Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

DESENVOLVIMENTO DE *FIRMWARE* E *SOFTWARE* APLICATIVO DE CONTROLE PARA UMA MÁQUINA DE ENSAIOS GEOLÓGICOS

Aluno: Fernando dos Santos Alves Fernandes
Matricula: 08.1.4027

Orientador: Ricardo Augusto de Oliveira Rabelo
Co-orientador: Robson Nunes Dal Col

Ouro Preto
24 de junho de 2011

Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

DESENVOLVIMENTO DE *FIRMWARE* E *SOFTWARE* APLICATIVO DE CONTROLE PARA MÁQUINA DE ENSAIOS GEOLÓGICOS

Relatório de atividades desenvolvidas apresentado ao curso de Bacharelado em Ciência da Computação, Universidade Federal de Ouro Preto, como requisito parcial para a conclusão da disciplina Monografia I (BCC390).

Aluno: Fernando dos Santos Alves Fernandes
Matricula: 08.1.4027

Orientador: Ricardo Augusto de Oliveira
Co-orientador: Robson Nunes Dal Col

Ouro Preto
24 de junho de 2011

Resumo

O presente relatório descreve as atividades já realizadas para o desenvolvimento de *firmware* e *software* aplicativo de interface com o usuário, para controle de uma máquina de ensaios geológicos. O microcontrolador escolhido para este projeto foi o *PIC18F4550* da *Microchip* e a *IDE* para desenvolvimento do *software* aplicativo o *Borland C++ Builder 6.0*, para programação em linguagem*C/C++*. A interface de comunicação entre *software* aplicativo e sistema embutido deverá utilizar o padrão *USB (Universal Serial Bus)*.

Palavras-chave: Firmware. Microcontroladores. Sistemas Embutidos. USB em Modo CDC.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Os métodos científicos são fundamentados em teorias, definições, conjecturas, mas podem também ser fundamentados em análises de procedimentos empíricos, cujos resultados podem consolidar ou fortalecer uma hipótese ou conjectura.

Em diversas áreas, esses procedimentos empíricos são denominados *Ensaios*, nos quais o material/substância em estudo pode ser submetido a ações externas e/ou condições extremas, segundo o que se pretende observar do experimento. Os objetivos são diversos, conforme a área de estudo, podendo ser realizados para análise de resistência de um material, observação de uma reação química e/ou física, verificação da presença de alterações na dinâmica molecular de determinado material ou substância, submetidos a esforços, condições extremas ou a um ambiente específico.

No caso da Geologia, estudos relacionados à movimentação de placas tectônicas ou formações rochosas se baseiam em diferentes tipos de ensaios, em que se procura simular a dinâmica de material geológico, submetido à pressão ou demais ações do tempo ou da natureza. O estudo dos resultados desses experimentos permite entender parte da dinâmica desses eventos na natureza e estimar o que ocorre num ambiente natural sob condições similares, ou seja, que elementos naturais (compostos minerais) podem ser originados num solo sob determinadas condições.

Considerando que esses processos ocorrem na natureza de forma lenta, em período de centenas a milhares de anos, normalmente os ensaios são realizados durante horas ininterruptas, pois os movimentos de compressão e extensão do material precisam ser realizados lentamente, de maneira a simular, o mais fielmente possível a dinâmica da natureza.

Em geral, os experimentos são realizados por máquinas, projetadas para este fim e que oferecem uma série de possibilidades de ambientes, condições e observações sobre o material em estudo. Essas máquinas precisam ser projetadas de maneira a terem uma grande resistência mecânica e precisão no controle dos movimentos. Seja qual for o ensaio, se o experimento for realizado por uma máquina automatizada, existe sempre a necessidade de se prever o projeto de um sistema de controle. Como se trata de um sistema dedicado a tarefas específicas, em geral, desenvolve-se um sistema de computação embutido para o controle do equipamento. Para que o operador da máquina possa ter o processo de configuração do equipamento facilitado e acesso às diversas informações coletas durante o processo, também é necessário desenvolver um *software* para interface entre o operador e o equipamento.

2 Justificativa

O projeto de uma máquina de ensaios como a descrita na seção anterior, cujo fim é a realização de tarefas automatizadas, precisa prever como subprojeto o desenvolvimento dos elementos que irão compor os módulos de controle de todo o equipamento.

Esses módulos podem ser definidos como o sistema de computação embutido, contendo um microcontrolador e seu *firmware* associados a outros circuitos eletrônicos, um *software* que permita ao operador do equipamento informar parâmetros de operação do sistema embutido e visualizar informações provenientes deste, obtidas a partir da leitura de sensores ou do resultado de algum processamento. Há que se prever ainda, a interface de comunicação entre *software aplicativo* e *firmware*.

No caso específico deste projeto, esses elementos se constituem de um *firmware* para a definição das tarefas realizadas pelo microcontrolador, ou seja, o *software* que estará "rodando"no sistema embutido, e de um *software* aplicativo, desenvolvido em linguagem de alto nível, capaz de permitir ao operador, visualizar as informações provenientes dos sensores instalados no equipamento, bem como acionar ou enviar ao equipamento, parâmetros que permitam configurar o modo como as tarefas serão realizadas automaticamente.

Como foi dito, o equipamento precisa ser controlado de maneira a oferecer precisão em seus movimentos. Além disso, é importante que o operador dos sistemas tenha facilidade em configurar o equipamento e acessar as mais diversas informações sobre o estado do equipamento durante e depois do experimento.

Este trabalho se justifica, portanto, em razão de seu resultado ser parte integrante de todo o projeto da máquina de ensaios. Toda a estrutura mecânica e os circuitos eletrônicos de controle e de potência se encontram projetados e especificados. Entretanto, sem o projeto e desenvolvimento do *software* de controle do sistema embutido (*firmware*), bem como do *software* de operação e monitoramento, que faz a interface entre operador e equipamento (*software* aplicativo), o próprio projeto da máquina não se justifica.

3 Objetivos

3.1 Objetivo geral

- O objetivo geral deste trabalho é o desenvolvimento de um *firmware* (*software* embutido para microcontroladores) e um *software* aplicativo de parametrização e supervisão de uma máquina para ensaios geológicos.

3.2 Objetivos específicos

- Desenvolver o *firmware* para um microcontrolador *PIC18F4550* da *Microchip*;
- Desenvolver *software* aplicativo para permitir ao operador a definição dos parâmetros de cada tipo de experimento que o equipamento oferece, bem como o monitoramento e supervisão de todo o experimento;
- Implementar a comunicação entre *software* aplicativo e *firmware* via interface *USB (Universal Serial Bus)*;
- Verificar a viabilidade de uso de paradigmas de *Programação Orientada a Objetos* e *RAD (Rapid Application Development)* para este e outros projetos;
- Aprofundar e/ou consolidar o conhecimento das tecnologias utilizadas;

4 Metodologia

A metodologia definida para o desenvolvimento deste trabalho é descrita a seguir, conforme a Proposta de Monografia.

1. Levantamento de requisitos: todos os requisitos serão levantados em reuniões periódicas com a equipe de desenvolvimento, composta por três alunos do curso de Engenharia de Controle e Automação da UFOP e pelo autor deste documento, sempre acompanhados pelo autor do projeto do equipamento, o Engenheiro Metalurgista e de Materiais e Técnico em Eletrônica do Departamento de Engenharia de Controle e Automação da UFOP, Robson Nunes Dal Col;

2. Modelagem dos sistemas: tanto o *firmware*, quanto o *software* aplicativo serão modelados após o levantamento dos requisitos;

3. Estudo das tecnologias: as tecnologias a serem utilizadas (microcontrolador *PIC18F4550*, com suporte a comunicação *USB*, compilador *CCS*, *IDE Microchip MPLAB*; a linguagem de programação, C/C++ e o ambiente de programação, *C++ Builder 6.0*) são requisitos preliminares definidos pelo autor do projeto — em razão da disponibilidade e necessidade de agilidade na conclusão dessas implementações — e serão estudadas durante todo o processo de desenvolvimento, visto que não se tem o domínio pleno de todas elas;

4. Implementação;

5. Realização de testes: diversos testes de operação serão realizados, desde os testes de módulos dos sistemas com o uso de protótipos que serão desenvolvidos pelo Técnico em Eletrônica, até os testes de operação e ajustes no projeto final do equipamento, após sua implantação.

É importante ressaltar, que embora se tenha definido uma sequência de etapas para o desenvolvimento do trabalho, algumas das atividades podem (ou devem) ser realizadas concomitantemente. Pode-se ver na próxima seção, como exemplo, que parte da modelagem foi realizada paralelamente ao levantamento dos requisitos, possíveis módulos do sistema foram implementados e testados e o estudo de parte das tecnologias utilizadas também foi realizado.

5 Desenvolvimento

Os primeiros passos para o desenvolvimento deste projeto foram reuniões realizadas semanalmente, para o levantamento de todos os requisitos de *software*, tanto do *firmware*, quanto do *software* aplicativo. Um modelo de arquitetura de um sistema embutido pode ser visto na figura abaixo. Este projeto se limita às duas camadas superiores da arquitetura: a camada de *Aplicação* e a camada de *Sistema*.

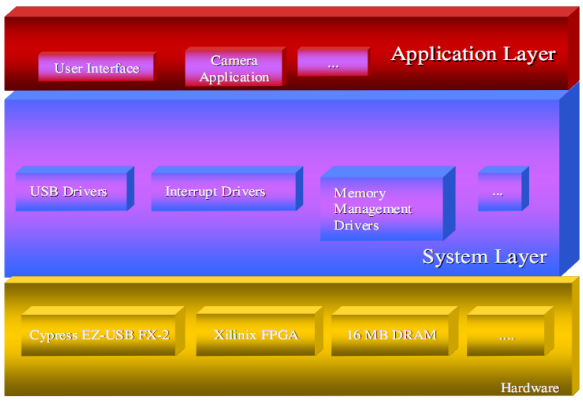

Figura 1: Modelo de Arquitetura de um Sistema Embutido

Para a definição dos requisitos do *software* do sistema embutido, inicialmente buscou-se verificar se o número de recursos de entrada e saída do microcontrolador escolhido — o *PIC 18F4550*, da *Microchip* — realmente seriam suficientes para a realização de todas as operações que se tinha interesse em disponibilizar.

Feito isto, o próximo passo foi definir os conjuntos de variáveis necessárias para o desenvolvimento do *firmware*. Além disso, buscou-se, por meio de diagramas, visualizar todas as informações que deveriam ser trocadas entre *firmware* e *software* aplicativo, para parametrização do sistema embutido e para o monitoramento dos sensores e dos estados de operação dos sistemas. Foram definidos mnemônicos para identificar cada um dos sensores, atuadores que seriam monitorados e controlados, fazendo a correspondência com os pinos de entrada e saída e outros recursos do microcontrolador.

5.1 Primeiras Definições para o Firmware

5.1.1 Definição dos Requisitos Funcionais do Microcontrolador

Na figura abaixo (figura 2), podem ser vistas algumas das definições relativas à variáveis correspondentes aos principais sensores, motores e restrições de movimento do equipamento projetado.

Para possibilitar a realização de todos os experimentos desejados, os conjuntos de placas precisam assumir três posicionamentos básicos nominados conforme o experimento: *Compressão Extensão Simples (CES)*, *Compressão Extensão Oblíquo*

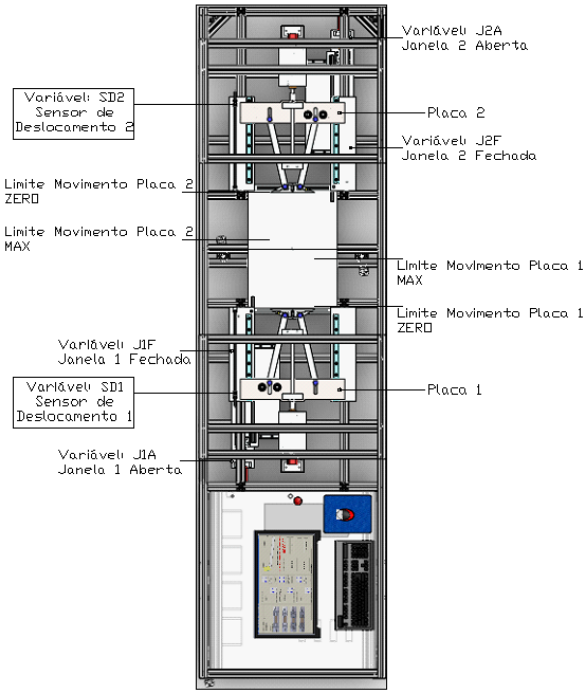

Figura 2: Definição de sensores, atuadores e restrições a serem controlados.

(CEO) e *Transcorrência Simples (TS)*, vide figuras(3, 4, 5).

Entenda-se por compressão o mecanismo de pressão do material em estudo. Nesse experimento, as placas se aproximam e comprimem o material (normalmente areias de cores diferentes, dispostas em camadas). Já a extensão é o mecanismo de afastamento das placas, diminuindo a pressão das placas sobre o material em estudo.

Na *Compreensão Extensão Simples*, os conjuntos devem ser posicionados de maneira que as placas fiquem sempre perpendiculares ao movimento de compressão ou extensão.

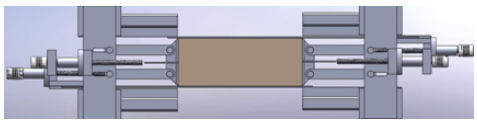

Figura 3: Compressão Extensão Simples

Na *Compressão Extensão Oblíquo*, os conjuntos são posicionados de maneira que as placas fiquem num determinado ângulo em relação à direção do movimento.

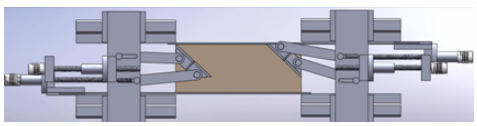

Figura 4: Compressão Extensão Oblíqua

Na *Transcorrência Simples*, apenas uma das placas de cada conjunto se move, enquanto a outra em cada conjunto fica imóvel. As placas devem ser posicionadas de forma perpendicular ao movimento de compressão e extensão.

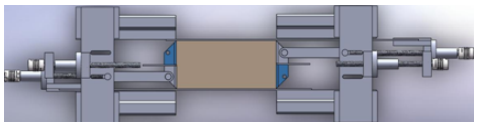

Figura 5: Transcorrência Simples

A partir das análises das funcionalidades necessárias e dos itens a serem controlados ou monitorados no sistema, foram feitas as definições exibidas na figura a seguir (figura 6). Nela vemos a lista de mnemônicos que identificam cada pino e suas características de funcionamento no microcontrolador.

Definição dos Requisitos Funcionais do sistema Microcontrolado (Sexta 17 de Abril 2011, 2:06h):

Sentido de Transmissão: **Firmware → Software**

Sentido de Transmissão: **Software → Firmware**

Mnemônico no Firmware	LEGENDA	Nome no Compilador para Atribuição do Mnemônico	Característica do Pino do PIC 18F4550-I/P	Pino
J1A	Janela 1 Aberta – sensor de final de curso	RA2	E	4
J1F	Janela 1 Fechada – sensor de final de curso	RA3	E	5
J2A	Janela 2 Aberta – sensor de final de curso	RC0	E	15
J2F	Janela 2 Fechada – sensor de final de curso	RC6	E	25
SD1	Sensor de Deslocamento 1	AN5	A/D	8
SD2	Sensor de Deslocamento 2	AN9	A/D	9
ST	Sensor de Temperatura	AN7	A/D	10
SU	Sensor de Umidade	AN0	A/D	2
J1555	Janela 1 555 – habilita rotação do motor	RC7	S	26
J2555	Janela 2 555 – habilita rotação do motor	RB0	S	33
J1SM	Janela 1 - Sentido de Giro do Motor	RB1	S	34
J2SM	Janela 2 - Sentido de Giro do Motor	RB3	S	36
P1SM	Placa 1 – Sentido de Giro do Motor	RB4	S	37
P2SM	Placa 2 – Sentido de Giro do Motor	RB5	S	38
P1PWM	Placa 1 PWM – define velocidade das placas	CCP1	PWM	17
P2PWM	Placa 2 PWM – define velocidade das placas	CCP2	PWM	16
ATIVAP1	ATIVA Placa 1	RB6	S	39
ATIVAP2	ATIVA Placa 2	RB7	S	40
ATIVIDADE	Atividade das placas 1 e 2	RA1	S	3

Figura 6: Relação *Mnemônicos* - *Pinos* do microcontrolador.

Relativo à transferência dos dados entre as camadas do sistema embutido citadas acima, foi elaborado o diagrama a seguir (figura 7), que visa facilicitar o entendimento da comunicação entre a *Camada de Sistema* e a *Camada de Aplicação*.

5.2 Modelagem do Software Aplicativo

5.2.1 Diagrama de Casos de Uso

No Diagrama de Casos de Uso, temos as principais funcionalidades do sistema e seus relacionamentos entre si e com os atores do sistema. A seguir uma descrição de cada Caso de Uso e seus relacionamentos.

Os principais Casos de Uso definidos para o software são:

- Realização de um *Novo Experimento*: Para realizar um novo experimento, o usuário deve criar um novo experiemento, selecionar um tipo de posicionamento, definir uma sequência de movimentos (atribuindo parâmetros a cada movimento), confirmar a lista de movimentos da sequência, posicionar os conjuntos de placas para início do experimento e iniciar o experimento. Os atores autorizados a realizar esta tarefa são: os operadores, os administradores e os usuários de suporte.

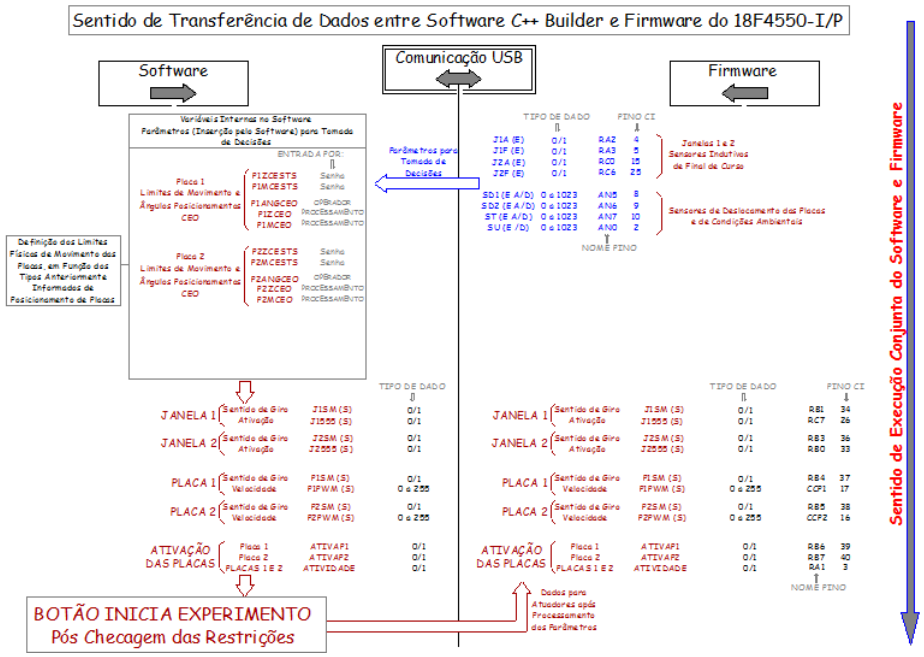

Figura 7: Interface entre *Camada de Sistema* e *Camada de Aplicação* do sistema embutido

- Realização de um *Experimento* existente: Para realizar um experimento já existente, o usuário precisa carregar as configurações do experimento, pode adicionar ou remover movimentos da lista da Sequência configurada, confirmar as alterações, posicionar os conjuntos de placas par ao início do experimento e iniciá-lo. Os atores autorizados a realizar esta tarefa são: os operadores, os administradores e os usuários de suporte.

- Gestão de *Usuários do Sistema*: Esta tarefa se refere a adicionar, remover ou editar um usuário do sistema. Apenas os atores com perfil de *Administrador* e de *Usuário de Suporte* têm acesso à esta tarefa.

- Configuração dos Limites das Placas: A tarefa se refere à edição dos limites previamente difinidos em fase de projeto, dos valores dos limites de movimentação dos conjuntos de placas do equipamento. Apenas os atores com perfil de Usuário de Suporte podem acessar essas configurações.

5.2.2 Diagrama de Classes

O Diagrama de Classes (figura 9) exibe as definições preliminares das principais classes que devem ser implementadas no sistema e seu relacionamentos.

Inicialmente, as classes definidas foram:

- Experimento - a classe tem seus campos relacionados ao experimento que se deseja configurar no sistema.

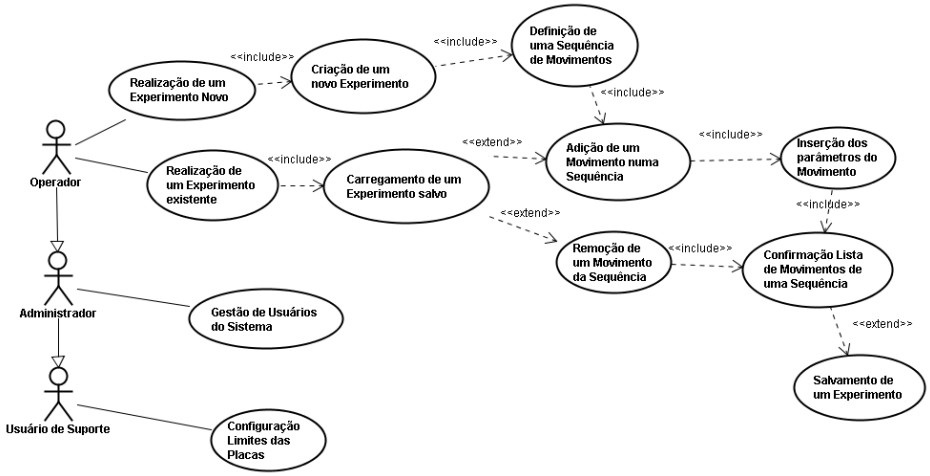

Figura 8: Diagrama de Casos de uso

- Sequencia - esta classe representa uma sequência de movimentos que devem ser configurados no sistema. Um objeto dessa classe deve possuir pelo um da classe Movimento.

- Movimento - esta classe é definida pelos campos associados aos movimentos que devem ser configurados no sistema, para a realização de um experimento. Um objeto dessa classe deve possui exatamente dois objetos da classe CnjPlacas.

- CnjPlacas - esta classe representa um conjunto contendo um par de placas instaladas em cada extremidade da máquina. Um objeto CnjPlacas deve possuir exatamente um par de objetos da classe Placa.

- Placa - esta classe representa cada uma das placas que, duas a duas compõem um dos dois conjuntos de placas do equipamento. Aos objetos dessa classe é que são associados os campos relacionados à posição de início, posição final, comprimento e posição corrente das placas de uma extremidade do equipamento.

Outras classes e relacionamentos podem (ou devem) ser criados para que tenhamos a mais fiel representação possível para a modelagem do *software* aplicativo. Novas avaliações do modelo deverão ser realizadas, dentro do cronograma proposto, para a conclusão dessas definições.

5.2.3 Diagramas de Atividades

Os *Diagramas de Atividades* buscam representar as principais atividades realizadas no *software* aplicativo, durante as etapas de configuração e execução dos experimentos.

As figuras 10 e 11 exibem alguns desses diagramas. Neles podemos visualizar qual deve ser a sequência de eventos que o sistema deve ter implementada, para a realização de uma tarefa. Diagramas como estes, para as demais tarefas devem ser modelados para cada uma das funcionalidades do sistema.

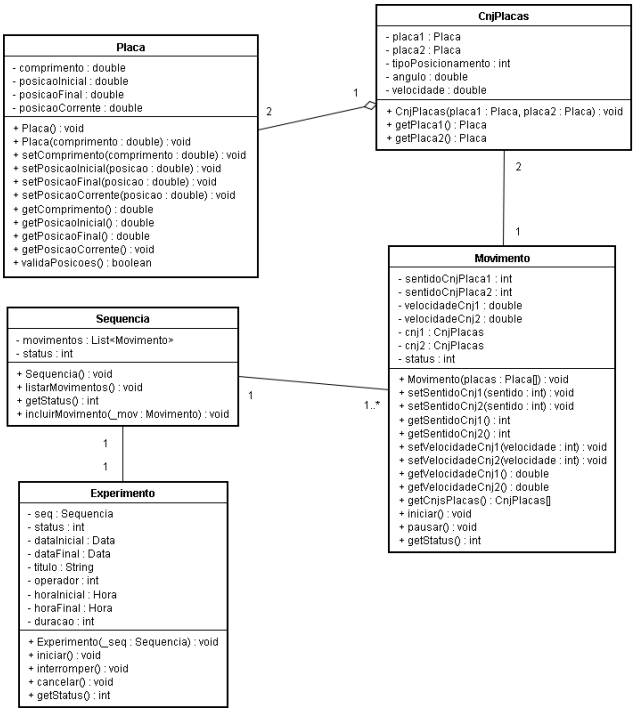

Figura 9: Diagrama de Classes

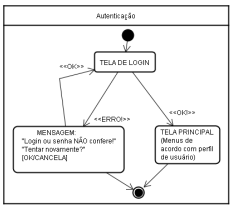

Figura 10: Efetuando *LOGIN*

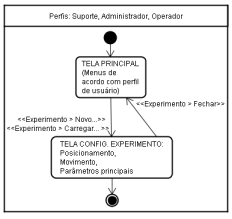

Figura 11: Criando/Carregando configurações de um *Experimento*.

5.2.4 Esboço de Interfaces de Usuário

Abaixo podemos ver o esboço de algumas interfaces gráficas de usuário, que devem ser projetadas e implementadas no sistema (figuras 12, 13, 14, 15).

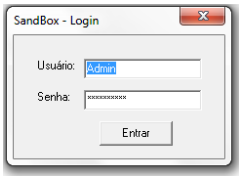

Figura 12: Tela de *Login* do Sistema

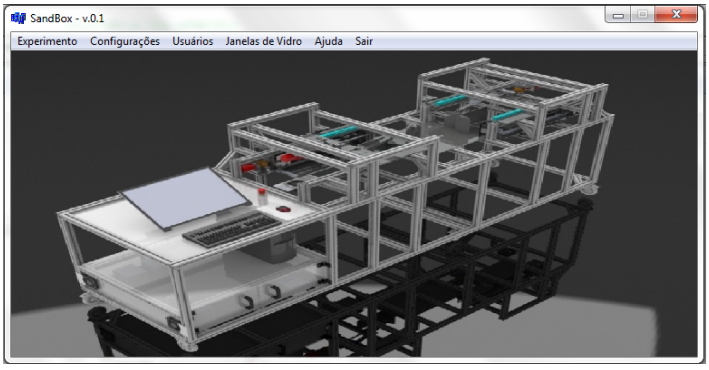

Figura 13: Tela inicial de abertura do sistema.

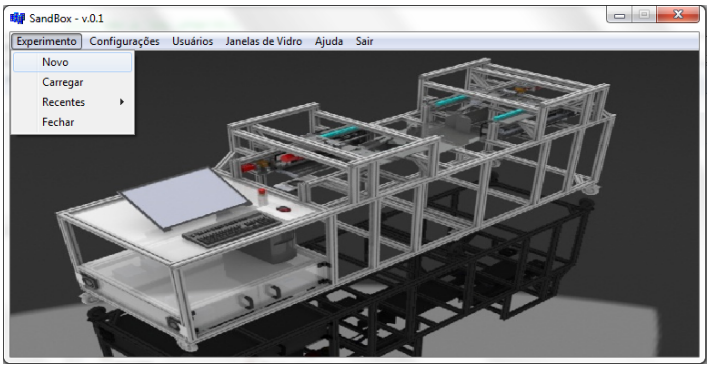

Figura 14: Selecionando o item *Novo* no menu *Experimento*.

5.3 Definições Sobre a Comunicação *USB*

Neste momento serão descritas algumas observações sobre a implementação da comunicação via interface *USB* (*Universal Serial Bus*) com o *Firmware*, utilizando as API's *Windows* de comunicação serial e biblioteca *usb_cdc*.

Para o microcontrolador, serão utilizadas bibliotecas em linguagem *C*, que implementam o modo *CDC* (*Communication Device Class*) de comunicação *USB*.

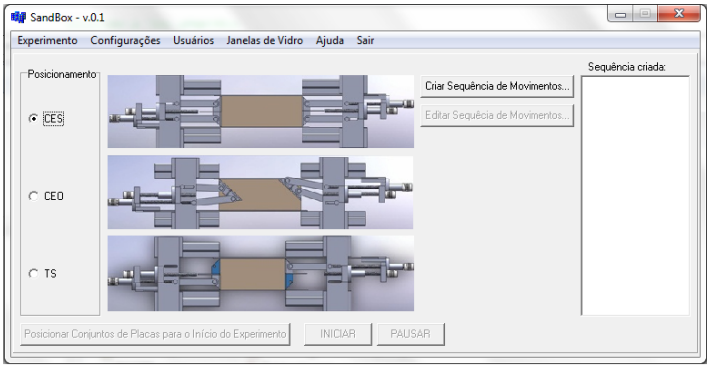

Figura 15: Tela de um novo *Experimento*

O modo *CDC* foi desenvolvido pela empresa *Microchip* para auxiliar programadores e projetistas de sistemas embutidos a migrar com facilidade, suas aplicações do padrão *RS-232* (padrão serial comum) para o padrão *USB*. O modo trabalha emulando uma porta serial comum (*COM*) no computador em que o dispositivo for instalado. Para isso, um *driver* disponibilizado pelo fabricante precisa ser utilizado.

Quando um sistema embutido, utilizando as implementações do modo *CDC* para comunicação *USB*, é conectado ao computador, o sistema operacional o detecta e solicita o *driver* de comunicação para emular uma porta serial comum. Ao instalar o dispositivo, o sistema operacional passa a ter uma nova porta serial disponível (uma porta virtual - *Virtual ComPort*).

O esquema de *hardware* para utilização da interface *USB* com um microcontrolador *PIC* pode ser visualizado a seguir. Serão utilizadas as bibliotecas C "usb_cdc.h"e "usb_cdc_firm.h", cujos conteúdos encontram-se disponíveis no *Apêndice B*.

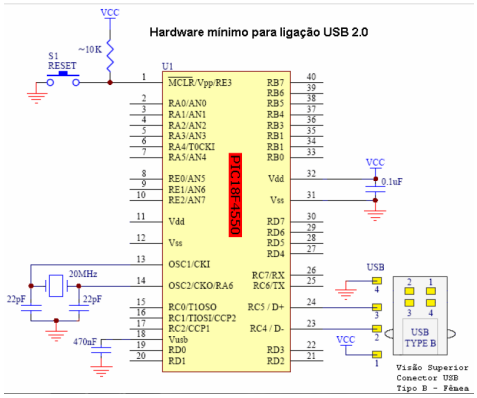

Figura 16: *Hardware* básico para conexão do *PIC* com uma interface *USB*.

A implementação da comunicação USB no *firmware* utilizando as bibliotecas desenvolvidas pela *Microchip* é feita, basicamente, através do uso de suas principais funções:

- `usb_cdc_getc()`

  - que lê um caracter recebido pelo *buffer* de dados do canal de comunicação;

- `usb_cdc_putc(char c)`

  - que envia um caracter para o *buffer* de dados do canal de comunicação;

- `usb_cdc_connected()`

  - que retorna TRUE, se o canal estiver pronto para recepção de dados;

- `usb_cdc_kbhit()`

  - que retorna TRUE, se exite um ou mais caracteres recebidos, aguardando no *buffer* de dados para processamento;

Abaixo, podemos ver um pequeno exemplo de uso dessas funções num código-fonte de um *firmware* para microcontrolador *PIC*. O compilador utilizado para testar foi o *CCS PCWH Compiler*, integrado ao ambiente *Microchip MPLAB IDE versão 8.5*.

```
#include <18F4550.h> //Carrega o cabeçalho do 18F4550
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN

/* Considerando que o cristal em uso seja o de 20MHz, estes fuses estão
    usando o multiplicador PLL
descrito na seção Configuração do Clock, para gerar o clock de 48MHz
    necessário ao funcionamento do
USB*/

#use delay(clock=48000000) // Seta o clock do PIC para 48MHz
#include "usb_cdc.h" /* Carrega o arquivo que possui as funções e o
    firmware necessários para o uso do
USB*/

#include <stdlib.h> // Disponibiliza o uso da função itoa

// Declaração das variáveis
int8 cont=0;
int8 segundos=0;
char envia[4];

#int_timer0 //Trata a iterrupção gerada pelo TIMER 0
void trata_tmr0 ()
{
  cont++;
  if (cont >= 23)
  {
    itoa(segundos,10,envia); // Converte a variável inteira segundos para
        char
    cont = 0;
    printf(usb_cdc_putc,envia); // Envia por USB a string envia para o
        computador
    segundos++;
  }
}

void main()
{
  // Estrutura básica para inicialização do USB
  delay_ms(300);
```

```
    usb_cdc_init();// Inicia o modo CDC
    usb_init();// Inicia a comunicação USB
    while (!usb_cdc_connected()){} // Só começa o programa quando o PIC for
        plugado ao PC
    setup_timer_0 (RTCC_DIV_8);
    enable_interrupts (INT_TIMER0);
    while (true);
}
```

Programa 1: Exemplo: Implementação para uso de *USB* num *PIC* no modo *CDC*.

Na camada de aplicação do sistema embutido, ou seja, no *software* aplicativo, a implementação da comunicação serial será feita utilizando *API's* de comunicação serial *Win32*. Um exemplo do uso dessas *API's*, baseado em [1] e em [2], pode ser visto no *Apêndice C*.

Não é recomendável utilizar tecnologias proprietárias para o desenvolvimento de um projeto de desenvolvimento de *software*. Entretanto, em sistemas embutidos, normalmente os projetos são bem específicos para o fim que se deseja. Especialmente neste projeto, por se tratar de uma primeira versão e considerando que todo o *hardware* já estava previamente especificado, inclusive o microcomputador que será utilizado, as restrições quanto a eventuais necessidades de portabilidade ou problemas com a dependência em relação à tecnologias não foram consideradas.

5.4 Definição das Etapas de Operação do Sistema

O sistema mecânico é composto por dois conjuntos de placas, instalados transversalmente. Cada conjunto é composto por um par de placas. Os conjuntos de placas são completamente independentes. O sistema pode conter placas de comprimentos diferentes, que podem ser instaladas em diferentes ângulos de posicionamento, conforme o tipo de movimento escolhido para o experimento.

Em razão disso, o primeiro passo para a operação da máquina é a definição dos parâmetros de posicionamento, velocidade, ângulo dos conjuntos de placas e o tipo do movimento que deseja efetuar.

Ficaram assim definidas as principais etapas para operação da máquina para realização de um ensaio (experimento), utilizando o *software* aplicativo:

1. Criar um novo experimento;
2. Selecionar o tipo de osicionamento desejado (CES, CEO ou TS);
3. Criar uma sequência de movimentos;
4. Posicionar placas para início do experimento;
5. Iniciar o experimento;

Um experimento pode ser composto de uma sequência de movimentos. Portanto, para que uma Sequência seja criada, os passos a seguir devem ser realizados:

1. Selecionar um tipo de *Movimento*;

2. Preencher campos com os parâmetros (*Posição inicial*, *posição final*, *velocidade* e *ângulo*, quando for o caso);

3. Clicar no botão *Adicionar* para incluir o Movimento configurado na sequência;

4. Confirmar a Sequência;

Note-se que a qualquer momento durante o processo de criação da sequência, um movimento qualquer pode ser removido da lista, usando-se o botão *Remover*. Após confirmada a sequência de movimentos, os próximos passos são "Posicionar Placas para Início do Experimento"e iniciar o experimento.

5.4.1 Detalhes de Implementação

Na inserção de um novo movimento na sequência, o sistema deve criar um objeto do tipo *Movimento*, com todos os parâmetros passados através dos campos do formulário e adicioná-lo à lista de Movimentos do objeto *Sequencia* correspondente.

Na remoção, pega-se o índice do item selecionado no *ListBox*, retirando da lista de Movimentos do objeto Sequência, o item de mesmo índice.

Um mecanismo de verificação e validação dos parâmetros para o experimento deve ser implementado, de maneira a evitar que tente fazer um experimento com parâmetros fora dos limites estabelecidos ou não informados.

5.5 Alguns Resultados

5.5.1 Desenvolvimento de um Simulador

Como resultado do processo de avaliar o entendimento das etapas a serem seguidas pelo operador dos sistemas, durante a configuração e realização de um experimento e, ao mesmo tempo, verificar a viabilidade de aplicação no projeto, dos paradigmas de *Programação Orientada a Objetos*, *Programação Orientada a Eventos* e *RAD (Rapid Application Development)*, um pequeno simulador foi desenvolvido.

O aplicativo foi desenvolvido na linguagem C++, no ambiente de programação *Borland C++ Builder 6.0*. As declarações das classes utilizadas podem ser visualizadas logo abaixo. As implementações referentes à definição da classe *Movimento* e os eventos de formulário que possibilitam a simulação do experimento, podem ser vistas no *Apêndice A*.

```
class Placa
{
        private:
                //Declaração dos campos do objeto
                double posLimitMax;
                double posLimitMin;
                double posInicial;
```

```
                double posFinal;
                double posCorrente;
                double comprimento;
                double angulo;
        public:
                //Declaração do construtor e métodos do objeto
                Placa();
                double getPosLimitMin();
                double getPosLimitMax();
                double getPosInicial();
                double getPosFinal();
                double getPosCorrente();
                double getComprimento();
                double getAngulo();
                void setPosLimitMin(double);
                void setPosLimitMax(double);
                void setPosInicial(double);
                void setPosFinal(double);
                void setPosCorrente(double);
                void setAngulo(double);
                void setComprimento(double);
};
```

Programa 2: Declaração da classe *Placa*.

```
#include "Placa.h"

class ConjuntoPlacas
{
        private:
                Placa placas[2];

        public:
                ConjuntoPlacas();
                void setPlaca1(Placa placa);
                void setPlaca2(Placa placa);
                Placa getPlaca1();
                Placa getPlaca2();
                Placa* getPlacas();
};
```

Programa 3: Declaração da classe *ConjuntoPlacas*.

```
#include "ConjuntoPlacas.h"

#define MINCPC 0 //posição limite mínima para o Cisalhamento Puro -
    Compressão
#define MAXCPC 100 //posição limite máxima para o CPC
#define MINCPE 0 //posição limite mínima para o Cisalhamento Puro -
    Expansão
#define MAXCPE 100 //posição limite máxima para o CPE
#define MINCSC 0 // posição limite mínima para o Cisalhamento Simples
    Combinado
#define MAXCSC 50 //posição limite máxima para o CSC
#define MINCSP 0 //posição limite mínima para o Cisalhamento Simples
    Positivo
#define MAXCSP 100 //posição limite máxima para o CSP
```

```
#define MINCSN 0 //posição limite mínima para o Cisalhamento Simples
    Negativo
#define MAXCSN 50 //posição limite máxima para o CSN

class Movimento
{
        private:
                ConjuntoPlacas cnjPlacas1, cnjPlacas2;
                bool status;
                int tipo;
        public:
               Movimento();
               Movimento(ConjuntoPlacas placas1, ConjuntoPlacas placas2);
               void iniciar();
               void interromper();
               bool getStatus();
               void setTipo(int);
               void setCnjPlacas1(ConjuntoPlacas);
               ConjuntoPlacas getCnjPlacas1();
               void setCnjPlacas2(ConjuntoPlacas);
               ConjuntoPlacas getCnjPlacas2();
               int getTipo();
};
```

Programa 4: Declaração da classe *Movimento*.

O simulador permite visualizar cada uma das etapas básicas que qualquer operador do sistema necessitará seguir, para conseguir configurar e iniciar a execução de um experimento. É necessário destacar que o simulador não contempla todas as funcionalidades necessárias para uma operação real. Ele apenas facilita o entendimento para o processo de desenvolvimento e permite estabelecer uma ordem preliminar para essas etapas, a fim de que todas as configurações necessárias tenham sejam realizadas antes do início da operação. As figuras a seguir ilustram essas etapas.

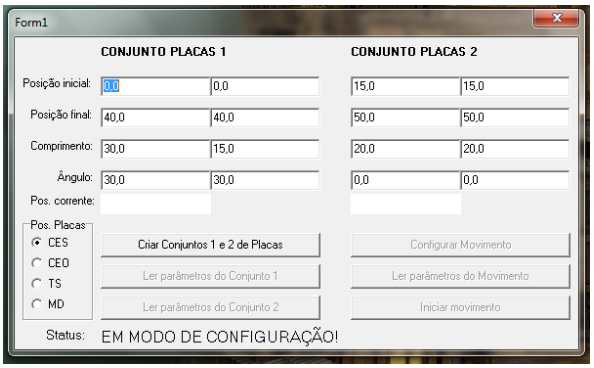

Figura 17: Tela inicial do simulador.

Na tela inicial do simulador, podem ser vistos campos de texto referentes aos atributos dos objetos da classe *Placa* (*Posição inicial*, *Posição final*, *Comprimento*, *Ângulo*).

Os campos *Posição inicial* e *Posição final* se referem, respectivamente, às posições inicial e final do conjunto de placas da máquina durante a operação. Ambos os conjuntos de placas precisam ser configurados. Os campos *Comprimento* e *Ângulo*

se referem a cada uma das placas e ao seu posicionamento inicial, respectivamente, dependendo do experimento desejado.

Durante o experimento, os conjuntos de placas se movimentam. Ambos para o centro da máquina, ambos para as extremidades da máquina, ou um conjunto para o centro e outro para uma das extremidades. O movimento depende do experimento que se deseja realizar.

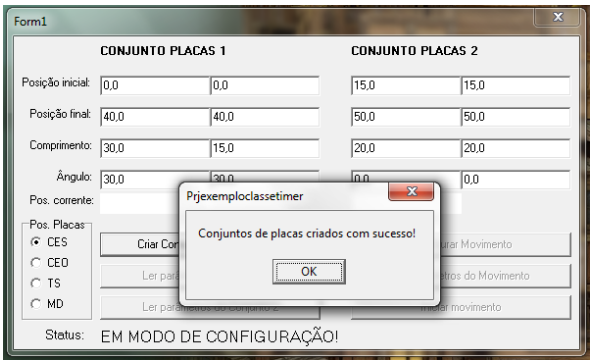

Figura 18: Verificando conjuntos de placas.

Um botão "Criar Conjuntos de Placas 1 e 2"foi definido para criar dois pares de objetos da classe *Placa*, criar dois objetos da classe *ConjuntoPlacas*, atribuir um par de objetos *Placa* a cada um dos objetos do tipo *ConjuntoPlacas*. Ao ser selecionado, a implementação verifica se os objetos foram criados e exibe a mensagem, conforme a figura 18.

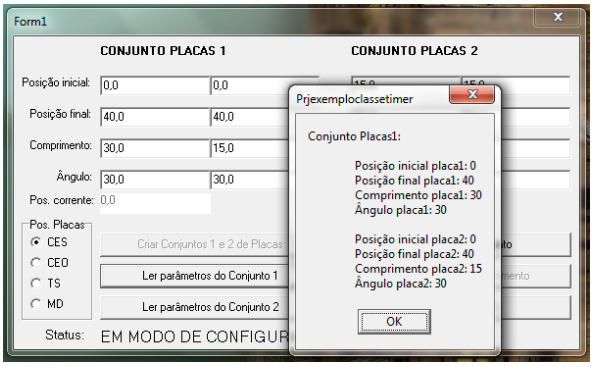

Figura 19: Exibindo os valores dos objetos.

Os botões "Ler parâmetros do Conjunto 1"e "Ler parâmetros do Conjunto 2"foram definidos com objetivo de certificar se os objetos foram realmente criados e seus campos podem ser acessados. Portanto, ao selecionar um dos dois botões, uma mensagem é exibida com os valores dos campos dos objetos correspondentes (ver figura 19).

A figura 20 exibe o resultado da seleção do botão "Configurar Movimento". O código para este botão cria um objeto da classe *Movimento*, atribuindo-lhe os conjuntos de placas criados. Nesse momento é possível observar que o status do

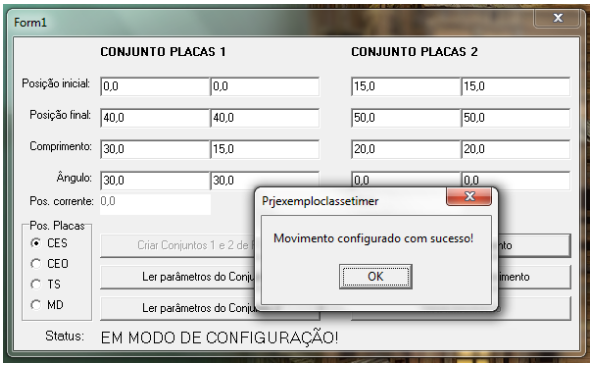

Figura 20: Configuração do Movimento.

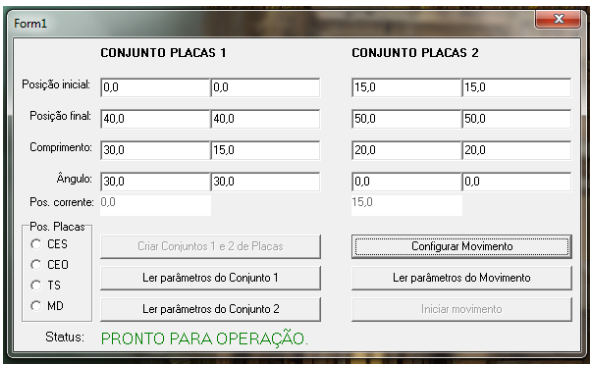

Figura 21: *Status* após configuração do movimento.

programa foi alterado de "EM MODO DE CONFIGURAÇÃO"(figura 20) para "PRONTO PARA OPERAÇÃO"(figura 21), com o texto na cor verde. O botão "Ler parâmetros Movimento"faz novamente a leitura dos valores dos campos dos objetos, agora agrupados como conjuntos. Mais uma vez pode-se constatar a corretude na manipulação dos objetos.

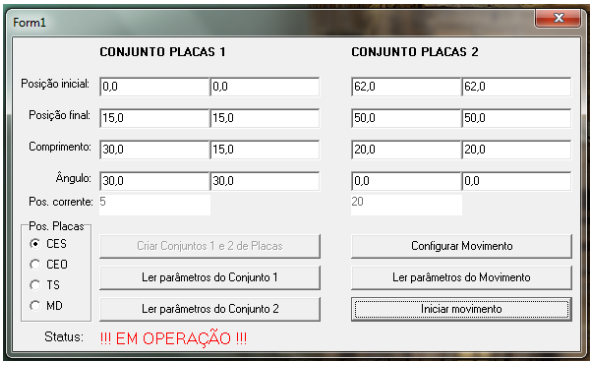

Figura 22: Máquina em operação.

Ao clicar no botão "Iniciar Movimento", o status muda para "EM OPERAÇÃO"(ver figura 22) e os campos *Posição corrente* dos dois conjuntos são atualizados, conforme o movimento configurado. Se os campos *Posição final* receberem valores maiores do que os campos *Posição inicial*, o valor da posição corrente do conjunto

é decrementado da posição inicial até a posição final. Do contrário, o valor é incrementado da posição inicial até a posição final.

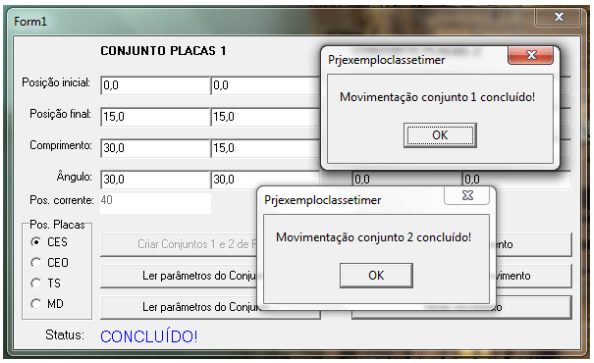

Figura 23: Simulação concluída

A atualização ocorre a cada 1 segundo, para simular a movimentação e, ao final, de forma independente, uma mensagem é exibida, informando que o movimento do conjunto foi concluído. Quando isso ocorrer para os dois conjuntos, o status da operação é alterado para "CONCLUÍDO"(ver figura 23).

5.6 Próximos Passos

Os próximos passos previstos para o desenvolvimento deste projeto são:

- Concluir o processo de análise de requisitos tanto do sistema embutido quanto do *software* de interface com o usuário, respeitando o cronograma estabelecido;
- A partir da conclusão das análises de requisitos, concluir toda a modelagem dos sistemas, também dentro do cronograma;
- Iniciar estudo mais aprofundado das tecnologias selecionadas para a implementação deste projeto, sobretudo às que se referem às *API's* de interface de comunicação entre o *firmware* e o *software* instalado no PC;

6 Cronograma de atividades

Na Tabela 1, temos o cronograma proposto para a realização de todas as etapas do desenvolvimento deste projeto.

Como se pode ver, a etapa de levantamento e análise de requisitos (*Requisitos*) está praticamente concluída, embora os documentos elaborados ainda possam sofrer alterações, baseadas nas avaliações periódicas que são realizadas. A modelagem dos sistemas também já está adiantada, muito embora, no que se refere ao desenvolvimento do *software* aplicativo, muitos outros diagramas e análises podem/devem ser realizadas, com o fim de facilitar o trabalho de implementação.

A atividade *Bibliografia*, conforme descrito na *Proposta de Monografia*, contempla a pesquisa e revisão de material bibliográfico relacionado às tecnologias que estão sendo utilizadas no projeto, incluindo manuais, tutoriais, *datasheets* (folhas de dados) de microcontroladores e outros componentes eletrônicos, e livros sobre as linguagens de programação que serão utilizadas, disponíveis nos laboratórios do Departamento de Engenharia de Controle e Automação. Foi iniciada, no entanto, ainda está em processo de levantamento. Conforme cronograma, pretende-se dar maior foco a esta atividade num segundo momento, com o intuito de contextualizar o projeto com o curso de graduação e as áreas relacionadas dentro da Ciência da Computação.

Os testes (*Testes modulares e de ajuste*)terão início a partir do momento em que versões ou módulos dos sistemas de *software* e *hardware* de controle e/ou protótipos do equipamento tiverem sido concluídos.

Atividades	**Abr**	**Mai**	**Jun**	**Jul**	**Ago**	**Set**	**Out**	**Nov**	**Dez**
Requisitos	X	X	X	X					
Modelagem		X	X	X					
Implementação			X	X	X				
Testes modulares e de ajustes				X	X				
Bibliografia	X	X	X	X	X	X			
Redigir a Monografia							X	X	X
Apresentação do Trabalho									X

Tabela 1: Cronograma de Atividades.

Referências

[1] MESSIAS, Antônio Rogério. *Conectando 8 Teclados na Porta Serial Através de um Microcontrolador PIC 16F877 Para Controle de Acessos*. Disponível em http://www.rogercom.com/index.htm, acessado em 27/04/2011.

[2] FILHO, Constantino Seixas. *Comunicação Serial - Capítulo 2 - Apostila adaptada a partir do texto: Allen Denver, Serial Communications in Win32, Microsoft Windows Developer Support, 1995*. UFMG-Departamento de Engenharia Eletrônica. Disponível em http://www.cpdee.ufmg.br/ seixas/PaginaSDA/Download/DownloadFiles/-Serial.PDF, acessado em 29/05/2011.

[3] JUCÁ, Sandro. *Apostila de Microcontroladores PIC e Periféricos*. Disponível em www.tinyurl.com/SanUSB, acessado em 23/05/2011.

[4] SANTANA, Lucas Vago. *Tutorial de Implementação da Interface de Comunicação USB 2.0 utilizando o PIC18F4550*. Departamento de Engenharia de Controle e Automação, Escola de Minas, UFOP, 2007.

[5] *PIC18F2455/2550/4455/4550 Data Sheet*. Microchip Technology Inc., 2007.

A Códigos-fonte do Simulador

A.1 Classe Movimento

```
#include "Movimento.h"
//Definição dos contrutores e métodos da classe Movimento

/**
 * Construtor padrão do objeto
 */
Movimento::Movimento(){ }
/**
 * Construtor com parâmetro
 */
Movimento::Movimento(ConjuntoPlacas placas1, ConjuntoPlacas placas2)
{
        this->cnjPlacas1 = placas1;
        this->cnjPlacas2 = placas2;
        this->status = false;
}
/**
 * Método que inicia o movimento
 */
void Movimento::iniciar()
{
        this->status = true;
}

/**
 * Método que interrompe o movimento
 */
void Movimento::interromper()
{
        this->status = false;
}

/**
 * Método que retorna o status do movimento: [true / false]
 */
bool Movimento::getStatus()
{
        return this->status;
}

/**
 * Método que seta o tipo de movimento: [0-CPC, 1-CPE, 2-CSC, 3-CSP, 4-
    CSN]
 */
void Movimento::setTipo(int tipo)
{
        this->tipo = tipo;
}

/**
 * Método que retorna o tipo de movimento: [0-CPC, 1-CPE, 2-CSC, 3-CSP,
    4-CSN]
```

```
 */
int Movimento::getTipo()
{
        return this->tipo;
}

/**
 * Método que atribui um conjunto de placas ao campo cnjPlacas1 do objeto
 */
void Movimento::setCnjPlacas1(ConjuntoPlacas cnjPlacas)
{
        this->cnjPlacas1.setPlaca1(cnjPlacas.getPlaca1());
        this->cnjPlacas1.setPlaca2(cnjPlacas.getPlaca2());
}

/**
 * Método que atribui um conjunto de placas ao campo cnjPlacas2 do objeto
 */
void Movimento::setCnjPlacas2(ConjuntoPlacas cnjPlacas)
{
        this->cnjPlacas2.setPlaca1(cnjPlacas.getPlaca1());
        this->cnjPlacas2.setPlaca2(cnjPlacas.getPlaca2());
}

/**
 * Método que retorna o conjunto 1 de placas do objeto
 */
ConjuntoPlacas Movimento::getCnjPlacas1()
{
        return this->cnjPlacas1;
}

/**
 * Método que retorna o conjunto 2 de placas do objeto
 */
ConjuntoPlacas Movimento::getCnjPlacas2()
{
        return this->cnjPlacas2;
}
```

Programa 5: Definição da classe Movimento.

A.2 Eventos de Formulário

```
//---------------------------------------------------------------------------


#include <vcl.h>
#pragma hdrstop

#include "ExemploClasse.h"
#include "Movimento.h"
//---------------------------------------------------------------------------

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
Placa placa1, placa2;
ConjuntoPlacas cnjPlacas1, cnjPlacas2;
Movimento movimento;
double sensorDeslocCnj1, sensorDeslocCnj2;//Simulam sensores de
    deslocamento das placas
double posCnj1, posCnj2;//receberão valores padrões de acordo com o
    posicionamento (CES, CEO, TS, MD)
bool cnj1Concluido, cnj2Concluido = false;
bool isRed = false;

//---------------------------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
        //Atribuindo valores aos campos do conjunto de placas 1
        placa1.setPosInicial(StrToFloat(posIniP1Cnj1->Text));
        placa1.setPosFinal(StrToFloat(posFimP1Cnj1->Text));
//        placa1.setPosCorrente(StrToFloat(posCorrenteP1Cnj1->Text));
        placa1.setComprimento(StrToFloat(comprP1Cnj1->Text));
        placa1.setAngulo(StrToFloat(angP1Cnj1->Text));
        cnjPlacas1.setPlaca1(placa1);

        placa2.setPosInicial(StrToFloat(posIniP2Cnj1->Text));
        placa2.setPosFinal(StrToFloat(posFimP2Cnj1->Text));
//        placa2.setPosCorrente(StrToFloat(posCorrenteP2Cnj1->Text));
        placa2.setComprimento(StrToFloat(comprP2Cnj1->Text));
        placa2.setAngulo(StrToFloat(angP2Cnj1->Text));
        cnjPlacas1.setPlaca2(placa2);

        //Atribuindo valores aos campos do conjunto de placas 2
        placa1.setPosInicial(StrToFloat(posIniP1Cnj2->Text));
        placa1.setPosFinal(StrToFloat(posFimP1Cnj2->Text));
```

```
//          placa1.setPosCorrente(StrToFloat(posCorrenteP1Cnj2->Text));
        placa1.setComprimento(StrToFloat(comprP1Cnj2->Text));
        placa1.setAngulo(StrToFloat(angP1Cnj2->Text));
        cnjPlacas2.setPlaca1(placa1);

        placa2.setPosInicial(StrToFloat(posIniP2Cnj2->Text));
        placa2.setPosFinal(StrToFloat(posFimP2Cnj2->Text));
//          placa2.setPosCorrente(StrToFloat(posCorrenteP2Cnj2->Text));
        placa2.setComprimento(StrToFloat(comprP2Cnj2->Text));
        placa2.setAngulo(StrToFloat(angP2Cnj2->Text));
        cnjPlacas2.setPlaca2(placa2);

        ShowMessage("Conjuntos de placas criados com sucesso!");
        Button2->Enabled = true;
        Button3->Enabled = true;
        Button4->Enabled = true;
        posCorrente1->Text = posIniP1Cnj1->Text;
        sensorDeslocCnj1 = StrToFloat(posCorrente1->Text);
        posCorrente2->Text = posIniP1Cnj2->Text;
        sensorDeslocCnj2 = StrToFloat(posCorrente2->Text);
        Button1->Enabled = false;
}
//
    ---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
        String posIniP1C1 = FloatToStr( cnjPlacas1.getPlaca1().
            getPosInicial() );
        String posFimP1C1 = FloatToStr( cnjPlacas1.getPlaca1().
            getPosFinal() );
        String compP1C1 = FloatToStr( cnjPlacas1.getPlaca1().
            getComprimento() );
        String angP1C1 = FloatToStr( cnjPlacas1.getPlaca1().getAngulo() )
            ;

        String posIniP2C1 = FloatToStr( cnjPlacas1.getPlaca2().
            getPosInicial() );
        String posFimP2C1 = FloatToStr( cnjPlacas1.getPlaca2().
            getPosFinal() );
        String compP2C1 = FloatToStr( cnjPlacas1.getPlaca2().
            getComprimento() );
        String angP2C1 = FloatToStr( cnjPlacas1.getPlaca2().getAngulo() )
            ;

        String mensagem = "Conjunto Placas1:\n\n";
        mensagem = mensagem + "\tPosição inicial placa1: " + posIniP1C1;
        mensagem = mensagem + "\n\tPosição final placa1: " + posFimP1C1;
        mensagem = mensagem + "\n\tComprimento placa1: " + compP1C1;
        mensagem = mensagem + "\n\tÂngulo placa1: " + angP1C1;
        mensagem = mensagem + "\n\n\tPosição inicial placa2: " +
            posIniP2C1;
        mensagem = mensagem + "\n\tPosição final placa2: " + posFimP2C1;
        mensagem = mensagem + "\n\tComprimento placa2: " + compP2C1;
        mensagem = mensagem + "\n\tÂngulo placa2: " + angP2C1;
        ShowMessage( mensagem );
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
        String posIniP1C2 = FloatToStr( cnjPlacas2.getPlaca1().
            getPosInicial() );
        String posFimP1C2 = FloatToStr( cnjPlacas2.getPlaca1().
            getPosFinal() );
        String compP1C2 = FloatToStr( cnjPlacas2.getPlaca1().
            getComprimento() );
        String angP1C2 = FloatToStr( cnjPlacas2.getPlaca1().getAngulo() )
            ;

        String posIniP2C2 = FloatToStr( cnjPlacas2.getPlaca2().
            getPosInicial() );
        String posFimP2C2 = FloatToStr( cnjPlacas2.getPlaca2().
            getPosFinal() );
        String compP2C2 = FloatToStr( cnjPlacas2.getPlaca2().
            getComprimento() );
        String angP2C2 = FloatToStr( cnjPlacas2.getPlaca2().getAngulo() )
            ;

        String mensagem = "Conjunto Placas2:\n\n";
        mensagem = mensagem + "\tPosição inicial placa1: " + posIniP1C2;
        mensagem = mensagem + "\n\tPosição final placa1: " + posFimP1C2;
        mensagem = mensagem + "\n\tComprimento placa1: " + compP1C2;
        mensagem = mensagem + "\n\tÂngulo placa1: " + angP1C2;
        mensagem = mensagem + "\n\n\tPosição inicial placa2: " +
            posIniP2C2;
        mensagem = mensagem + "\n\tPosição final placa2: " + posFimP2C2;
        mensagem = mensagem + "\n\tComprimento placa2: " + compP2C2;
        mensagem = mensagem + "\n\tÂngulo placa2: " + angP2C2;
        ShowMessage( mensagem );
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
//        movimento = Movimento( cnjPlacas1, cnjPlacas2 );
        movimento.setCnjPlacas1(cnjPlacas1);
        movimento.setCnjPlacas2(cnjPlacas2);
        posCnj1 = 0;
        posCnj2 = 0;
        Button5->Enabled = true;
        ShowMessage("Movimento configurado com sucesso!");
        Label8->Caption = "PRONTO PARA OPERAÇÃO.";
        Label8->Font->Color = clGreen;
        Label8->Font->Style << fsBold;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
        ConjuntoPlacas cnj1, cnj2;
```

```
        cnj1 = movimento.getCnjPlacas1();
        cnj2 = movimento.getCnjPlacas2();

        String mensagem = "Conjunto1: ";
        mensagem += "\n\tPosicao inicial: " + FloatToStr(cnj1.getPlaca1()
            .getPosInicial());
        mensagem += "\n\tPosicao final: ";
        mensagem += FloatToStr(cnj1.getPlaca1().getPosFinal());
        mensagem += "\n\tComprimento: ";
        mensagem += FloatToStr(cnj1.getPlaca1().getComprimento());
        mensagem += "\n\tÂngulo: ";
        mensagem += FloatToStr(cnj1.getPlaca1().getAngulo());
        mensagem += "\n\n\tPosicao inicial: ";
        mensagem += FloatToStr(cnj1.getPlaca2().getPosInicial());
        mensagem += "\n\tPosicao final: ";
        mensagem += FloatToStr(cnj1.getPlaca2().getPosFinal());
        mensagem += "\n\tComprimento: ";
        mensagem += FloatToStr(cnj1.getPlaca2().getComprimento());
        mensagem += "\n\tÂngulo: ";
        mensagem += FloatToStr(cnj1.getPlaca2().getAngulo());

        mensagem += "\nConjunto2: ";
        mensagem += "\n\tPosicao inicial: ";
        mensagem += FloatToStr(cnj2.getPlaca1().getPosInicial());
        mensagem += "\n\tPosicao final: ";
        mensagem += FloatToStr(cnj2.getPlaca1().getPosFinal());
        mensagem += "\n\tComprimento: ";
        mensagem += FloatToStr(cnj2.getPlaca1().getComprimento());
        mensagem += "\n\tÂngulo: ";
        mensagem += FloatToStr(cnj2.getPlaca1().getAngulo());
        mensagem += "\n\n\tPosicao inicial: ";
        mensagem += FloatToStr(cnj2.getPlaca2().getPosInicial());
        mensagem += "\n\tPosicao final: ";
        mensagem += FloatToStr(cnj2.getPlaca2().getPosFinal());
        mensagem += "\n\tComprimento: ";
        mensagem += FloatToStr(cnj2.getPlaca2().getComprimento());
        mensagem += "\n\tÂngulo: ";
        mensagem += FloatToStr(cnj2.getPlaca2().getAngulo());

        ShowMessage(mensagem);
        Button6->Enabled = true;
}
//
    ---------------------------------------------------------------------------


void __fastcall TForm1::Button6Click(TObject *Sender)
{
        Timer1->Enabled = true;
}
//
    ---------------------------------------------------------------------------


void __fastcall TForm1::Timer1Timer(TObject *Sender)
```

```
{
        if (isRed)
        {
                Label8->Font->Color = clWhite;
                isRed = false;
        }
        else
        {
                Label8->Font->Color = clRed;
                Label8->Caption = "!!! EM OPERAÇÃO !!!";
                isRed = true;
        }

        //Atualizando posicionamento do conjunto 1
        if ( (movimento.getCnjPlacas1().getPlaca1().getPosInicial() <
              movimento.getCnjPlacas1().getPlaca1().getPosFinal()) )
        {
                if ( !cnj1Concluido )
                {
                        sensorDeslocCnj1++;
                        posCorrente1->Text = FloatToStr(sensorDeslocCnj1)
                            ;
                        posCorrente1->Update();
                }
        }
        else
        {
                if ( !cnj1Concluido )
                {
                        sensorDeslocCnj1--;
                        posCorrente1->Text = FloatToStr(sensorDeslocCnj1)
                            ;
                        posCorrente1->Update();
                }
        }

        //Atualizando posicionamento do conjunto 2
        if ( (movimento.getCnjPlacas2().getPlaca1().getPosInicial() <
              movimento.getCnjPlacas2().getPlaca1().getPosFinal()) )
        {
                if ( !cnj2Concluido )
                {
                        sensorDeslocCnj2++;
                        posCorrente2->Text = FloatToStr(sensorDeslocCnj2)
                            ;
                        posCorrente2->Update();
                }
        }
        else
        {
                if ( !cnj2Concluido )
                {
                        sensorDeslocCnj2--;
                        posCorrente2->Text = FloatToStr(sensorDeslocCnj2)
                            ;
                        posCorrente2->Update();
                }
```

```
        }
        if ( sensorDeslocCnj1 == movimento.getCnjPlacas1().getPlaca1().
            getPosFinal() && !cnj1Concluido)
        {
                cnj1Concluido = true;
                ShowMessage("Movimentação conjunto 1 concluído!");
        }

        if ( sensorDeslocCnj2 == movimento.getCnjPlacas2().getPlaca1().
            getPosFinal() && !cnj2Concluido)
        {
                cnj2Concluido = true;
                ShowMessage("Movimentação conjunto 2 concluído!");
        }
        if ( cnj1Concluido && cnj2Concluido )
        {
                Label8->Font->Color = clBlue;
                Label8->Caption = "CONCLUÍDO!";
        }
}
//
```

Programa 6: Implementação dos Eventos de Formulário.

B Bibliotecas C para implementação do *Modo CDC*

B.1 Biblioteca *usb_cdc.h*

```
////////////////////////////////////////////////////////////////////////////
////                                                                    ////
////                            usb_cdc.h                               ////
////                                                                    ////
//// Library for adding a virtual COM port on your PC over USB using    ////
//// the standard Communication Device Class (CDC) specification.       ////
//// Including this file in your code will add all USB code,            ////
//// interrupts, descriptors and handlers required.  No other           ////
//// modifications need to be made.                                     ////
////                                                                    ////
//// This library creates a virtual RS232 link between the PC and       ////
//// the PIC, therefore the library provided will be familiar to        ////
//// anyone with standard UART stream I/O:                              ////
////                                                                    ////
//// usb_cdc_kbhit() - Returns TRUE if there is one or more             ////
////      character received and waiting in the receive buffer.         ////
////                                                                    ////
//// usb_cdc_getc() - Gets a character from the receive buffer.  If     ////
////      there is no data in the receive buffer it will wait until     ////
////      there is data in the receive buffer.  If you do not want      ////
////      to wait in an infinit loop, use usb_cdc_kbhit() first to      ////
////      check if there is data before calling usb_cdc_getc().         ////
////                                                                    ////
//// usb_cdc_putc(char c) - Puts a character into the transmit          ////
////      buffer.  If the transmit buffer is full it will wait until    ////
////      the transmit buffer is not full before putting the char       ////
////      into the transmit buffer.  The transmit buffer is read by     ////
////      the PC very quickly, and therefore the buffer should only     ////
////      be full for a few milli-seconds.  If you are concerned        ////
////      and don't want to be stuck in a long or infinite loop,        ////
////      use usb_cdc_putready() to see if there is space in the        ////
////      transmit buffer before putting data into the transmit         ////
////      buffer.                                                       ////
////                                                                    ////
//// usb_cdc_putready() - Returns TRUE if there is room left in the     ////
////      transmit buffer for another character.                        ////
////                                                                    ////
//// usb_cdc_connected() - Returns TRUE if we received a                ////
////      Set_Line_Coding.  On most serial terminal programs (such      ////
////      as Hyperterminal), they will send a Set_Line_Coding           ////
////      message when the program starts and it opens the virtual      ////
////      COM port.  This is a simple way to determine if the PC        ////
////      is ready to display data on a serial terminal program,        ////
////      but is not garaunteed to work all the time or on other        ////
////      terminal programs.                                            ////
////                                                                    ////
//// usb_cdc_putc_fast(char c) - Similar to usb_cdc_putc(), except      ////
////      if the transmit buffer is full it will skip the char.         ////
////                                                                    ////
//// usb_cdc_line_coding - A structure used for Set_Line_Coding and     ////
////       Get_Line_Coding.  Most of the time you can ignore this.      ////
////                                                                    ////
```

```
//// usb_cdc_break - If the PC has sent a break command, this will   ////
////       hold the break time (in milli-seconds).  If the PC sends  ////
////       a value of 0xFFFF the device is supposed to hold the      ////
////       break until it sends a value of 0                         ////
////                                                                 ////
//// usb_cdc_carrier - Where Set_Control_Line_State value is stored. ////
////       Of most relevance is the field dte_present, which is the  ////
////       DTR setting.                                              ////
////                                                                 ////
//// The following functions are also provided, and are ports of the ////
//// I/O functions in input.c.  See input.c and the CCS manual for   ////
//// documentation:                                                  ////
////   get_float_usb() - Read a float number from the user           ////
////   get_long_usb() - Read a long number from the user             ////
////   get_int_usb() - Read an integer number from the user          ////
////   get_string_usb(char *s, int max) - Read a string from the user.
   ////
////   gethex_usb() - Read a byte, in HEX, from the user             ////
////   gethex1_usb() - Read a HEX character                          ////
////                                                                 ////
//// This driver will load all the rest of the USB code, and a set   ////
//// of descriptors that will properly describe a CDC device for a   ////
//// virtual COM port (usb_desc_cdc.h)                               ////
////                                                                 ////
//// An .INF file is provided (cdc_NTXP.inf) that will load the      ////
//// standard CDC drivers for a virtual COM port in Windows          ////
//// NT/2000/XP and above.  Unfortunately we do not provide an .inf  ////
//// file for Windows 98 and ME.                                     ////
////                                                                 ////
/////////////////////////////////////////////////////////////////////////
////                                                                 ////
//// VERSION HISTORY                                                 ////
////                                                                 ////
//// October 27th, 2005: Changed the way incoming packets are        ////
////                     handled in that CDC driver will not flush   ////
////                     endpoint until user has handled all data.   This
////                     will prevent overflows as the USB will NAK  ////
////                     incoming packets until it is ready to receive
////                     more.                                       ////
////                 When using 18F4550 family, the RX buffer is     ////
////                     mapped directly to the endpoint buffer - this
////                     saves a chunk of RAM.                       ////
////                 When using the 18F4550 family, you can increase ////
////                     the TX and RX size over 64 bytes.           ////
////                 No longer send 0len packets in the TBE interrupt.
////                 Hopefully fixed bugs that caused random crashes ////
////                     if you tried sending more than 64 bytes.    ////
////                                                                 ////
//// July 6th, 2005: Global interrupts disabled when writing to TX   ////
////                      buffer.                                    ////
////                                                                 ////
//// July 1st, 2005: Initial Release.                                ////
////                                                                 ////
/////////////////////////////////////////////////////////////////////////
////          (C) Copyright 1996,2005 Custom Computer Services       ////
//// This source code may only be used by licensed users of the CCS  ////
//// C compiler.  This source code may only be distributed to other  ////
```

```
//// licensed users of the CCS C compiler.  No other use,              ////
//// reproduction or distribution is permitted without written         ////
//// permission.  Derivative programs created using this software      ////
//// in object code form are not restricted in any way.                ////
//////////////////////////////////////////////////////////////////////////


//api for the user:
#define usb_cdc_kbhit() (usb_cdc_get_buffer_status.got)
#define usb_cdc_putready() (usb_cdc_put_buffer_nextin<
    USB_CDC_DATA_IN_SIZE)
#define usb_cdc_connected() (usb_cdc_got_set_line_coding)
void usb_cdc_putc_fast(char c);
char usb_cdc_getc(void);
void usb_cdc_putc(char c);

//input.c ported to use CDC:
float get_float_usb();
signed long get_long_usb();
signed int get_int_usb();
void get_string_usb(char* s, int max);
BYTE gethex_usb();
BYTE gethex1_usb();

//functions automatically called by USB handler code
void usb_isr_tkn_cdc(void);
void usb_cdc_init(void);
void usb_isr_tok_out_cdc_control_dne(void);
void usb_isr_tok_in_cdc_data_dne(void);
void usb_isr_tok_out_cdc_data_dne(void);

void usb_cdc_flush_out_buffer(void);

//Tells the CCS PIC USB firmware to include HID handling code.
#DEFINE USB_HID_DEVICE  FALSE
#DEFINE USB_CDC_DEVICE  TRUE

#define USB_CDC_COMM_IN_ENDPOINT          1
#define USB_CDC_COMM_IN_SIZE              8
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE  USB_CDC_COMM_IN_SIZE

//pic to pc endpoint config
#define USB_CDC_DATA_IN_ENDPOINT          2
#define USB_CDC_DATA_IN_SIZE              64
#define USB_EP2_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP2_TX_SIZE  USB_CDC_DATA_IN_SIZE

//pc to pic endpoint config
#define USB_CDC_DATA_OUT_ENDPOINT          2
#define USB_CDC_DATA_OUT_SIZE              64
#define USB_EP2_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP2_RX_SIZE  USB_CDC_DATA_OUT_SIZE

//
   //////////////////////////////////////////////////////////////////////////
```

```
//
// Include the CCS USB Libraries.  See the comments at the top of these
// files for more information
//
//
    ////////////////////////////////////////////////////////////////////////

#ifndef __USB_PIC_PERIF__
#define __USB_PIC_PERIF__ 1
#endif

#if __USB_PIC_PERIF__
 #if defined(__PCM__)
  #error CDC requires bulk mode!  PIC16C7x5 does not have bulk mode
 #else
  #include <pic18_usb.h>    //Microchip 18Fxx5x hardware layer for usb.c
 #endif
#else
 #include <usbn960x.c>    //National 960x hardware layer for usb.c
#endif
#include "usb_cdc_firm.h" //USB Configuration and Device descriptors for
    this UBS device
#include <usb.c>          //handles usb setup tokens and get descriptor
    reports

struct {
        int32    dwDTERrate;    //data terminal rate, in bits per second
        int8     bCharFormat;   //num of stop bits (0=1, 1=1.5, 2=2)
        int8     bParityType;   //parity (0=none, 1=odd, 2=even, 3=mark, 4=
            space)
        int8     bDataBits;     //data bits (5,6,7,8 or 16)
} usb_cdc_line_coding;

//length of time, in ms, of break signal as we received in a SendBreak
    message.
//if ==0xFFFF, send break signal until we receive a 0x0000.
int16 usb_cdc_break;

int8 usb_cdc_encapsulated_cmd[8];

int8 usb_cdc_put_buffer[USB_CDC_DATA_IN_SIZE];
int1 usb_cdc_put_buffer_free;
#if USB_CDC_DATA_IN_SIZE>=0x100
 int16 usb_cdc_put_buffer_nextin=0;
// int16 usb_cdc_last_data_packet_size;
#else
 int8 usb_cdc_put_buffer_nextin=0;
// int8 usb_cdc_last_data_packet_size;
#endif

struct {
   int1 got;
  #if USB_CDC_DATA_OUT_SIZE>=0x100
   int16 len;
   int16 index;
  #else
   int8 len;
```

```
    int8 index;
  #endif
} usb_cdc_get_buffer_status;

int8 usb_cdc_get_buffer_status_buffer[USB_CDC_DATA_OUT_SIZE];
#if (defined(__PIC__))
 #if __PIC__
  //#locate usb_cdc_get_buffer_status_buffer=0x500+(2*
      USB_MAX_EP0_PACKET_LENGTH)+USB_CDC_COMM_IN_SIZE
  #if USB_MAX_EP0_PACKET_LENGTH==8
     #locate usb_cdc_get_buffer_status_buffer=0x500+24
  #elif USB_MAX_EP0_PACKET_LENGTH==64
     #locate usb_cdc_get_buffer_status_buffer=0x500+136
  #else
   #error CCS BUG WONT LET ME USE MATH IN LOCATE
  #endif
 #endif
#endif

int1 usb_cdc_got_set_line_coding;

struct  {
   int1 dte_present; //1=DTE present, 0=DTE not present
   int1 active;      //1=activate carrier, 0=deactivate carrier
   int reserved:6;
} usb_cdc_carrier;

enum {USB_CDC_OUT_NOTHING=0, USB_CDC_OUT_COMMAND=1,
   USB_CDC_OUT_LINECODING=2, USB_CDC_WAIT_0LEN=3} __usb_cdc_state=0;

#byte INTCON=0xFF2
#bit INT_GIE=INTCON.7

//handle OUT token done interrupt on endpoint 0 [read encapsulated cmd
    and line coding data]
void usb_isr_tok_out_cdc_control_dne(void) {
   debug_usb(debug_putc,"CDC %X ",__usb_cdc_state);

   switch (__usb_cdc_state) {
      //printf(putc_tbe,"@%X@|r|n", __usb_cdc_state);
      case USB_CDC_OUT_COMMAND:
         //usb_get_packet(0, usb_cdc_encapsulated_cmd, 8);
         memcpy(usb_cdc_encapsulated_cmd, usb_ep0_rx_buffer,8);
        #if USB_MAX_EP0_PACKET_LENGTH==8
         __usb_cdc_state=USB_CDC_WAIT_0LEN;
         usb_request_get_data();
        #else
         usb_put_0len_0();
         __usb_cdc_state=0;
        #endif
         break;

    #if USB_MAX_EP0_PACKET_LENGTH==8
      case USB_CDC_WAIT_0LEN:
         usb_put_0len_0();
         __usb_cdc_state=0;
         break;
```

```
    #endif

        case USB_CDC_OUT_LINECODING:
            //usb_get_packet(0, &usb_cdc_line_coding, 7);
            //printf(putc_tbe,"\r\n!GSLC FIN!\r\n");
            memcpy(&usb_cdc_line_coding, usb_ep0_rx_buffer,7);
            __usb_cdc_state=0;
            usb_put_0len_0();
            break;

        default:
            __usb_cdc_state=0;
            usb_init_ep0_setup();
            break;
    }
}

//handle IN token on 0 (setup packet)
void usb_isr_tkn_cdc(void) {
    //make sure the request goes to a CDC interface
    if ((usb_ep0_rx_buffer[4] == 1) || (usb_ep0_rx_buffer[4] == 0)) {
        //printf(putc_tbe,"!%X!\r\n", usb_ep0_rx_buffer[1]);
        switch(usb_ep0_rx_buffer[1]) {
            case 0x00:  //send_encapsulated_command
                __usb_cdc_state=USB_CDC_OUT_COMMAND;
                usb_request_get_data();
                break;

            case 0x01:  //get_encapsulated_command
                memcpy(usb_ep0_tx_buffer, usb_cdc_encapsulated_cmd, 8);
                usb_request_send_response(usb_ep0_rx_buffer[6]);  //send
                    wLength bytes
                break;

            case 0x20:  //set_line_coding
                debug_usb(debug_putc,"!GSLC!");
                __usb_cdc_state=USB_CDC_OUT_LINECODING;
                usb_cdc_got_set_line_coding=TRUE;
                usb_request_get_data();
                break;

            case 0x21:  //get_line_coding
                memcpy(usb_ep0_tx_buffer, &usb_cdc_line_coding, sizeof(
                    usb_cdc_line_coding));
                usb_request_send_response(sizeof(usb_cdc_line_coding)); //
                    send wLength bytes
                break;

            case 0x22:  //set_control_line_state
                usb_cdc_carrier=usb_ep0_rx_buffer[2];
                usb_put_0len_0();
                break;

            case 0x23:  //send_break
                usb_cdc_break=make16(usb_ep0_rx_buffer[2],usb_ep0_rx_buffer
                    [3]);
                usb_put_0len_0();
```

```
            break;

        default:
            usb_request_stall();
            break;
      }
   }
}

//handle OUT token done interrupt on endpoint 3 [buffer incoming received
    chars]
void usb_isr_tok_out_cdc_data_dne(void) {
   usb_cdc_get_buffer_status.got=TRUE;
   usb_cdc_get_buffer_status.index=0;
#if (defined(__PIC__))
   #if __PIC__
    usb_cdc_get_buffer_status.len=usb_rx_packet_size(
        USB_CDC_DATA_OUT_ENDPOINT);
   #else
    usb_cdc_get_buffer_status.len=usb_get_packet_buffer(
       USB_CDC_DATA_OUT_ENDPOINT,&usb_cdc_get_buffer_status_buffer[0],
           USB_CDC_DATA_OUT_SIZE);
   #endif
#else
   usb_cdc_get_buffer_status.len=usb_get_packet_buffer(
      USB_CDC_DATA_OUT_ENDPOINT,&usb_cdc_get_buffer_status_buffer[0],
          USB_CDC_DATA_OUT_SIZE);
#endif
}

//handle IN token done interrupt on endpoint 2 [transmit buffered
    characters]
void usb_isr_tok_in_cdc_data_dne(void) {
   if (usb_cdc_put_buffer_nextin) {
      usb_cdc_flush_out_buffer();
   }
   //send a 0len packet if needed
//    else if (usb_cdc_last_data_packet_size==USB_CDC_DATA_IN_SIZE) {
//       usb_cdc_last_data_packet_size=0;
//       printf(putc_tbe, "FL 0\r\n");
//       usb_put_packet(USB_CDC_DATA_IN_ENDPOINT,0,0,USB_DTS_TOGGLE);
//    }
   else {
      usb_cdc_put_buffer_free=TRUE;
      //printf(putc_tbe, "FL DONE\r\n");
   }
}

void usb_cdc_flush_out_buffer(void) {
   if (usb_cdc_put_buffer_nextin) {
      usb_cdc_put_buffer_free=FALSE;
      //usb_cdc_last_data_packet_size=usb_cdc_put_buffer_nextin;
      //printf(putc_tbe, "FL %U\r\n", usb_cdc_put_buffer_nextin);
      usb_put_packet(USB_CDC_DATA_IN_ENDPOINT,usb_cdc_put_buffer,
          usb_cdc_put_buffer_nextin,USB_DTS_TOGGLE);
      usb_cdc_put_buffer_nextin=0;
   }
```

```
}

void usb_cdc_init(void) {
    usb_cdc_line_coding.dwDTERrate=9600;
    usb_cdc_line_coding.bCharFormat=0;
    usb_cdc_line_coding.bParityType=0;
    usb_cdc_line_coding.bDataBits=8;
    (int8)usb_cdc_carrier=0;
    usb_cdc_got_set_line_coding=FALSE;
    usb_cdc_break=0;
    usb_cdc_put_buffer_nextin=0;
    usb_cdc_get_buffer_status.got=0;
    usb_cdc_put_buffer_free=TRUE;
}

////////////////// END USB CONTROL HANDLING
    ////////////////////////////////

////////////////// BEGIN USB<->RS232 CDC LIBRARY
    ////////////////////////////

char usb_cdc_getc(void) {
    char c;

    while (!usb_cdc_kbhit()) {}

    c=usb_cdc_get_buffer_status_buffer[usb_cdc_get_buffer_status.index++];
    if (usb_cdc_get_buffer_status.index >= usb_cdc_get_buffer_status.len)
        {
        usb_cdc_get_buffer_status.got=FALSE;
        usb_flush_out(USB_CDC_DATA_OUT_ENDPOINT, USB_DTS_TOGGLE);
    }

    return(c);
}

void usb_cdc_putc_fast(char c) {
    int1 old_gie;

    //disable global interrupts
    old_gie=INT_GIE;
    INT_GIE=0;

    if (usb_cdc_put_buffer_nextin >= USB_CDC_DATA_IN_SIZE) {
        usb_cdc_put_buffer_nextin=USB_CDC_DATA_IN_SIZE-1;  //we just
            overflowed the buffer!
    }
    usb_cdc_put_buffer[usb_cdc_put_buffer_nextin++]=c;

    //renable global interrupts
    INT_GIE=old_gie;

    /*
    if (usb_tbe(USB_CDC_DATA_IN_ENDPOINT)) {
        if (usb_cdc_put_buffer_nextin)
            usb_cdc_flush_out_buffer();
    }
```

```
    */
    if (usb_cdc_put_buffer_free) {
        usb_cdc_flush_out_buffer();
    }
}

void usb_cdc_putc(char c) {
    while (!usb_cdc_putready()) {
        if (usb_cdc_put_buffer_free) {
            usb_cdc_flush_out_buffer();
        }
        //delay_ms(500);
        //printf(putc_tbe,"TBE=%U CNT=%U LST=%U\r\n",usb_tbe(
            USB_CDC_DATA_IN_ENDPOINT), usb_cdc_put_buffer_nextin,
            usb_cdc_last_data_packet_size);
    }
    usb_cdc_putc_fast(c);
}

#include <ctype.h>

BYTE gethex1_usb() {
    char digit;

    digit = usb_cdc_getc();

    usb_cdc_putc(digit);

    if(digit<='9')
      return(digit-'0');
    else
      return((toupper(digit)-'A')+10);
}

BYTE gethex_usb() {
    int lo,hi;

    hi = gethex1_usb();
    lo = gethex1_usb();
    if(lo==0xdd)
      return(hi);
    else
      return( hi*16+lo );
}

void get_string_usb(char* s, int max) {
    int len;
    char c;

    --max;
    len=0;
    do {
      c=usb_cdc_getc();
      if(c==8) {  // Backspace
          if(len>0) {
            len--;
            usb_cdc_putc(c);
```

```
            usb_cdc_putc(' ');
            usb_cdc_putc(c);
          }
        } else if ((c>=' ')&&(c<='~'))
          if(len<max) {
            s[len++]=c;
            usb_cdc_putc(c);
          }
    } while(c!=13);
    s[len]=0;
}


// stdlib.h is required for the ato_ conversions
// in the following functions
#ifdef _STDLIB

signed int get_int_usb() {
  char s[5];
  signed int i;

  get_string_usb(s, 5);

  i=atoi(s);
  return(i);
}

signed long get_long_usb() {
  char s[7];
  signed long l;

  get_string_usb(s, 7);
  l=atol(s);
  return(l);
}

float get_float_usb() {
  char s[20];
  float f;

  get_string_usb(s, 20);
  f = atof(s);
  return(f);
}

#endif
```

Programa 7: Declarações e definições da biblioteca *usb_cdc.h*.

B.2 Biblioteca *usb_cdc_firm.h*

```
//
   ////////////////////////////////////////////////////////////////////////

////                          usb_desc_cdc.h
   ////
////
   ////
//// An example set of device / configuration descriptors for use with
   ////
//// CCS's CDC Virtual COM Port driver (see usb_cdc.h)
   ////
////
   ////
//// Two examples are provided:
   ////
////       ex_usb_serial.c
   ////
////       ex_usb_serial2.c
   ////
////
   ////
//
   ////////////////////////////////////////////////////////////////////////

////
   ////
//// Version History:
   ////
////
   ////
//// 10/28/05:
   ////
////    Bulk endpoint sizes updated to allow more than 255 byte
   ////
////    packets.
   ////
////    Changed device to USB 1.10
   ////
////
   ////
//
   ////////////////////////////////////////////////////////////////////////

////        (C) Copyright 1996,2005 Custom Computer Services
   ////
//// This source code may only be used by licensed users of the CCS
   ////
//// C compiler.  This source code may only be distributed to other
   ////
//// licensed users of the CCS C compiler.  No other use,
   ////
//// reproduction or distribution is permitted without written
   ////
//// permission.  Derivative programs created using this software
   ////
```

```
//// in object code form are not restricted in any way.
    ////
//
    ////////////////////////////////////////////////////////////////////////////


#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>


//////////////////////////////////////////////////////////////////////
///
///    start config descriptor
///    right now we only support one configuration descriptor.
///    the config, interface, class, and endpoint goes into this array.
///
//////////////////////////////////////////////////////////////////////

    #DEFINE USB_TOTAL_CONFIG_LEN        67   //config+interface+class+
        endpoint+endpoint (2 endpoints)

    const char USB_CONFIG_DESC[] = {
    //IN ORDER TO COMPLY WITH WINDOWS HOSTS, THE ORDER OF THIS ARRAY MUST
        BE:
        //      config(s)
        //      interface(s)
        //      class(es)
        //      endpoint(s)

    //config_descriptor for config index 1
          USB_DESC_CONFIG_LEN, //length of descriptor size           ==0
          USB_DESC_CONFIG_TYPE, //constant CONFIGURATION (CONFIGURATION 0
              x02)      ==1
          USB_TOTAL_CONFIG_LEN,0, //size of all data returned for this
              config       ==2,3
          2, //number of interfaces this device supports        ==4
          0x01, //identifier for this configuration.  (IF we had more than
               one configurations)       ==5
          0x00, //index of string descriptor for this configuration
              ==6
          0xC0, //bit 6=1 if self powered, bit 5=1 if supports remote
              wakeup (we don't), bits 0-4 unused and bit7=1          ==7
          0x32, //maximum bus power required (maximum milliamperes/2)  (0
              x32 = 100mA)  ==8

    //interface descriptor 0 (comm class interface)
          USB_DESC_INTERFACE_LEN, //length of descriptor       =9
          USB_DESC_INTERFACE_TYPE, //constant INTERFACE (INTERFACE 0x04)
                =10
          0x00, //number defining this interface (IF we had more than one
              interface)     ==11
          0x00, //alternate setting      ==12
          1, //number of endpoints    ==13
          0x02, //class code, 02 = Comm Interface Class      ==14
          0x02, //subclass code, 2 = Abstract      ==15
```

```
        0x01, //protocol code, 1 = v.25ter       ==16
        0x00, //index of string descriptor for interface       ==17

//class descriptor [functional header]
        5, //length of descriptor     ==18
        0x24, //dscriptor type (0x24 == )       ==19
        0, //sub type (0=functional header) ==20
        0x10,0x01, //       ==21,22 //cdc version

//class descriptor [acm header]
        4, //length of descriptor     ==23
        0x24, //dscriptor type (0x24 == )       ==24
        2, //sub type (2=ACM)    ==25
        2, //capabilities     ==26  //we support Set_Line_Coding,
            Set_Control_Line_State, Get_Line_Coding, and the notification
             Serial_State.

//class descriptor [union header]
        5, //length of descriptor     ==27
        0x24, //dscriptor type (0x24 == )       ==28
        6, //sub type (6=union)     ==29
        0, //master intf      ==30  //The interface number of the
            Communication or Dat a Cl ass interface, designated as the
            masteror controlling interface for the union.
        1, //save intf0       ==31  //Interface number of first slave or
            associated interface in the union. *

//class descriptor [call mgmt header]
        5, //length of descriptor     ==32
        0x24, //dscriptor type (0x24 == )       ==33
        1, //sub type (1=call mgmt)    ==34
        0, //capabilities           ==35  //device does not handle call
            management itself
        1, //data interface         ==36  //interface number of data
            class interface

//endpoint descriptor
        USB_DESC_ENDPOINT_LEN, //length of descriptor
            ==37
        USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)
                    ==38
        USB_CDC_COMM_IN_ENDPOINT | 0x80, //endpoint number and direction
        0x03, //transfer type supported (0x03 is interrupt)          ==40
        USB_CDC_COMM_IN_SIZE,0x00, //maximum packet size supported
                            ==41,42
        250,  //polling interval, in ms.  (cant be smaller than 10)
                ==43

//interface descriptor 1 (data class interface)
        USB_DESC_INTERFACE_LEN, //length of descriptor       =44
        USB_DESC_INTERFACE_TYPE, //constant INTERFACE (INTERFACE 0x04)
                  =45
        0x01, //number defining this interface (IF we had more than one
            interface)     ==46
        0x00, //alternate setting      ==47
        2, //number of endpoints    ==48
        0x0A, //class code, 0A = Data Interface Class      ==49
```

```
        0x00, //subclass code       ==50
        0x00, //protocol code       ==51
        0x00, //index of string descriptor for interface       ==52

    //endpoint descriptor
        USB_DESC_ENDPOINT_LEN, //length of descriptor
            ==60
        USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)
                ==61
        USB_CDC_DATA_OUT_ENDPOINT, //endpoint number and direction (0x02
             = EP2 OUT)          ==62
        0x02, //transfer type supported (0x02 is bulk)          ==63
//          make8(USB_CDC_DATA_OUT_SIZE,0),make8(USB_CDC_DATA_OUT_SIZE,1),
     //maximum packet size supported                          ==64, 65
        USB_CDC_DATA_OUT_SIZE & 0xFF, (USB_CDC_DATA_OUT_SIZE >> 8) & 0
            xFF, //maximum packet size supported                    ==64,
            65
        250,  //polling interval, in ms.  (cant be smaller than 10)
                ==66

    //endpoint descriptor
        USB_DESC_ENDPOINT_LEN, //length of descriptor
            ==53
        USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)
                ==54
        USB_CDC_DATA_IN_ENDPOINT | 0x80, //endpoint number and direction
             (0x82 = EP2 IN)        ==55
        0x02, //transfer type supported (0x02 is bulk)          ==56
//          make8(USB_CDC_DATA_IN_SIZE,0),make8(USB_CDC_DATA_IN_SIZE,1),
     //maximum packet size supported                          ==57, 58
        USB_CDC_DATA_IN_SIZE & 0xFF, (USB_CDC_DATA_IN_SIZE >> 8) & 0xFF,
             //maximum packet size supported                  ==64, 65
        250,  //polling interval, in ms.  (cant be smaller than 10)
                ==59
    };

    //****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
    //since we can't make pointers to constants in certain pic16s, this is
         an offset table to find
    //  a specific descriptor in the above table.

    //the maximum number of interfaces seen on any config
    //for example, if config 1 has 1 interface and config 2 has 2
        interfaces you must define this as 2
    #define USB_MAX_NUM_INTERFACES    2

    //define how many interfaces there are per config.  [0] is the first
        config, etc.
    const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={2};

    //define where to find class descriptors
    //first dimension is the config number
    //second dimension specifies which interface
    //last dimension specifies which class in this interface to get, but
        most will only have 1 class per interface
    //if a class descriptor is not valid, set the value to 0xFFFF
    const int16 USB_CLASS_DESCRIPTORS[USB_NUM_CONFIGURATIONS][
```

```
    USB_MAX_NUM_INTERFACES][4]=
{
//config 1
    //interface 0
        //class 1-4
        18,23,27,32,
    //interface 1
        //no classes for this interface
        0xFFFF,0xFFFF,0xFFFF,0xFFFF
};

#if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
    #error USB_TOTAL_CONFIG_LEN not defined correctly
#endif


////////////////////////////////////////////////////////////////
///
///   start device descriptors
///
////////////////////////////////////////////////////////////////

    const char USB_DEVICE_DESC[USB_DESC_DEVICE_LEN] ={
        //starts of with device configuration. only one possible
            USB_DESC_DEVICE_LEN, //the length of this report    ==0
            0x01, //the constant DEVICE (DEVICE 0x01)  ==1
            0x10,0x01, //usb version in bcd  ==2,3
            0x02, //class code. 0x02=Communication Device Class ==4
            0x00, //subclass code ==5
            0x00, //protocol code ==6
            USB_MAX_EP0_PACKET_LENGTH, //max packet size for endpoint 0. (
                SLOW SPEED SPECIFIES 8) ==7

            0xD8,0x04,              //vendor id (0x04D8 identifica a Microchip,
                 descobrir pq digita ao contrario...)
            0x0A,0x00,              //Product id(0x000A identifica PICs da
                familia 18F, para reconhecimento do driver doWindows)

// 0x61,0x04, //vendor id (0x04D8 is Microchip, or is it 0x0461 ??)
    ==8,9
// 0x33,0x00, //product id    ==10,11
//

            0x00,0x01, //device release number  ==12,13
            0x01, //index of string description of manufacturer. therefore
                we point to string_1 array (see below)  ==14
            0x02, //index of string descriptor of the product  ==15
            0x00, //index of string descriptor of serial number  ==16
            USB_NUM_CONFIGURATIONS  //number of possible configurations
                ==17
    };


////////////////////////////////////////////////////////////////
///
///   start string descriptors
///   String 0 is a special language string, and must be defined.  People
```

```
     in U.S.A. can leave this alone.
///
///    You must define the length else get_next_string_character() will
    not see the string
///    Current code only supports 10 strings (0 thru 9)
///
////////////////////////////////////////////////////////////////////

//the offset of the starting location of each string.  offset[0] is the
    start of string 0, offset[1] is the start of string 1, etc.
char USB_STRING_DESC_OFFSET[]={0,4,12};

char const USB_STRING_DESC[]={
   //string 0
          4, //length of string index
          USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
          0x09,0x04,    //Microsoft Defined for US-English
   //string 1
          8, //length of string index
          USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
           'L',0,
           'V',0,
           'S',0,
   //string 2 --> nombre del dispositivo
          22, //length of string index, STRING QUE APARECE QUANDO O
              DISPOSITIVO É CONECTADO AO COMPUTADOR
          USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
           'T',0,
           'U',0,
           'O',0,
           'R',0,
           'I',0,
           'A',0,
           'L',0,
    ' ',0,
           'U',0,
           'S',0,
           'B',0
};

#ENDIF
```

Programa 8: Declarações e definições da biblioteca *usb_cdc_firm.h*.

C Uso das API's *Win32* Para Comunicação Serial

C.1 Exemplo 1 - Usando *DCB* - Device Control Block

```
#include <windows.h>
int main(int argc, char *argv[])
{
  DCB dcb;
  HANDLE hCom;
  BOOL fSuccess;
  char *pcCommPort = "COM2";
  hCom = CreateFile(
       pcCommPort,
       GENERIC_READ | GENERIC_WRITE,
       0, // dispositivos comm abertos com acesso exclusivo
       NULL, // sem atributos de segurança
       OPEN_EXISTING, // deve usar OPEN_EXISTING
       0, // I/O sem ovelap
       NULL // hTemplate deve ser NULL para comm
       );

  if (hCom == INVALID_HANDLE_VALUE)
  {
    // Tata o erro
    printf ("CreateFile falhou com o erro %d.\n", GetLastError());
    return (1);
  }

  // Vamos mudar a configuração corrente e saltar a definição de do
     tamanho dos buffers de entrada e saída com SetupComm.
  fSuccess = GetCommState(hCom, &dcb);
  if (!fSuccess)
  {
    // Trata o erro
    printf ("GetCommState falhou com erro %d.\n", GetLastError());
    return (2);
  }

  // Preenche DCB: baud=57,600 bps, 8 bits de dados, sem paridade, 1 stop
       bit
  dcb.BaudRate = CBR_57600; // define o baud rate
  dcb.ByteSize = 8; // data size, xmit, and rcv
  dcb.Parity = NOPARITY; // sem paridade
  dcb.StopBits = ONESTOPBIT; // um stop bit

  fSuccess = SetCommState(hCom, &dcb);
  if (!fSuccess)
  { // Trata o erro
    printf ("SetCommState falhou com erro %d.\n", GetLastError());
    return (3);
  }
  printf ("Porta serial %s configurada com sucesso.\n", pcCommPort);
  return (0);
}
```

Programa 9: Uso de *DBC* para comunicação serial.