda(canal,memoria,mem)
char *canal;
char *memoria;
double mem[];

{
    char resp [MAXBUF];
    char *respl;
    int f,pos,pos_nums,pos_mem;
    char *nums;
    double numf;

    pos = 0;
    pos_mem = 0;
    /*
    inicialização de strings e variaveis*/
    nums = "          ";
    respl = "          ";
    DL1200_Save_data(canal,memoria); /* é necessário gravar
    primeiro a onda*/
    DL1200_Stop ();
    /* faz stop ao osciloscópio*/
    Fmt(cmd,"XSND A,%s,0,1000,0",memoria); /* enviar os dados presentes na
    memória */
    DL1200_write_data (cmd,NumFmtdBytes());
    ibrd (dev, respl, 32); /* respl contém mensagem com a forma como vai enviar os
    dados e o seu nº*/
    ibrd (dev, resp, MAXBUF); /* lê os
    dados para resp*/
    for (f = 0; f < MAXBUF; f++)
    {
        if (resp[f] != 69) /* Este ciclo permite retirar
        os caracteres indesejáveis, ou seja*/
        if (resp[f] != 46) /* todos os caracteres
        diferentes dos nº em notação científica*/
        if (resp[f] != 43)
        if (resp[f] != 45)
        if ((resp[f] > 57) || (resp[f] < 48))
            resp[f] = 32;
    }
    while ( pos < MAXBUF )
    {
        /* converte os dados de formato string para*/
        pos_nums = 0;
        /* formato real*/
        while ((resp[pos] == 32) && (pos < (MAXBUF - 1))) /* avança espaços em branco*/
        {
            pos++;
        }
    }
}
```

```
while ((resp[pos] != 32) && (pos < (MAXBUF - 1))) /* passa para nums uma string com  
o nº */  
{  
/* a converter*/  
    nums[pos_nums] = resp[pos];  
    pos++;  
    pos_nums++;  
}  
Scan(nums,"%s>%f", &numf);  
pos++;  
mem[pos_mem] = numf; /* coloca os dados em formato  
real no buffer mem */  
pos_mem++;  
}  
}
```

Tal como o nome indica, no bloco fechar encontram-se as funções que permitem desligar um canal, e fechar o osciloscópio terminando assim a comunicação.

A listagem com todas estas funções encontra-se no Anexo III.

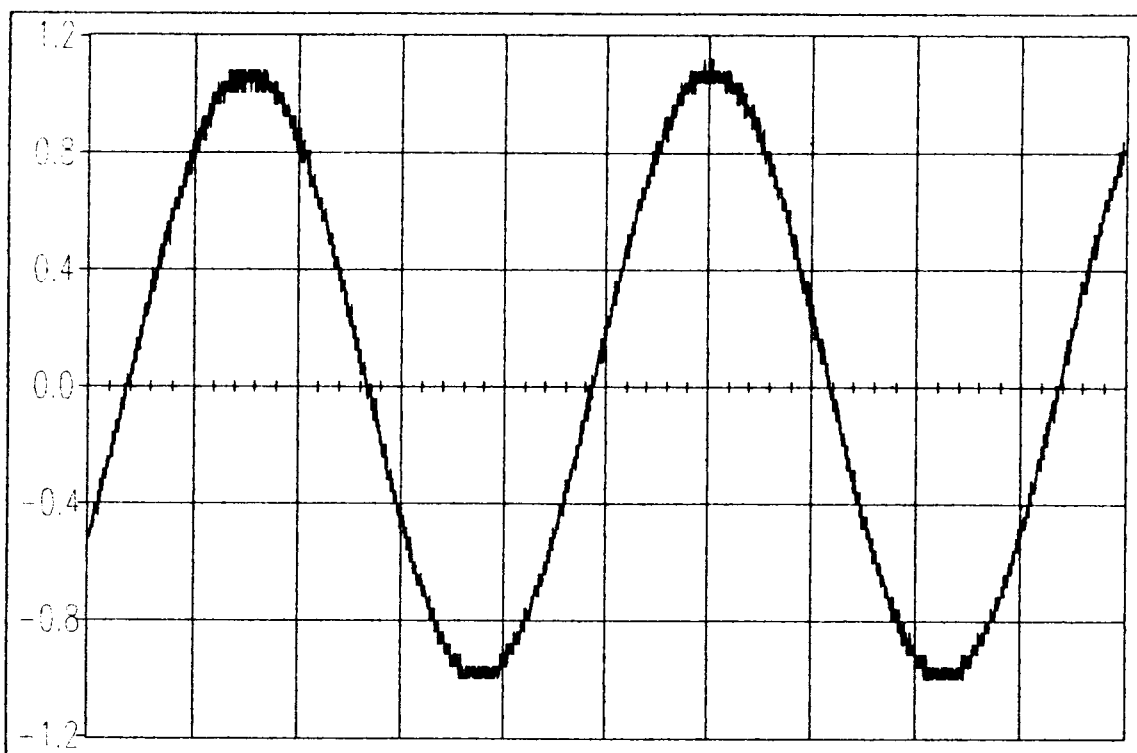
O programa principal permite operar os painéis assim como controlar o osciloscópio.

Os painéis foram escritos em ficheiros diferentes por forma a ser mais fácil o seu desenvolvimento e implementação no programa, (o LabWindows tem também a possibilidade de colocar estes painéis todos no mesmo ficheiro). Foi construído um painel principal, cuja função é permitir aceder às várias opções do programa, abrindo por vezes outros painéis que contêm as funções mais específicas.

O programa está dividido numa função principal, o main e várias funções específicas. O main permite tratar os vários eventos ocorridos no painel principal e chamar as funções necessárias para cada caso. Como por exemplo, carregando na tecla (Mode) do painel irá desencadear um evento, que será tratado chamando a subrotina trata_mode(), esta subrotina afixa um subpainel (no programa LabWindows chamado de PopupPanel) que irá permitir fazer a selecção do modo de trigger pretendido e que em seguida irá actualizar o osciloscópio segundo essa selecção. É deste modo que funcionam a maior parte das funções disponibilizadas por este programa.

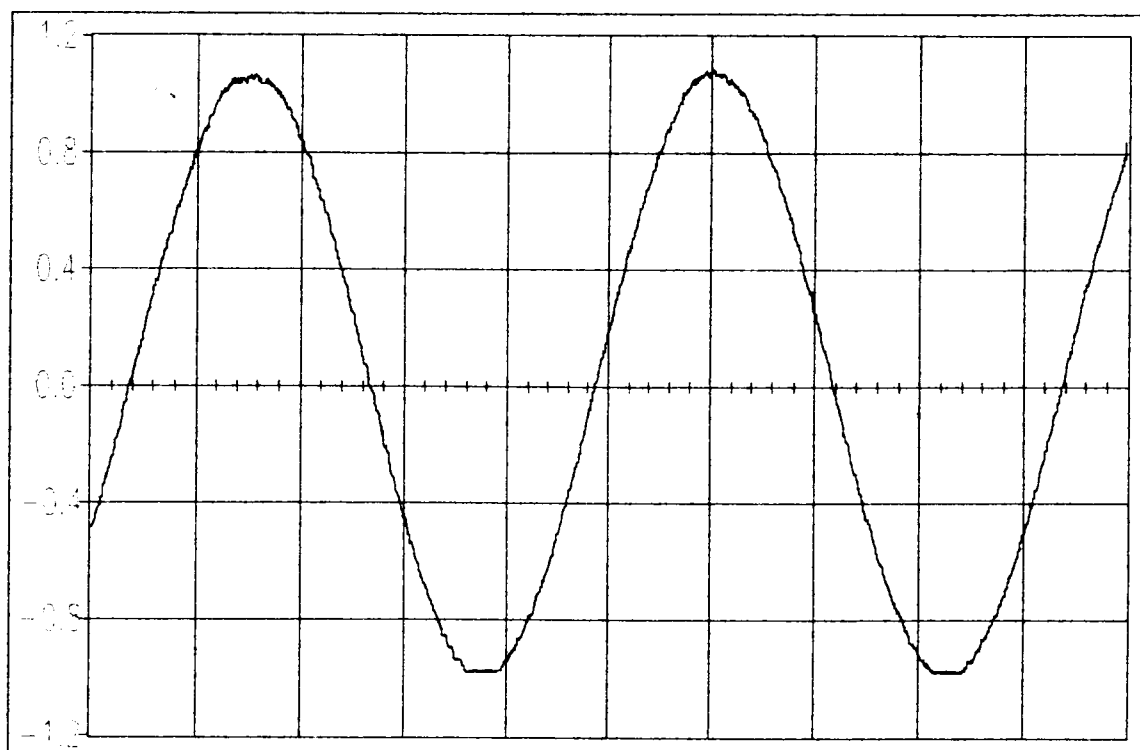
A função trata_converte() faz uma cópia do gráfico recolhido para um ficheiro no formato HP-GL, para posterior operação por outros programas.

Uma função que convém destacar, é a trata_onda(). Foi realizada com vista a eliminar as discontinuidades existentes na onda devido à resolução finita do osciloscópio. De seguida apresenta-se um exemplo que ilustra o aspecto referido.



A função `Trata_onda( )`, permite substituir cada ponto, pela média do próprio ponto com os quatro pontos mais próximos.

Fazendo um tratamento desta onda, através da função referida, o seu aspecto passará a ser o seguinte.



Foi também construído outro programa "ADQONDA", que permite apenas a aquisição de ondas e respectivo tratamento. A sua listagem pode ser consultada no anexo V.

5-Conclusões

Foram realizados alguns programas de controlo de instrumentação recorrendo ao uso da interface GPIB e do ambiente de desenvolvimento do LabWindows. Os instrumentos utilizados foram um multímetro " Philips PM2519 " e um osciloscópio digital " Yokogawa DL1200A ".

Verificou-se que os programas desenvolvidos correm a uma velocidade bastante mais baixa quando executados através do LabWindows, relativamente á sua velocidade quando compilados em Microsoft C6.0. No entanto ao compilar alguns dos programas encontraram-se algumas dificuldades devido a uma selecção errada do modelo de memória e tamanho da pilha necessários, nomeadamente no caso do programa "OSCI". Para resolver este problema mudou-se o modelo de memória para "huge" e aumentou-se o tamanho da pilha.

Bibliografia

Manual de utilização do multímetro Philips PM2519/51
Manual de utilização do osciloscópio Yokogawa DL1200A
LabWindows User Manual
LabWindows Instrument Library Developer's Guide
LabWindows User Interface Library Reference Manual
LabWindows Standard Libraries Reference Manual
LabWindows Data Acquisition Library Reference Manual
Getting Started with LabWindows
Application Guide for PC-Controlled test setups

Anexos

Anexo I

Módulo para o multímetro PM2519

```
#include "\lw\include\gpib.h"
#include "\lw\include\formatio.h"
#include "exp2519.h"

static int bd;
static char cmd[50];

int PM2519_invalid_integer_range();
int PM2519_invalid_real_range();
int PM2519_device_closed();
int PM2519_read_data();
int PM2519_write_data();
int PM2519_read_data_file();
int PM2519_write_data_file();
int PM2519_poll();
int PM2519_set_timeout();
void PM2519_setup_arrays();

void PM2519_init ()
{
    bd = 0;
    bd = ibfind("PM2519");

    /*          testa identificacao          */

    if (PM2519_write_data ("ID?", 3) != 0)
        return;
    if (PM2519_read_data (cmd, 50) != 0)
        return;
    Scan (cmd, "PM2519C:S1");
    if (NumFmtdBytes () != 10)
    {
        PM2519_err = 223;
        return;
    }
}

/*          fim de init          */

/*----- Rotinas dependentes do instrumento -----*/

/* Configura o multímetro */

void PM2519_config(velocidade, range, trig, zref)
char *velocidade;
char *range;
char *trig;
char *zref;
```

```

{
    if (PM2519_device_closed()) return;
    if ( PM2519_write_data(velocidade,2) ) return;
    if ( PM2519_write_data(range,2) ) return;
    if ( PM2519_write_data(trig,2) ) return;
    if ( PM2519_write_data(zref,2) ) return;
    PM2519_err = 0;
}

/* Lê o valor e converte para um formato adequado */

void PM2519_leitura(tipo, valor)
char *tipo;
double *valor;
{
    if (PM2519_device_closed()) return;
    if ( PM2519_read_data(cmd,19) ) return;
    Scan(cmd,"%s>%s%f",tipo,valor);
}

/* Fecha o multímetro */

void PM2519_close ()
{
    /* verifica se o dispositivo já está fechado */

    if (PM2519_device_closed())
        return;

    /* Fecha o dispositivo */

    if ( ibonl (bd,0) < 0)
        PM2519_err = 221;
    else
        PM2519_err = 0;

    bd = 0;
}

/*----- ROTINAS UTILITÁRIAS -----*/

int PM2519_invalid_real_range (val, min, max, err_code)
double val;
double min;
double max;
int err_code;
{
    if ((val < min) || (val > max)) {
        PM2519_err = err_code;
        return -1;
    }
    return 0;
}

```

```
/* Verifica se o multímetro est  fechado */
```

```
int PM2519_device_closed ()
```

```
{
    if (bd <= 0) {
        PM2519_err = 232;
        return -1;
    }
    return 0;
}
```

```
/* Faz a leitura de dados */
```

```
int PM2519_read_data (buf, cnt)
```

```
char *buf;
```

```
int cnt;
```

```
{
    if (ibrd(bd, buf, (long)cnt) <= 0)
        PM2519_err = 231;
    else
        PM2519_err = 0;

    ibloc (bd);
    return PM2519_err;
}
```

```
/* Escreve para o multímetro */
```

```
int PM2519_write_data (buf, cnt)
```

```
char *buf;
```

```
int cnt;
```

```
{
    if (ibwrt(bd, buf, (long)cnt) <= 0)
        PM2519_err = 230;
    else
        PM2519_err = 0;

    return PM2519_err;
}
```

Anexo II

Listagem do programa aquidat.c, (sistema de aquisição de dados)

```
#include "C:\LW\include\formatio.h"
```

```
#include "C:\LW\INSTR\PM2519.h"
```

```
#include "C:\LW\include\lwsystem.h"
```

```
#include "C:\LW\include\userint.h"
```

```
/*
```

```
*/
```

```
/* L^ os valores do multímetro a uma cad^ncia determinada pelo utilizador */
```

```
/*
```

```
*/
```

```
#include "aquidat.h"
#include "aquiconf.h"

void config_pm2519();
void adquire_dados();
void mostra();

int menu1, panel_config, handle, conid, val;
int id_pop, panel;
int val_mem;
double mem1[500], mem2[500];

main()
{
double intervalo, min, max;
char *resp;
long comp;
int desc_fich, sair, aux;

resp = "          ";
panel = LoadPanel ("AQUIDAT.uir", P1);
menu1 = LoadMenuBar ("aquidat.uir", MENU);
val_mem = 1;

DisplayPanel (panel);
while (1)
{
  GetUserEvent (0, &handle, &conid);
  if (handle == menu1)          /* handle={painel, menu} */
  {
    switch (conid)              /* conid={botões premidos} */
    {
      case MENU_FILE_SMEM1:
        resp = "          ";
        PromptPopup ("Nome do ficheiro", resp, 12);
        sair = 1;
        if ((GetFileInfo (resp, &comp) == 1))
        {
          if (!(ConfirmPopup("Esse ficheiro ja existe. Gravar por cima ?")))
            sair=0;
        }
        if (sair)
        {
          desc_fich = OpenFile (resp, 2, 0, 0);
          if (desc_fich == -1)
          {
            MessagePopup ("Erro ao abrir ficheiro");
          }
          aux = FmtFile (desc_fich, "%500f<%500f", mem1);
          aux = CloseFile (desc_fich);
        }
        break;
      case MENU_FILE_SMEM2:
        resp = "          ";
```

```

    PromptPopup ("Nome do ficheiro", resp, 12);
    sair = 1;
    if ((GetFileInfo (resp, &comp) == 1))
    {
        if (!(ConfirmPopup("Esse ficheiro ja existe. Gravar por cima ?")))
            sair=0;
    }
    if (sair)
    {
        desc_fich = OpenFile (resp, 2, 0, 0);
        if (desc_fich == -1)
        {
            MessagePopup ("Erro ao abrir ficheiro");
        }
        aux = FmtFile (desc_fich, "%500f<%500f", mem2);
        aux=CloseFile (desc_fich);
    }

    break;
case MENU_FILE_LMEM1:
    resp =
";
    aux = FileSelectPopup ("", "*.*", "Seleccionar o ficheiro", 0, 0, 1,
resp);
    desc_fich = OpenFile (resp, 1, 2, 0);
    if (desc_fich == -1)
    {
        MessagePopup ("Erro ao abrir ficheiro");
    }
    aux = ScanFile (desc_fich, "%500f>%500f", mem1);
    aux = CloseFile (desc_fich);
    break;
case MENU_FILE_LMEM2:
    resp =
";
    aux = FileSelectPopup ("", "*.*", "Seleccionar o ficheiro", 0, 0, 1,
resp);
    desc_fich = OpenFile (resp, 1, 2, 0);
    if (desc_fich == -1)
    {
        MessagePopup ("Erro ao abrir ficheiro");
    }
    aux = ScanFile (desc_fich, "%500f>%500f", mem2);
    aux = CloseFile (desc_fich);
    break;
case MENU_SAIR:
    UnloadMenuBar ();
    UnloadPanel (panel);
    exit(1);
    break;
case MENU_CONFIG:
    panel_config = LoadPanel ("aquiconf.uir", P_CONFIG);
    if (panel_config < 0 )
    {
        MessagePopup ("Erro ao abrir o painel de configuracao");
    }

```

```

    }
    InstallPopup (panel_config);
    GetPopupEvent (1, &id_pop);
    config_pm2519();
    if ( id_pop == P_CONFIG_OK )
        RemovePopup (1);
    break;
}
} /*if*/
else
{
    switch (conid)
    {
        case P1_INI:
            GetCtrlVal (panel, P1_INI, &val);
            SetCtrlVal (panel, P1_LED, val);
            SetInputMode (panel, P1_MEM1, val);
            SetInputMode (panel, P1_MEM2, val);
            SetInputMode (panel, P1_START, val);
            SetInputMode (panel, P1_STOP, val);
            SetInputMode (panel, P1_MOSTRA, val);
            SetInputMode (panel, P1_MIN, val);
            SetInputMode (panel, P1_MAX, val);
            if (val == 1)
            {
                GetCtrlVal (panel, P1_MAX, &max);
                GetCtrlVal (panel, P1_MIN, &min);
                ConfigureAxes (panel, P1_GRAF, -1, 0.0, 1.0, 0, min, max);
            }
            DeletePlots (panel, P1_GRAF);
            PM2519_init ();
            break;
        case P1_INTER:
            break;
        case P1_MEM1:
            SetCtrlVal (panel, P1_MEM1, 1);
            SetCtrlVal (panel, P1_MEM2, 0);
            val_mem = 1;
            break;
        case P1_MEM2:
            SetCtrlVal (panel, P1_MEM2, 1);
            SetCtrlVal (panel, P1_MEM1, 0);
            val_mem = 0;
            break;
        case P1_START:
            GetCtrlVal (panel, P1_MIN, &min);
            GetCtrlVal (panel, P1_MAX, &max);
            ConfigureAxes (panel, P1_GRAF, -1, 0.0, 1.0, 0, min, max);
            SetInputMode (panel, P1_MEM1, 0);
            SetInputMode (panel, P1_MEM2, 0);
            SetInputMode (panel, P1_MOSTRA, 0);
            GetCtrlVal (panel, P1_INTER, &intervalo);
            adquire_dados(intervalo);
            SetInputMode (panel, P1_MEM1, 1);
            SetInputMode (panel, P1_MOSTRA, 1);
    }
}

```

```

        SetInputMode (panel, P1_MEM2, 1);
        break;
    case P1_STOP:
        break;
    case P1_MOSTRA:
        GetCtrlVal (panel, P1_MIN, &min);
        GetCtrlVal (panel, P1_MAX, &max);
        ConfigureAxes (panel, P1_GRAF, -1, 0.0, 1.0, 0, min, max);
        SetInputMode (panel, P1_FRENTE, 1);
        SetInputMode (panel, P1_START, 0);
        SetInputMode (panel, P1_STOP, 0);
        SetInputMode (panel, P1_MEM1, 0);
        SetInputMode (panel, P1_MEM2, 0);
        SetInputMode (panel, P1_INI, 0);
        if (val_mem) mostra(mem1);
            else mostra(mem2);
        SetInputMode (panel, P1_MEM1, 1);
        SetInputMode (panel, P1_MEM2, 1);
        SetInputMode (panel, P1_INI, 1);
        SetInputMode (panel, P1_START, 1);
        SetInputMode (panel, P1_STOP, 1);
        SetInputMode (panel, P1_FRENTE, 0);
        SetCtrlVal (panel, P1_MOSTRA, 0);
        break;
    }
} /* else */
} /* while */
UnloadMenuBar ();
UnloadPanel (panel);
}

/* Faz a configuração do multímetro */

void config_pm2519()
{
    char *vel,*range,*trig,*zref;

    vel =" ";
    range =" ";    /* É necessário inicializar sempre as strings */
    trig =" ";
    zref =" ";
    GetCtrlVal (panel_config, P_CONFIG_VEL, vel);
    GetCtrlVal (panel_config, P_CONFIG_RANGE, range);
    GetCtrlVal (panel_config, P_CONFIG_TRIG, trig);
    GetCtrlVal (panel_config, P_CONFIG_ZREF, zref);
    PM2519_config (vel, range, trig, zref);
    if ( PM2519_err != 0 ) MessagePopup ("Erro na configuracao do multímetro");
}

/* Adquire dados do multímetro */

void adquire_dados(intervalo)
double intervalo;
{
    int t,con_id,handle;

```

```

double leitura,tempo,aux[1];
char *tipo;

DeletePlots (panel, P1_GRAF);
tipo = " ";
t = 0;
while (t<500)
{
    tempo = timer ();
    GetUserEvent (0, &handle, &con_id);
    if ((handle == P1 )||(con_id == P1_STOP))
        return;

    PM2519_leitura (tipo, &leitura);
    aux[0] = (double) leitura;
    PlotStripChart (panel, P1_GRAF, aux, 1, 0, 0, 4);
    if (val_mem) mem1[t] = leitura;
        else mem2[t] = leitura;
    t++;
    syncwait (tempo, intervalo);
}
}

/* Mostra a aquisição de dados */

void mostra(mem)
double mem[500];
{
    int sair,handle,conid,f,pos,p,i;
    double aux[10];

    DeletePlots (panel, P1_GRAF);
    PlotStripChart (panel, P1_GRAF, mem, 100, 0, 0, 4);
    sair = 0;
    pos = 99;
    while (sair != 1)
    {
        GetUserEvent (0, &handle, &conid);
        if ((conid == P1_FRENTE) || (handle == P1))
        {
            for (i=0; i < 10; i++)
            {
                aux[i] = mem[pos];
                pos++;
                if (pos == 499) sair = 1;
            }
            PlotStripChart(panel, P1_GRAF, aux,10,0,0,4);
            if (pos >= 499) MessagePopup ("Amostragem completa");
        }
        if ((conid == P1_MOSTRA) || (handle == P1)) sair = 1;
    } /*while */
    for (p = f; p < 100; p++) mem [p] = 0;
}

```

Anexo III

Módulo para o Osciloscópio Digital DL1200A

```
#include "c:\\lw\\include\\gpi.h"
#include "c:\\lw\\include\\formatio.h"
#include "DL1200.h"

/*- VARIÁVEIS ESTÁTICAS-----*/
/* dev contém o descritor do dispositivo.          */
/* brd contém o descritor da placa.                */
/* cmd é um buffer para receber informação de comunicação GPIB */
/*-----*/
static int dev,brd;
static char cmd[50];

/*- ROTINAS UTILITÁRIAS-----*/
int DL1200_invalid_integer_range();
int DL1200_invalid_real_range();
int DL1200_device_closed();
int DL1200_read_data();
int DL1200_write_data();
int DL1200_read_data_file();
int DL1200_write_data_file();
void DL1200_init();

.

/*-----*/
/* Esta função abre a placa e o dispositivo, e faz o clear da placa */
/*-----*/
void DL1200_init ()

{
brd = ibfind ("GPIB0");          /* inicializa a placa e retorna o seu descritor */
/*
dev = ibfind ("DL1200A");        /* inicializa o dispositivo e retorna o seu descriptor*/
ibsic(brd);                      /* faz clear da placa */
/*
}

/*-----*/
/* Esta função permite mostrar o canal seleccionado */
/*-----*/
void DL1200_Mostra_canal(canal)
char *canal;
{
Fmt(cmd,"%sON;",canal);          /* formata o buffer com o comando de abertura*/
DL1200_write_data (cmd,NumFmtdBytes()); /* do canal e envia-o ao aparelho*/
return;
}
```

```

/*-----*/
/*          Esta função permite fazer inicializar o osciloscópio          */
/*-----*/
void DL1200_inicialize()

{
    DL1200_write_data("INIT;IEXEC;", 11);    /* envia o comando de inicialize */
}

/*-----*/
/*          Esta função permite fazer auto-setup ao osciloscópio          */
/*-----*/
void DL1200_Auto_setup()

{
    DL1200_write_data("ASET;AEXEC;", 11);    /* envia o comando de autoseup */
}

/*-----*/
/*          Esta função permite fazer start ao osciloscópio          */
/*-----*/
void DL1200_Start()
{
    Fmt(cmd,"STA;");
    DL1200_write_data (cmd,4);                /* envia o comando de start ao dispositivo */
    return;
}

/*-----*/
/*          Esta função permite fazer stop  ao osciloscópio          */
/*-----*/
void DL1200_Stop()
{
    Fmt(cmd,"ST0;");
    DL1200_write_data (cmd,4);                /* envia o comando de stop ao dispositivo */
    return;
}

/*-----*/
/*          Esta função permite executar a instrução gotolocal          */
/*-----*/
void DL1200_gotolocal()

/* Executa a função GPIB "gotolocal" */
{
    ibloc (dev);
}

/*-----*/
/*          Esta função permite gravar dados do osciloscópio          */
/*-----*/
void DL1200_Save_data(canal,memoria)
char *canal;
char *memoria;

/* permite seleccionar o canal que se deseja gravar e qual a memória, enviando em
seguida*/

```

```

/* o comando ao osciloscópio*/

{
    Fmt(cmd,"SAVEL;DATSA %s,%s;DATSE;",canal,memoria);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}
/*-----*/
/*      Esta função permite gravar o painel do osciloscópio      */
/*-----*/
void DL1200_Save_panel(memoria)
char *memoria;

/* permite seleccionar a memória em que se deseja gravar o painel, enviando em seguida*/
/* o comando ao osciloscópio*/

{
    Fmt(cmd,"SAVEL;PNLSA %s;PNLSE;",memoria);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}

/*-----*/
/*      Esta função permite carregar dados provenientes de uma memória      */
/*-----*/
void DL1200_Load_data(canal,memoria,executar)
char *canal;
char *memoria;
char *executar;

/* Esta função permite recarregar os dados provenientes da memória seleccionada */

{
    Fmt(cmd,"SAVEL;DATLO %s,%s;DATL%s;",memoria,canal,executar);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}
/*-----*/
/*      Esta função permite carregar um painel proveniente de uma memória      */
/*-----*/
void DL1200_Load_panel(memoria)
char *memoria;

/* Esta função permite recarregar o painel proveniente da memória seleccionada */

{
    Fmt(cmd,"SAVEL;PNLLO %s;PNLLE",memoria);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}

/*-----*/
/*      Esta função selecciona o modo de trigger      */
/*-----*/

```

```

void DL1200_Mode(modo)
char *modo;

/* Os modos SGL(S) e SGL(L) necessita que seja feito stop ao dispositivo, por isso */
/* a comparação para formatar o buffer de modo diferente */

{
    int result1,result2;
    result1 = CompareStrings(modo,0,"SGL(S)",0,0);
    result2 = CompareStrings(modo,0,"SGL(L)",0,0);
    if ((result1 == 0) || (result2 == 0))
    {
        DL1200_write_data("ST0;", 4);
        Fmt(cmd,"TRGM;MOD %s;",modo);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
    else
    {
        Fmt(cmd,"TRGM;MOD %s;",modo);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
}
/*-----*/
/*          Esta função controla o nível de trigger          */
/*-----*/
void DL1200_Level(canal,nível)
char *canal;
double nível;

/* Envia o nível de trigger desejado para o canal seleccionado */

{
    Fmt(cmd,"TRGL;%s %fV;",canal,nível);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}
/*-----*/
/*          Esta função controla a fonte de trigger          */
/*-----*/
void DL1200_Source(canal,slope)
char *canal;
char *slope;

/* Envia a fonte de trigger desejado para o canal seleccionado */

{
    int result;
    result = CompareStrings(canal,0,"LINE",0,0);          /* Line é o trigger externo, não
necessita de */
    if (result == 0)
/* enviar o slope */
    {

```

```

        Fmt(cmd,"TRGS;SOU %s;",canal);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
    else
    {
        Fmt(cmd,"TRGS;SOU %s;%sSL %s",canal,canal,slope);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
}
/*-----*/
/*          Esta função controla o tempo de atraso do trigger          */
/*-----*/
void DL1200_Delay(atraso)
int atraso;

/* Verifica se o tempo de atraso está entre os limites especificados, se sim, envia o */
/* valor desejado ao osciloscópio*/

{
    if (DL1200_invalid_integer_range (atraso, 0, 500, 300 ) == 0)
    {
        Fmt(cmd,"TRGD;DEL %d;",atraso);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
    else
        return;
}
/*-----*/
/*          Esta função controla o tipo de acoplamento de trigger          */
/*-----*/
void DL1200_Coupling(tipo,tv_f,tv_l)
char *tipo;
int tv_f;
int tv_l;

/* permite seleccionar e enviar o modo de trigger, no caso de ser TV, é necessário enviar */
/* o "field" e a "line" */

{
    int result;
    result = CompareStrings(tipo,0,"TV",0,0);
    if (result == 0)
    {
        if (DL1200_invalid_integer_range (tv_l, 0, 1023, 300 ) == 0)
        {
            Fmt(cmd,"TRGC;COU %s;FI %d;LIN %d;",tipo,tv_f,tv_l);
            DL1200_write_data (cmd,NumFmtdBytes());
            return;
        }
        else return;
    }
}

```

```

    else
    {
        Fmt(cmd,"TRGC;COU %s;".tipo);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
}
/*-----*/
/*      Esta função controla o Hold Off de trigger      */
/*-----*/
void DL1200_Hold_off(select,valor)
char *select;
double valor;

/* Quando o hold off está activo, é necessário enviar o tempo de hold off */

{
    int result;
    result = CompareStrings(select,0,"ON",0,0);
    if (result == 0)
    {
        if (DL1200_invalid_real_range (valor, 160E-9, 80E-3, 300) == 0)
        {
            Fmt(cmd,"TRGH;HOLD0 %s:HOLDT %fs;",select,valor);
            DL1200_write_data (cmd,NumFmtdBytes());
            return;
        }
        else return;
    }
    else
    {
        Fmt(cmd,"TRGH;HOLD0 %s;",select);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
}
/*-----*/
/*      Esta função controla o tempo por divisão do osciloscópio      */
/*-----*/
void DL1200_Time_div(tempo)
char *tempo;

/* Envia o tempo seleccionado, ao osciloscópio */

{
    Fmt(cmd,"TIM;T/DIV %s;",tempo);
    DL1200_write_data (cmd,NumFmtdBytes());
    return;
}

/*-----*/
/*      Esta função controla o modo de aquisição do osciloscópio      */
/*-----*/
void DL1200_Acquisition(modo,count)
char *modo;

```

```

char *count;

/*Ao seleccionar o modo de aquisição AVG, é necessário enviar o valor de " average count
**/

{
    int result;
    result = CompareStrings(modo,0,"AVG",0,0);
    if (result == 0)
    {
        Fmt(cmd,"ACQ ;AVGC %s;",count);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
    else
    {
        Fmt(cmd,"ACQ ;MOD %s;",modo);
        DL1200_write_data (cmd,NumFmtdBytes());
        return;
    }
}

/*-----*/
/*      Esta função permite configurar o canal quanto ao acoplamento,      */
/*      inversão da onda e atenuação                                         */
/*-----*/
void DL1200_Input(acop,inv,ate,canal)
char *acop;
char *inv;
char *canal;
char *ate;

/* formata o buffer de acordo com os valores seleccionados, e envia-o ao aparelho*/

{
    Fmt(cmd,"%s;COU %s;INV %s;PRO %s;", canal, acop, inv, ate);
    DL1200_write_data (cmd,NumFmtdBytes());
}

/*-----*/
/*      Esta função permite configurar a escala vertical.                    */
/*-----*/
void DL1200_V_DIV(esc_ver,canal,esc_pro,half,select)
char *esc_ver;
char *canal;
char *esc_pro;
char *half;
int select;

/* A variavel select faz a distinção entre a selecção de " CAL " ou " HALF " */

{
    if (select)
    {
        Fmt(cmd,"%s;CAL %s;HALF %s;",canal,esc_ver,half);
    }
}

```

```

        DL1200_write_data (cmd, NumFmtBytes());
    }
else
{
    Fmt(cmd, "%s;VAR %s;HALF %s;", canal, esc_pro, half);
    DL1200_write_data (cmd, NumFmtBytes());
}
}

/*-----*/
/*      Esta função permite saber quais os canais ligados      */
/*-----*/
int DL1200_testa_canal(canal)
char *canal;

/* Verifica se o canal escolhido está ligado, se sim retorna 0 */

{
    int result;
    char *resp;

    resp = "      ";
    Fmt(cmd, "%s?;", canal);
    DL1200_write_data (cmd, NumFmtBytes());
    DL1200_read_data (cmd, 5);
    Fmt(resp, "%sON", canal);
    result = CompareStrings (cmd, 0, resp, 0, 0);
    return(result);
}

/*-----*/
/*      Esta função permite ajustar a referência de zero da função      */
/*-----*/
void DL1200_position(canal, pos)
char *canal;
int pos;

/* Verifica se referência de zero está entre os limites especificados, se sim, envia o */
/* valor desejado ao osciloscópio */

{
    if (DL1200_invalid_integer_range (pos, -400, 400, 300) == 0)
    {
        Fmt(cmd, "%s;POS %d;", canal, pos);
        DL1200_write_data (cmd, NumFmtBytes());
        return;
    }
    else
        return;
}

/*-----*/
/*      Esta função permite adquirir uma onda do osciloscópio      */
/*-----*/
void DL1200_Aquire_onda(canal, memoria, mem)

```

```

char *canal;
char *memoria;
double mem[];

{
    char resp [MAXBUF];
    char *respl;
    int f,pos,pos_nums,pos_mem;
    char *nums;
    double numf;

    pos = 0;
    pos_mem = 0;
    inicialização de strings e variáveis*/
    nums = " ";
    respl = " ";
    DL1200_Save_data(canal,memoria); /* é necessário gravar
primeiro a onda*/
    DL1200_Stop ();
    /* faz stop ao osciloscópio*/
    Fmt(cmd,"XSND A,%s,0,1000,0",memoria); /* enviar os dados presentes na
memória */
    DL1200_write_data (cmd,NumFmtdBytes());
    ibrd (dev, respl, 32); /* respl contem mensagem com a forma como vai enviar os
dados e o seu nº*/
    ibrd (dev, resp, MAXBUF); /* lê os
dados para resp*/
    for (f = 0; f < MAXBUF; f++)
    {
        if (resp[f] != 69) /* Este ciclo permite retirar
os caracteres indesejáveis, ou seja*/
        if (resp[f] != 46) /* todos os caracteres
diferentes dos nº em notação científica*/
        if (resp[f] != 43)
        if (resp[f] != 45)
        if ((resp[f] > 57) || (resp[f] < 48))
            resp[f] = 32;
    }
    while ( pos < MAXBUF )
    {
        /* converte os dados de formato string para*/
        pos_nums = 0;
        /* formato real*/
        while ((resp[pos] == 32) && (pos < (MAXBUF - 1))) /* avança espaços em branco*/
        {
            pos++;
        }
        while ((resp[pos] != 32) && (pos < (MAXBUF - 1))) /* passa para nums uma string com
o nº */
        {
            /* a converter*/
            nums[pos_nums] = resp[pos];
            pos++;
            pos_nums++;
        }
    }
}

```

```

        Scan(nums,"%s>%f", &numf);
        pos++;
        mem[pos_mem] = numf;
real no buffer mem */
        pos_mem++;
    }
}
/*-----*/
/*          Esta função permite fazer v rios tipos de medidas          */
/*-----*/
void DL1200_Measure(modo)
char *modo;

/* coloca o osciloscópio no modo de medida seleccionado */

{
    Fmt(cmd,"MEAS;MOD %s;",modo);
    DL1200_write_data (cmd,NumFmtBytes());
    return;
}

/*-----*/
/*          Esta função fecha o canal seleccionado          */
/*-----*/
void DL1200_Fechar_canal(canal)
char *canal;

/* fecha o canal seleccionado */

{
    Fmt(cmd,"%sOFF;",canal);
    DL1200_write_data (cmd,NumFmtBytes());
    return;
}

/*-----*/
/* Esta função fecha o dispositivo e a placa, faz go to local, e dev = 0 */
/*-----*/
void DL1200_close ()

/* Verifica se o dispositivo já foi fechado, se não, fecha-o, terminando a comunicação */

{
    if (DL1200_device_closed())
        return;
    ibloc (dev);
    ibonl (dev, 0);
    ibonl (brd, 0);
    dev = 0;
    brd = 0;
}

/*- ROTINAS UTILITARIAS -----*/
/*-----*/

```

```

/* Esta função verifica se um numero inteiro se encontra nos limites */
/*                               seleccionados                          */
/*-----*/
int DL1200_invalid_integer_range (val, min, max, err_code)
int val;
int min;
int max;
int err_code;
{
    if ((val < min) || (val > max)) {
        DL1200_err = err_code;
        return -1;
    }
    return 0;
}

/*-----*/
/* Esta função permite verificar se um numero real se encontra entre os */
/*                               limites seleccionados                      */
/*-----*/
int DL1200_invalid_real_range (val, min, max, err_code)
double val;
double min;
double max;
int err_code;
{
    if ((val < min) || (val > max)) {
        DL1200_err = err_code;
        return -1;
    }
    return 0;
}

/*-----*/
/* Esta função permite verificar se um dispositivo já foi inicializado */
/*-----*/
int DL1200_device_closed ()
{
    if (dev <= 0) {
        DL1200_err = 232;
        return -1;
    }
    return 0;
}

/*-----*/
/* Esta função permite ler um buffer de dados da interface GPIB */
/*-----*/
int DL1200_read_data (buf, cnt)
char *buf;
int cnt;
{
    if (fbrd(dev, buf, (long)cnt) <= 0)
        DL1200_err = 231;
    else

```

```

        DL1200_err = 0;

    return DL1200_err;
}

/*-----*/
/*      Esta função escreve um buffer de dados para a interface GPIB      */
/*-----*/
int DL1200_write_data (buf, cnt)
char *buf;
int cnt;
{
    if (ibwrt(dev, buf, (long)cnt) <= 0)
        DL1200_err = 230;
    else
        DL1200_err = 0;

    return DL1200_err;
}

/*-----*/
/* Esta função permite ler um buffer de dados do instrumento e colocá-los */
/* num ficheiro designado por "filename" */
/*-----*/
int DL1200_read_data_file (filename)
char *filename;
{
    if (ibrdf (dev, filename) <= 0)
        DL1200_err = 229;
    else
        DL1200_err = 0;

    return DL1200_err;
}

/*-----*/
/*      Esta função escreve um buffer dados do ficheiro especificado por   */
/*      "filename" para o instrumento */
/*-----*/
int DL1200_write_data_file (filename)
char *filename;
{
    if (ibwrtf (dev, filename) <= 0)
        DL1200_err = 228;
    else
        DL1200_err = 0;

    return DL1200_err;
}

```

Anexo IV

Listagem do programa Osci.c

```

#include "p_input.h"
#include "p_vdiv.h"

```

```
#include "p_positi.h"
#include "p_mode.h"
#include "p_source.h"
#include "p_delay.h"
#include "p_hold.o.h"
#include "p_acqui.h"
#include "pnosc.h"          /* includes das constantes dos v rio paineis construidos */
#include "p_level.h"
#include "p_coupli.h"
#include "p_save.h"
#include "p_load.h"
#include "p_measur.h"
#include "p_conver.h"
#include "c:\lw\instr\DL1200.h"
#include "p_expand.h"
#include "p_sel.h"
#include "p_espera.h"

void trata_mode();
void trata_level();
void trata_source();
void trata_delay();
void trata_coupl();
void trata_aumentar();
void trata_hoff();
void trata_starstop();
void trata_input();
void trata_v_div();
void trata_save();
void trata_load();
void trata_acqui();
void trata_measure();
void trata_converte();
void trata_posit();
void expand();
void adquire_ondas ();
void trata_tempo();
void trata_adq_dados();
void trata_limpa();
void trata_canaais();
void trata_onda();

int p_expand;      /* descritor de painel */
int panell,menu1;  /* descritor de painel */
double mem1[1002],mem2[1002],mem3[1002],mem4[1002]; /* arrays onde iram ser colcadas as ondas */

main()
{
    int id_p,id_c,resp,event,err;

    panell = LoadPanel ("pnosc.uir", PNOSC);    /* carrega painel principal */
    menu1 = LoadMenuBar ("pnosc.uir", BAR);     /* carrega menu */
    p_expand = LoadPanel ("p_expand.uir", GRAF); /* carrega painel p_expand */
}
```

```

DL1200.init ();                                /* inicializa o osciloscópio */
DisplayPanel (panel1);                         /* mostra o painel */
trata_canais();                               /* deteta quais os canais que se encontram
ligados */

while (1)
{
    GetUserEvent (1, &id_p, &id_c); /* obtém o botão ou a opção do menu seleccionada */

    if (id_p == 1)                          /* se for primido um botão no painel */
    {
        switch(id_c)                        /* trata as várias opções seleccionadas */
        {
            case PNOSC_MODE:
                trata_mode();
                break;
            case PNOSC_LEVEL:
                trata_level();
                break;
            case PNOSC_SOURCE:
                trata_source();
                break;
            case PNOSC_DELAY:
                trata_delay();
                break;
            case PNOSC_COUPL:
                trata_coupl();
                break;
            case PNOSC_HOFF:
                trata_hoff();
                break;
            case PNOSC_STARSTOP:
                trata_starstop();
                break;
            case PNOSC_CH1: /* se for primido este botão liga ou desliga o canal conforme o
caso */
                GetCtrlVal (panel1, PNOSC_CH1, &event);
                if (event == 1)
                    DL1200_Mostra_canal ("CH1");
                else
                    DL1200_Fechar_canal ("CH1");
                break;
            case PNOSC_CH2: /* se for primido este botão liga ou desliga o canal conforme o
caso */
                GetCtrlVal (panel1, PNOSC_CH2, &event);
                if (event == 1)
                    DL1200_Mostra_canal ("CH2");
                else
                    DL1200_Fechar_canal ("CH2");
                break;
            case PNOSC_CH3: /* se for primido este botão liga ou desliga o canal conforme o
caso */
                GetCtrlVal (panel1, PNOSC_CH3, &event);
                if (event == 1)
                    DL1200_Mostra_canal ("CH3");

```

```
        else
            DL1200_Fechar_canal ("CH3");
        break;
    case PNOSC_CH4: /* se for primido este but_o liga ou desliga o canal conforme o
caso */
        GetCtrlVal (panel1, PNOSC_CH4, &event);
        if (event == 1)
            DL1200_Mostra_canal ("CH4");
        else
            DL1200_Fechar_canal ("CH4");
        break;
    case PNOSC_INPUT:
        trata_input();
        break;
    case PNOSC_V_DIV:
        trata_v_div();
        break;
    case PNOSC_POSIT:
        trata_posit();
        break;
    case PNOSC_EXPC:
        trata_aumentar();
        break;
    case PNOSC_AUTOSETUP:
        DL1200_Auto_setup (); /* faz a configurat_ao autom tica do osciloscopio */
        break;
    case PNOSC_SAVE:
        trata_save();
        break;
    case PNOSC_LOAD:
        trata_load();
        break;
    case PNOSC_ACQUI:
        trata_acqui();
        break;
    case PNOSC_MEASURE:
        trata_measure();
        break;
    case PNOSC_ADQ_DADOS:
        trata_adq_dados();
        break;
    case PNOSC_TEMPO:
        trata_tempo();
        break;
    case PNOSC_LIMPAR:
        trata_limpa();
        break;
} /*switch*/
} /* if */
else /* se for no menu */
{
    switch (id_c)
    {
        case BAR_FILE_SAVE:
            trata_converte();
```

```

        break;
    case BAR_QUIT:
        resp = ConfirmPopup ("Quer sair ?"); /* pede confirmação */
        if (resp == 1)
        {
            UnloadPanel (panel1); /* descarrega o painel */
            UnloadMenuBar (); /* descarrega o menu */
            DL1200_close (); /* fecha a comunicação com o osciloscópio */
            exit(1);
        }
        break;
    case BAR_TRATA:
        trata_onda();
        break;
    } /* switch */
} /* else */
} /* while */
} /* main */

void trata_canais()
{
    int result1, result2, result3, result4;

    result1 = DL1200_testa_canal ("CH1");
    result2 = DL1200_testa_canal ("CH2"); /* vê o estado dos canais */
    result3 = DL1200_testa_canal ("CH3");
    result4 = DL1200_testa_canal ("CH4");
    if ( result1 == 0 ) SetCtrlVal (panel1, PNOSC_CH1, 1);
    if ( result2 == 0 ) SetCtrlVal (panel1, PNOSC_CH2, 1); /* conforme o resultado liga ou
desliga os botões respectivos */
    if ( result3 == 0 ) SetCtrlVal (panel1, PNOSC_CH3, 1);
    if ( result4 == 0 ) SetCtrlVal (panel1, PNOSC_CH4, 1);
}

void trata_mode() /* pede ao utilizador o modo de trigger */
{
    int p_mode, err, id;
    char *event;

    event = " "; /* sempre necessário inicializar este tipo de arrays */
    p_mode = LoadPanel ("p_mode.uir", MODE); /* carrega o respectivo painel */
    if (p_mode < 0 ) /* testa se existiu erro ao carregar */
    {
        MessagePopup ("Erro ao carregar 'p_mode.uir'");
    }
    err = InstallPopup (p_mode); /* instala o painel */
    GetPopupEvent (1, &id); /* espera até se carregar no botão de OK */
    GetCtrlVal (p_mode, MODE_MODE, event); /* passa para event o valor seleccionado de
utilizador */
    RemovePopup (1);
    DL1200_Mode(event); /* envia o comando */
}

void trata_level() /* pede nível de trigger */

```

```

{
    int p_level,id;
    char *event1;
    double event2;
    event1 = "          ";
    p_level = LoadPanel ("p_level.uir", LEVEL);
    if (p_level < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_level.uir'");
    }
    InstallPopup (p_level);
    GetPopupEvent (1, &id);
    GetCtrlVal (p_level,LEVEL_CH, event1);
    GetCtrlVal (p_level,LEVEL_LEV,&event2);
    if (id == LEVEL_OK )
        RemovePopup (1);
    DL1200_Level(event1,event2);
}

void trata_source()    /* pede fonte de trigger */
{
    int p_source,id;
    char *event1,*event2;

    event1 = "          ";
    event2 = "          ";
    p_source = LoadPanel ("p_source.uir", SOURCE);
    if (p_source < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_source.uir'");
    }
    InstallPopup (p_source);
    GetPopupEvent (1, &id);
    GetCtrlVal (p_source,SOURCE_SOUR, event1);
    GetCtrlVal (p_source,SOURCE_SLOPE,event2);
    if (id == SOURCE_OK )
        RemovePopup (1);
    DL1200_Source(event1,event2);
}

void trata_delay()    /* pede atraso do trigger */
{
    int p_delay,id,event;

    p_delay = LoadPanel ("p_delay.uir", DEL);
    if (p_delay < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_delay.uir'");
    }
    InstallPopup (p_delay);
    GetPopupEvent (1, &id);
    GetCtrlVal (p_delay, DEL_DEL,&event);
    if (id == DEL_OK )

```

```
        RemovePopup (1);
    DL1200_Delay(event);
    if (DL1200_err == EXC_LIMIT) MessagePopup ("Valor excedeu os limites,0 a 500");
    /* a variavel DL1200_err retorna o valor EXC_LIMIT se o valor enviado exceder os
    limites */
}

void trata_coupl() /* Acoplamento de trigger */
{
    int p_coupl,id,event2,event3,sair,result,err,err2;
    char *event1;

    event1 = "          ";
    p_coupl = LoadPanel ("p_coupli.uir", COUPLING);
    if (p_coupl < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_coupli.uir'");
    }
    InstallPopup (p_coupl);
    GetPopupEvent (1, &id);
    GetCtrlVal (p_coupl, COUPLING_TCOUPL,event1);
    GetCtrlVal (p_coupl, COUPLING_TVFI, &event3);
    GetCtrlVal (p_coupl, COUPLING_LINE, &event2);
    DL1200_Coupling(event1, event2, event3);
    if (DL1200_err == EXC_LIMIT) MessagePopup ("Valor excedeu os limites,0 a 1023");
    RemovePopup(1);
}

void trata_hoff() /* pede se quer hold off e qual o seu valor */
{
    int p_hoff,id;
    char *event1;
    double event2;

    event1 = "          ";
    p_hoff = LoadPanel ("p_hold_o.uir", HOLD_OFF);
    if (p_hoff < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_hold_o.uir'");
    }
    InstallPopup (p_hoff);
    GetPopupEvent (1, &id);
    GetCtrlVal (p_hoff, HOLD_OFF_HOLD,event1);
    GetCtrlVal (p_hoff,HOLD_OFF_TIME , &event2);
    if (DL1200_err == EXC_LIMIT) MessagePopup ("Valor excedeu os limites,160E-9 a 80E-3");
    DL1200_Hold_off(event1, event2);
    RemovePopup(1);
}

void trata_starstop() /* faz o start e o stop */
{
    int event,err;
    GetCtrlVal (panel1, PNOSC_STARSTOP, &event);
    if ( event == 1 ) DL1200_Start ();
    else DL1200_Stop();
}
```

```

void trata_input() /* pede o modo de input */
{
    int id,p_input;
    char *event1,*event2,*event3,*event4;

    p_input = LoadPanel ("p_input.uir", HOLD_OFF);
    if (p_input < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_input.uir'");
    }
    InstallPopup (p_input);
    GetPopupEvent (1, &id);
    event1 = " ";
    event2 = " ";
    event3 = " ";
    event4 = " ";
    GetCtrlVal (p_input, P1_COU, event1);
    GetCtrlVal (p_input, P1_CH, event2);
    GetCtrlVal (p_input, P1_INV, event3);
    GetCtrlVal (p_input, P1_PROBE, event4);
    DL1200_Input (event1, event3, event4, event2);
    RemovePopup(1);

}

void trata_v_div() /* escala vertical de nada canal */
{
    int id,p_vdiv,event2;
    char *event1,*event3,*event4,*event5;

    p_vdiv = LoadPanel ("p_vdiv.uir", HOLD_OFF);
    if (p_vdiv < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_vdiv.uir'");
    }
    InstallPopup (p_vdiv);
    GetPopupEvent (1, &id);
    event1 = " ";
    event3 = " ";
    event4 = " ";
    event5 = " ";
    GetCtrlVal (p_vdiv, V_DIV_CH, event1);
    GetCtrlVal (p_vdiv, V_DIV_SCALE, &event2);
    GetCtrlVal (p_vdiv, V_DIV_SIZE, event3);
    GetCtrlVal (p_vdiv, V_DIV_CAL, event4);
    GetCtrlVal (p_vdiv, V_DIV_VAR, event5);
    DL1200_V_DIV (event4, event1, event5, event3, event2);
    RemovePopup(1);

}

void trata_posit() /* pede ao utilizador qual a posico da onda */
{

```

```

int id,p_posit,event2;
char *event1;

p_posit = LoadPanel ("p_positi.uir", POSI);
if (p_posit < 0 )
{
    MessagePopup ("Erro ao carregar 'p_positi.uir'");
}
InstallPopup (p_posit);
GetPopupEvent (1, &id);
event1 = " ";
GetCtrlVal (p_posit, POSI_CH, event1);
GetCtrlVal (p_posit, POSI_POSI, &event2);
if (DL1200_err == EXC_LIMIT) MessagePopup ("Valor excedeu os limites,-400 a 400");
DL1200_Position (event1, event2);/* determina se o valor de event1 est  fora dos limites */
RemovePopup(1);
}

void trata_save()      /* grava uma onda ou um painel para um das memorias do osciloscopio */
/*
{
    int id,p_save;
    char *event1,*event2;

    p_save = LoadPanel ("p_save.uir", SAVE);
    if (p_save < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_save.uir'");
    }
    InstallPopup (p_save);
    GetPopupEvent (1, &id);
    event1 = " ";
    event2 = " ";
    GetCtrlVal (p_save,SAVE_MEM, event1);
    GetCtrlVal (p_save, SAVE_CANAL, event2);
    DL1200_Save_data(event2,event1);
    RemovePopup(1);
}

void trata_load()      /* carrega uma onda ou um painel das memorias do osciloscópio */
{
    int id,p_load;
    char *event1,*event2,*event3;

    p_load = LoadPanel ("p_load.uir", LOAD);
    if (p_load < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_load.uir'");
    }
    InstallPopup (p_load);
    GetPopupEvent (1, &id);
    event1 = " ";
    event2 = " ";
    event3 = " ";

```

```

    GetCtrlVal (p_load,LOAD_MEM, event1);
    GetCtrlVal (p_load, LOAD_CANAL, event2);
    GetCtrlVal (p_load, LOAD_EXE, event3);
    DL1200_Load_data(event2,event1,event3);
    RemovePopup(1);
}

void trata_acqui()      /* pede ao utilizador qual o modo de aquisiç_ão */
{
    int id,p_acqui;
    char *event1,*event2;

    p_acqui = LoadPanel ("p_acqui.uir", LOAD);
    if (p_acqui < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_acqui.uir'");
    }
    InstallPopup (p_acqui);
    GetPopupEvent (1, &id);
    event1 = "          ";
    event2 = "          ";
    GetCtrlVal (p_acqui,ACQUI_ACQUI, event1);
    GetCtrlVal (p_acqui, ACQUI_AVERG, event2);
    DL1200_Acquisition(event1,event2);
    RemovePopup(1);
}

void trata_measure()      /* pede ao utilizador qual o modo de measure */
{
    int id,p_measure;
    char *event1;

    p_measure = LoadPanel ("p_measur.uir", MEASURE);
    if (p_measure < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_measur.uir'");
    }

    InstallPopup (p_measure);
    GetPopupEvent (1, &id);
    event1 = "          ";
    GetCtrlVal (p_measure,MEASURE_MODAL, event1);    /* passa para event1 o modo seleccionado
*/
    DL1200_Measure(event1);
    RemovePopup(1);
}

void trata_converte() /* converte para HP-GL */
{
    int p_converte,id,p_sel,ch1,ch2,ch3,ch4,err,p_menu,handle;
    char *nome_graf;

    nome_graf = "          ";
    p_converte = LoadPanel ("p_conver.uir", CONVERTE);

```

```

    if (p_converte < 0 )
    {
        MessagePopup ("Erro ao carregar 'p_conver.uir'");
    }
    InstallPopup (p_converte); /* afixa mensagem */
    GetPopupEvent (1, &id);
    RemovePopup(1);
    PromptPopup ("Nome do ficheiro ?", nome_graf, 50); /* o nome do ficheiro passa para
nome graf */
    p_sel = LoadPanel ("p_sel.uir", SEL);
    InstallPopup (p_sel); /* permite seleccionar os canais a passar para HP-GL */
    GetPopupEvent (1, &id);
    GetCtrlVal (p_sel, SEL_C1, &ch1);
    GetCtrlVal (p_sel, SEL_C2, &ch2); /* recolhe as opções */
    GetCtrlVal (p_sel, SEL_C3, &ch3);
    GetCtrlVal (p_sel, SEL_C4, &ch4);
    RemovePopup(1);

    HidePanel (panel1);
    DisplayPanel (p_expand);
    UnloadMenuBar ();
    if (ch1 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem1, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 0);
    if (ch2 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem2, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 1);
    if (ch3 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem3, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 2);
    if (ch4 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem4, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 4);
    p_menu = LoadMenuBar ("p_expand.uir", M2);
    OutputGraph (3, nome_graf, 0, p_expand, GRAF_GRAF); /* passa apenas o grafico para
formato HP-GL */
    GetUserEvent (1, &handle, &id);
    UnloadMenuBar ();
    HidePanel (p_expand);
    menu1 = LoadMenuBar ("pnosc.uir", BAR);
    DisplayPanel (panel1);
}

void adquiere_ondas (ch1,ch2,ch3,ch4) /* adquiere as ondas para memi */
int ch1;
int ch2; /* ondas a adquirir */
int ch3;
int ch4;
{
    int p_espera;

    p_espera = LoadPanel ("p_espera.uir", ESPERA2);
    InstallPopup (p_espera);
    if (ch1 == 1) DL1200_Acquire_onda ("C1", "#1", mem1);
    if (ch2 == 1) DL1200_Acquire_onda ("C2", "#2", mem2);
    if (ch3 == 1) DL1200_Acquire_onda ("C3", "#3", mem3);
    if (ch4 == 1) DL1200_Acquire_onda ("C4", "#4", mem4);
    RemovePopup (1);
}

```

```

void expand(ch1,ch2,ch3,ch4)    /* mostra as ondas em todo o ecran */
int ch1;
int ch2;
int ch3;
int ch4;

{
    int handle,id,p_menu;

    HidePanel (panel1);
    DisplayPanel (p_expand);
    UnloadMenuBar ();
    if (ch1 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem1, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 0);
    if (ch2 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem2, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 1);
    if (ch3 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem3, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 2);
    if (ch4 == 1) PlotWaveform (p_expand, GRAF_GRAF, mem4, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0,
0, 1, 4);
    p_menu = LoadMenuBar ("p_expand.uir", M2);
    GetUserEvent (1, &handle, &id);
    UnloadMenuBar ();
    HidePanel (p_expand);
    menu1 = LoadMenuBar ("pnosc.uir", BAR);
    DisplayPanel (panel1);
}

void trata_adq_dados()        /* aquire as ondas e mostra os dados */
{
    int c1,c2,c3,c4;

    GetCtrlVal (panel1, PNOSE_CH1, &c1);
    GetCtrlVal (panel1, PNOSE_CH2, &c2); /* detecta quais os canais seleccionados */
    GetCtrlVal (panel1, PNOSE_CH3, &c3);
    GetCtrlVal (panel1, PNOSE_CH4, &c4);
    adquiere_ondas(c1,c2,c3,c4);
    if (c1 == 1) PlotWaveform (panel1, PNOSE_GRAF, mem1, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0, 0,
1, 0);
    if (c2 == 1) PlotWaveform (panel1, PNOSE_GRAF, mem2, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0, 0,
1, 1);
    if (c3 == 1) PlotWaveform (panel1, PNOSE_GRAF, mem3, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0, 0,
1, 2);
    if (c4 == 1) PlotWaveform (panel1, PNOSE_GRAF, mem4, 1000, 4, 1.0, 0.0, 0.0, 1.0, 0, 0,
1, 4);
}

void trata_aumentar()        /* aumenta a onda aquirida para todo o ecran */
{
    int c1,c2,c3,c4;

    GetCtrlVal (panel1, PNOSE_CH1, &c1);
    GetCtrlVal (panel1, PNOSE_CH2, &c2);

```

```
GetCtrlVal (panel1, PNOSC_CH3, &c3);
GetCtrlVal (panel1, PNOSC_CH4, &c4);
expand(c1,c2,c3,c4);
}

void trata_tempo()      /* modifica a escala de tempo */
{
    char *resp;

    resp = "          ";
    GetCtrlVal (panel1, PNOSC_TEMPO, resp);
    DL1200_Time_div (resp);
}

void trata_limpa()      /* limpa o ecran e as memorias */
{
    DeletePlots (panel1, PNOSC_GRAF);
    DeletePlots (p_expand, GRAF_GRAF);
    ClearID (mem1, 1002);
    ClearID (mem2, 1002);
    ClearID (mem3, 1002);
    ClearID (mem4, 1002);
}

void trata_onda() /* trata as ondas adquiridas */
{
    int f,c1,c2,c3,c4;

    GetCtrlVal (panel1, PNOSC_CH1, &c1);
    GetCtrlVal (panel1, PNOSC_CH2, &c2);
    GetCtrlVal (panel1, PNOSC_CH3, &c3);
    GetCtrlVal (panel1, PNOSC_CH4, &c4);
    if (c1 == 1)
    {
        mem1 [0] = (mem1[0]+mem1[1]+mem1[2]+mem1[3]+mem1[4]) /5;
        mem1 [1] = (mem1[0]+mem1[1]+mem1[2]+mem1[3]+mem1[4]) /5;
        for (f = 2; f < 999; f++)
        {
            mem1[f] = (mem1[f-2]+mem1[f-1]+mem1[f]+mem1[f+1]+mem1[f+2])/5;
        }
    }
    if (c2 == 1)
    {
        mem2 [0] = (mem2[0]+mem2[1]+mem2[2]+mem2[3]+mem2[4]) /5;
        mem2 [1] = (mem2[0]+mem2[1]+mem2[2]+mem2[3]+mem2[4]) /5;
        for (f = 2; f < 1000; f++)
        {
            mem2[f] = (mem2[f-2]+mem2[f-1]+mem2[f]+mem2[f+1]+mem2[f+2])/5;
        }
    }
    if (c3 == 1)
    {
        mem3 [0] = (mem3[0]+mem3[1]+mem3[2]+mem3[3]+mem3[4]) /5;
        mem3 [1] = (mem3[0]+mem3[1]+mem3[2]+mem3[3]+mem3[4]) /5;
        for (f = 2; f < 1000; f++)
```

```

    {
        mem3[f] = (mem3[f-2]+mem3[f-1]+mem3[f]+mem3[f+1]+mem3[f+2])/5;
    }

}

if (c4 == 1)
{
    mem4[0] = (mem4[0]+mem4[1]+mem4[2]+mem4[3]+mem4[4]) /5;
    mem4[1] = (mem4[0]+mem4[1]+mem4[2]+mem4[3]+mem4[4]) /5;
    for (f = 2; f < 1000; f++)
    {
        mem3[f] = (mem3[f-2]+mem3[f-1]+mem3[f]+mem3[f+1]+mem3[f+2])/5;
    }
}

}

```

Anexo V

Listagem do programa Adqonda.c

```

#include "exp_osc.h"

void copia ();
void mostra_dados();

int panel,men,panel_esc;
double mem1[1002],mem2[1002],mem3[1002],mem4[1002];    /* Arrays onde iram ser guadas as
*/

/*ondas    adquiridas */

main()
{
    int handle,id;
    char *nome_fich;

    DL1200_init ();
    nome_fich = "                                ";
    panel = LoadPanel ("exp_osc.uir", P1);
    panel_esc = LoadPanel ("exp_osc.uir", ESCOLHA); /* carrega os paineis e o menu */
    men = LoadMenuBar ("exp_osc.uir", M1);
    DisplayPanel (panel);
    while (1)
    {
        GetUserEvent (0, &handle, &id);
        switch (id)                                /* trata os eventos */
        {
            case M1_SAIR:
                UnloadPanel (panel);
                UnloadMenuBar ();
                DL1200_close ();
                exit(1);
                break;
        }
    }
}

```

```

    case M1_MOSTRA:
        mostra_dados();
        break;
    case M1_COPIAR:
        copia();
        break;
}
}
}

void copia ()
{
    char *nome_fich;

    nome_fich
";
    PromptPopup ("Nome do ficheiro onde quer gravar ?", nome_fich, 50);
    OutputGraph (3, nome_fich, 0, panel, P1_GRAF);          /* passa para
formato HP-GL */
}

void mostra_dados()
{
    int c1,c2,c3,c4,id;
    int result1, result2, result3,result4;
    char *nome_fich;

    nome_fich
";
    InstallPopup (panel_esc);      /* instalo um painel com uma mensagem de espera */
    result1 = DL1200_testa_canal ("CH1");
    result2 = DL1200_testa_canal ("CH2"); /* vê o estado dos canais */
    result3 = DL1200_testa_canal ("CH3");
    result4 = DL1200_testa_canal ("CH4");
    if ( result1 == 0 ) SetCtrlVal (panel_esc, ESCOLHA_C1, 1);
    if ( result2 == 0 ) SetCtrlVal (panel_esc, ESCOLHA_C2, 1); /* conforme o resultado
liga          ou          desliga          os          */
if ( result3 == 0 ) SetCtrlVal (panel_esc, ESCOLHA_C3, 1);          /* botões
respectivos */
    if ( result4 == 0 ) SetCtrlVal (panel_esc, ESCOLHA_C4, 1);
    GetPopupEvent (1, &id);
    GetCtrlVal (panel_esc, ESCOLHA_C1, &c1);
    GetCtrlVal (panel_esc, ESCOLHA_C2, &c2);          /* adquire o valor da selecção */
    GetCtrlVal (panel_esc, ESCOLHA_C3, &c3);
    GetCtrlVal (panel_esc, ESCOLHA_C4, &c4);
    RemovePopup (1);
    DL1200_Stop ();
    if (c1 == 1) DL1200_Acquire_onda ("C1", "#1", mem1);
    if (c2 == 1) DL1200_Acquire_onda ("C2", "#2", mem2);          /* adquire as ondas */
    if (c3 == 1) DL1200_Acquire_onda ("C3", "#3", mem3);
    if (c4 == 1) DL1200_Acquire_onda ("C4", "#4", mem4);
    DL1200_Start ();
}

```



FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

BIBLIOTECA



0000101497