ESCOLA POLITÉCNICA
DE PERNAMBUCO

SISTEMA DE COMUNICAÇÃO USB COM MICROCONTROLADOR

Trabalho de Conclusão de Curso

Engenharia da Computação

Leonardo de Sá Leal Santos
Orientador: Prof. Sérgio Campello Oliveira

LEONARDO DE SÁ LEAL SANTOS

SISTEMA DE COMUNICAÇÃO USB COM MICROCONTROLADOR

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Junho de 2009.

Dedico este trabalho a minha família, em especial a meus pais que em todo tempo me apoiaram e me incentivaram para a buscas desta conquista.

Agradecimentos

Agradeço primeiramente a Deus, pois ele é o centro da minha vida e tudo que sou, isto é, minha educação, meu entendimento, meu modo de pensar, minha vida. Tudo devo a Ele. Nada faria sem o apoio do meu Deus todo poderoso.

Agradeço com grande orgulho a minha família, pois ela tem se dedicado me ajudando a conquistar essa vitoria. Em todos os momentos pude contar com o apoio, a disciplina e os conselhos que me fizeram vencer. Em especial quero agradecer a meus pais, Fátima e Leal, pois eles são a grande parte das minhas vitorias.

Queria agradecer a minha companheira fiel, Raquel, pois sempre tem me apoiado, apesar das dificuldades e sempre esteve ao meu lado para me levantar nos momentos em que não conseguia mais caminhar. Queria agradecer os momentos de compreensão e de abdicação, pois eles são expressos hoje como momentos de sabedoria. Agradeço também a sua família, que por laços de sangue é “sua”, mas na pratica é minha família a qual sou de mesmo modo grato.

Presto meus agradecimentos a meus amigos de turma, companheiros em momentos de desespero e trabalho árduo. Também a meus professores, que promoveram a orientação necessária para concluir esta vitoria hoje. Estes não só transmitiram o conhecimento, mas também lições para a vida.

Agradeço em especial a meu orientador, Sergio Campello pela paciência e companheirismo, não só nestes momentos finais, mas também em todos esses 5 anos.

Um forte abraço e um grande obrigado !!

Resumo

A cada dia, com o crescimento de tecnologia voltada a sistemas embarcados, o uso microcontroladores tem se tornado mais freqüente. Este tipo de componente apresenta inúmeras vantagens como baixo consumo de energia, programação via linguagem de alto e baixo nível, entre outras. Há sempre uma necessidade de comunicação entre sistemas embarcados e computadores, sejam para diagnostico, monitoramento ou transferência de dados armazenados. Assim é necessário sempre um modo de comunicação simples e disponível em vários tipos de equipamento. É proposto por este trabalho o desenvolvimento de um sistema de comunicação USB com microcontroladores. Esse sistema é projetado para fornecer comunicação em projetos desenvolvidos, em desenvolvimento ou em planejamento, pois pode ser adicionado como um módulo a outros sistemas. Já estão presentes no sistema todas as configurações necessárias para fornecer comunicação USB. É proposto também por este trabalho o desenvolvimento de uma biblioteca de funções para comunicação USB que proporciona facilidade de configuração, abstração do funcionamento em baixo nível e instruções de uso no desenvolvimento de um sistema personalizado ou de outras funcionalidades caso seja necessário. O desenvolvimento é feito em linguagem de programação de alto nível.

Abstract

Each day, with the embedded systems technologies growing, the use of microcontrollers has been more common. This kind of component has a lot of advantages, like small energy consumption, high or low level of programming, and others. There is always the necessity of communications between computers and embedded systems, for diagnosis, monitoring or data transfers. Thus, is always necessary a simple communication way avaliable for at a sort of equipments. This work purposes na USB communication system with microcontrollers development. This system is designed for giving communications in projects in development, developed or in planning, because it can be added as a module with another systems. There is already inside the system all the necessary configurations for giving USB communication. This work also purposes a library of functions for USB communication, providing configuration facilites, better level of low level programming abstraction, instructions of use for custom system development or another funcionalities, if necessary. All the development in this work is made using high-level programming language.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

USB - *Universal Serial Bus*

RAM – *Random Access Memory*

EEPROM - *Electrically-Erasable Programmable Read-Only Memory*

ALU - *Arithmetic Logic Unit*

FS - *Full Speed*

LS - *Low Speed*

HID - *Human Interface Device*

MSC - *Mass Storage Device*

CDC - *Communications Device Class*

SPP - *Streaming Parallel Port*

CP - *Code-Protect bit*

WRT - *Write-Protect bit*

EBTR - *External Block Table Read bit*

WDT - *Watchdog Timer*

ICSP- *In Circuit Serial Programming*

LVP - *Low-Voltage Programming*

PLL - *Phase Locked Loop*gsdygs

Capítulo 1

Introdução

Há uma tendência nos dias atuais para a utilização da comunicação USB devido a sua gama de vantagens em relação a outros tipos de comunicação, como por exemplo, possuir configuração fácil por parte do usuário, ter um baixo custo de desenvolvimento, ter disponível vasta documentação e suporte, entre outras. Neste trabalho foram observadas essas vantagens tornando a comunicação USB em microcontroladores como alvo principal. A comunicação USB proporciona uma conexão padrão para diversos tipos de dispositivos tornando muito fácil a instalação deste projeto em qualquer computador. Esse tipo de comunicação também proporciona a rápida utilização após a instalação, característica chamada de *"plug and play".*

O foco inicial era obter um melhoramento do Sistema de Detecção Óptica de Descargas Parciais em Cadeias de Isoladores em Linhas de Transmissão de Alta Tensão [1], incluindo a transmissão USB para comunicação direta com um computador. Esse melhoramento seria realizado na central de processamento desse sistema que usa o microcontrolador PIC 16F877A [2] e deveria ser adaptada para o uso do PIC 18F4550 [3] migrando todas as funções definidas anteriormente, contudo a ampliação deste trabalho para uso em diversos sistemas se tornou em foco principal, tornando-o assim mais abrangente.

Neste trabalho foi proposto esse tipo de comunicação pela razão de ampla compatibilidade com diversas plataformas e sistemas operacionais. O Windows, por exemplo, o suporta desde a versão 98. Sistemas operacionais Linux e Mac também são compatíveis.

Atualmente, é possível conectar dispositivos USB em vários outros aparelhos eletrônicos. Além de que os dispositivos USB podem ser conectados e desconectados a qualquer momento. Em um computador, por exemplo, não é necessário reiniciá-lo ou desligá-lo para conectar ou desconectar. A interface de comunicação USB tem a característica de fornecer tensão para alimentação elétrica

dos dispositivos conectados através dela. Por exemplo, um celular conectado em um computador através de conexão USB não necessita de nenhuma alimentação externa, no entanto é possível o carregamento de sua bateria. O sistema desenvolvido neste trabalho usa a alimentação que é fornecida pela conexão USB.

A manipulação dos dispositivos com comunicação USB é intuitiva, rápida e de simples configuração para o usuário, contudo o seu desenvolvimento necessita de vários módulos de configuração, esses descritos neste trabalho em detalhes. Com o uso de microcontroladores esse tipo de comunicação necessita também de descritores (trazem informações sobre o dispositivo), biblioteca de funções, entre outros. Uma nova biblioteca foi desenvolvida, também descrita neste trabalho, para tornar mais simples a implementação de futuros projetos, seja nesse sistema, seja em qualquer outro sistema que use microcontroladores da *Microchip* [4] e programação em linguagem C.

A carência de portas de comunicação paralela em novos dispositivos provocou o desincentivo ao desenvolvimento desse tipo de comunicação nos sistemas embarcados em geral. Esse é outro motivo para o desenvolvimento deste trabalho com comunicação USB para microcontroladores.

Capítulo 2

Microcontroladores

Microcontrolador é um circuito integrado composto por um microprocessador e dispositivos periféricos essenciais para o seu funcionamento como: memória de programa e de dados; e também periféricos acessórios como: interfaces de entrada e saída de dados. Os microcontroladores também são equipados com diversos circuitos eletrônicos tais como: conversor analógico digital, temporizadores, comparadores, interfaces de comunicação, geradores de pulsos, entre outros.

São muito populares devido ao seu baixo custo. Isso vem tornando os microcontroladores como soluções de viabilidade de vários projetos que tem como prioridade o baixo consumo de energia. Por serem programáveis podem ser utilizados nas mais diversas aplicações em sistemas embarcados, como celulares, eletrodomésticos, equipamentos de automação industrial, relógios, alarmes, brinquedos e outros, pois podem ser desenvolvidos para aplicações especificas. Uma grande parte de componentes eletrônicos hoje é composta por microcontroladores.

A capacidade de processamento e de armazenamento varia entre os microcontroladores definindo desta forma famílias de processadores com funções semelhantes. Existem famílias que são de linhas compactas, isto é, possuem poucas funções, ocupam menos espaço, consomem menos energia. Esses são usados para executar operações mais simples que não necessitem de muitos recursos. Outros são compostos por maior capacidade de armazenamento de dados, palavras de bits maiores, diversas funcionalidades e podem ser usados para a execução de atividades mais complexas, e em algumas ocasiões substituindo computadores.

Além de terem baixo custo como vantagem ainda consomem pouca energia, são portáteis, eliminam a necessidade de muitos componentes externos, podem ser reconfigurados com facilidade e necessitam de pouco tempo para o desenvolvimento.

2.1 Principais Componentes

A maior parte dos componentes encontrados nos microcontroladores está citada abaixo, sendo estes os principais:

2.1.1 Memória

A memória é um componente essencial em um microcontrolador e é dividida em dois tipos: memória de programa (Flash) e memória de dados (RAM – *Random Access Memory* e EEPROM - *Electrically-Erasable Programmable Read-Only Memory*).

A memória de programa é onde estão armazenadas as tarefas que o microcontrolador deve executar. Nessa os programas podem modificar configurações, manipular os dispositivos, efetuar comunicação de entrada e saída, executar instruções aritméticas, entre outros. A memória de dados é usada para armazenar resultados e dados que serão usados pelo microcontrolador. Ambas as memórias tem tamanho bem limitado se comparado com outros dispositivos.

2.1.2 ALU

A ALU (*Arithmetic Logic Unit*) é um modulo do microcontrolador que trabalha com operações lógicas de comparação como maior, menor, igual; operações booleanas como *and*, *or*, *xor*; operações aritméticas como adição, subtração, incrementação, multiplicação e divisão. É considerada a central de processamento.

2.1.3 Temporizadores e contadores

São usados para executar rotinas que precisem de noções de tempo ou contadores temporais. Podem gerar pulsos, rotinas em períodos específicos, entre outros. Seus parâmetros são alteráveis, tornando o seu uso programável para uso especifico ou geral.

2.1.4 Interfaces de entrada e saída

Este tipo de componente é responsável por prover formas de comunicação do microcontrolador com dispositivos externos. É o meio usado para a troca de dados que podem ser transmissão serial e paralela em vários protocolos como RS232 e

USB por exemplo. Podem receber informações de mouses, teclados, sensores e enviar informações para display, atuadores, entre outros.

2.1.5 Interrupções

Este é o componente que controla os pedidos de interrupção. Vários são os dispositivos que estão inclusos dentro de um microcontrolador e a sua maioria dispara pedidos de interrupção o qual pode ser usado para a execução de rotinas especificas.

Capítulo 3

Universal Serial Bus (USB)

O protocolo de comunicação USB foi desenvolvido por um conjunto de empresas que observaram a necessidade de obter uma forma de comunicação única [5]. A diversidade de protocolos e conectores dificultava a comunicação de produtos com os computadores.

O protocolo USB foi concebido com a finalidade de poder ser implementado em qualquer dispositivo, possuir configuração fácil por parte do usuário, ter um baixo custo de desenvolvimento, ser *Plug and Play* e ter disponível vasta documentação e suporte. A velocidade da transmissão de dados também foi algo em foco no desenvolvimento desse padrão.

Varias versões foram desenvolvidas com mudança nas velocidades implementadas. Sua taxa de transmissão hoje está acima de muitas formas de comunicação. A versão 1.1 tem como fator negativo a baixa velocidade na transmissão de dados (1,5 a 12 Mbps). Esse valor é considerado alto em relação as portas seriais, mas muito deficiente em relação a outros tipos de barramentos como o SCSI (80 a 160 Mbps) e o *Firewire* (400 Mbps). O padrão USB 2.0 tem a velocidade de 480 Mbps. Essa supera a velocidade das primeiras implementações do *Firewire*, tornando a USB também uma opção viável. Pode-se observar na Figura 1 um comparativo entre as formas de comunicação em relação à velocidade de transmissão.

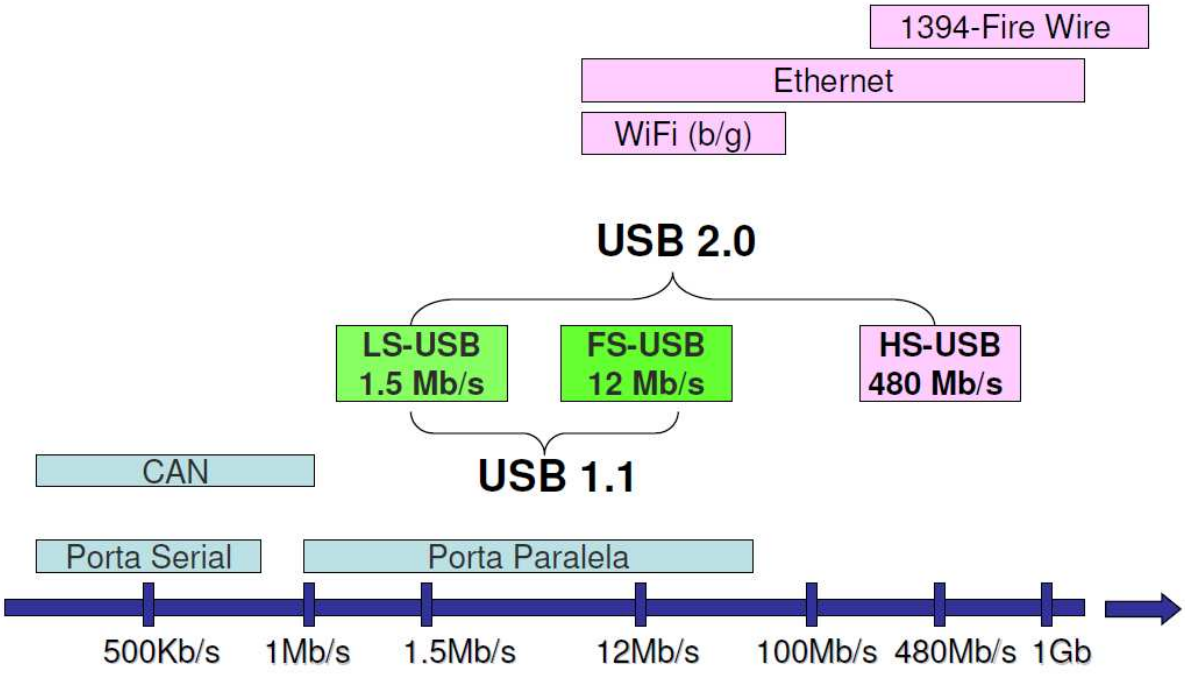

Figura 1. Comparativo entre as formas de comunicação em relação à velocidade.

3.1 Conectores

Existem vários tipos de conectores, contudo todos seguem o mesmo padrão de fabricação. A Figura 2 mostra a diversidade dos conectores e as diferenças físicas entre eles. Os diversos tipos têm a função de evitar conexões perigosas, isto é, que possam danificar algum dispositivo, como por exemplo, encadeamentos com dispositivos não permitidos ou concatenação de cabos. Os conectores mini foram criados para dispositivos menores como telefones celulares, PDAs, maquinas fotográficas, etc.

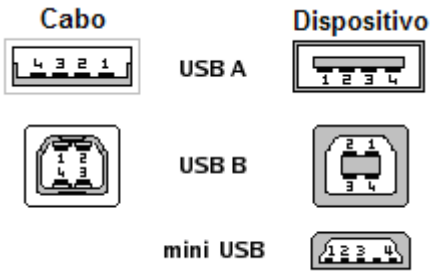

Figura 2. Conectores USB.

Cada conector possui quatro contatos funcionais. Dois para alimentação e dois para dados diferenciais e seguem as conexões indicadas na Tabela 1.

Tabela 1. Pinagem dos conectores USB.

Contato	Sinal	Cor
1	Vcc (5V)	Vermelho
2	D-	Branco
3	D+	Verde
4	Terra	Preto

3.2 Identificação do Dispositivo

As transferências de dados podem ser síncronas, em alta velocidade, ou assíncronas, com padrões de velocidade menores. A identificação desses dispositivos é feita por meio de *hardware*.

Quando um dispositivo é conectado ele é questionado quanto a sua velocidade de operação e quanto ao *driver* a ser carregado para uma configuração adequada. O dispositivo USB indica a sua velocidade ao *host* colocando a linha D+ ou a linha D- no nível lógico alto (3.3 Volts) quando se conecta ao *host*. O dispositivo FS-USB (*Full Speed*) que transmitem com taxa de 12 Mbps, usa um resistor de *pull-up* de 1.5k Ω ligado à linha D+ para se identificar como dispositivo FS-USB conforme ilustrado na Figura 3 [6]. Quando essa linha está em nível alto, o host detecta a presença do dispositivo conectado a sua porta.

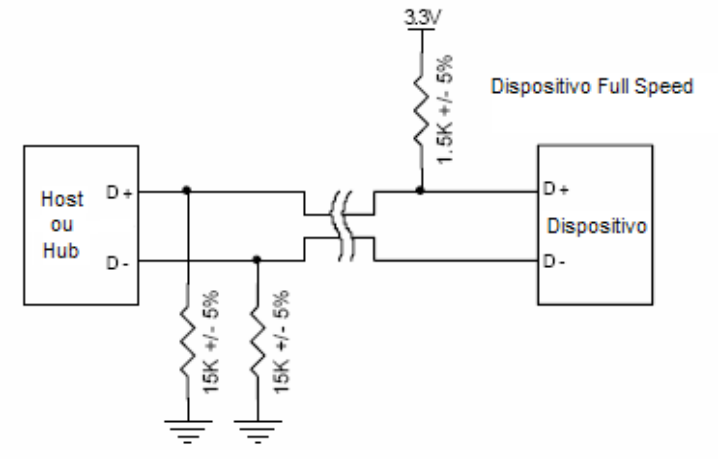

Figura 3. Configuração de hardware *Full Speed Device*

Já os dispositivos LS-USB (*Low Speed*), que transmitem com taxa de 1,5 Mbps, têm um resistor *pull-up* de 1.5k Ω ligado à linha D-, como ilustrado na Figura 4 [6]. Alguns dispositivos possuem esse resistor projetado internamente no chip.

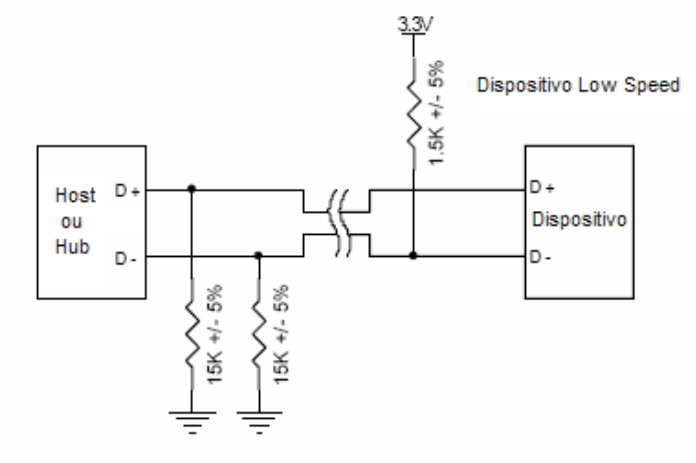

Figura 4. Configuração de hardware *Low Speed Device*

3.3 Tipos de Fluxo de Dados

Os dispositivos USB são usados para diversas aplicações. Ele pode ser usado conforme a demanda de uma aplicação e configurado para uso especifico. Existem quatro tipos de fluxos de dados que podem ser transferidos por um dispositivo USB são eles: controle, massa, interrupção, isossíncrona [6].

3.3.1 Transferência de Controle

Usada para o reconhecimento e configuração do dispositivo, acontece à troca de dados de informações sobre o dispositivo disponibilizando os descritores e parâmetros de configuração. Nesse tipo de transmissão há um robusto controle de erros, pois a troca de dados deve ser precisa.

3.3.2 Transferência de Massa

É o tipo de transferência que recebe e envia grande quantidade de dados, podendo existir simultaneamente o envio e o recebimento. Geralmente permanecem com uma conexão por muito tempo e variam a velocidade de transmissão de acordo com a carga de congestionamento do transmissor e do receptor. Geralmente usada

para a transmissão de arquivos, pois provê forte controle de erros, assim como a transferência de controle. Os dispositivos de armazenamento como *pen drivers*, cartões de memória, impressoras, scanners entre outros usam este tipo de transferência.

3.3.3 Transferência de Interrupção

Usada para enviar pequena quantidade de dados. Geralmente são caracteres ou coordenadas de dispositivos de interface humana como mouses, teclados, controladores de jogos e outros. Os dados são gerados a partir de uma detecção de mudança no estado do dispositivo que é chamada de evento. Como por exemplo, o pressionar de um botão em um controlador de jogo ou movimentar o mouse.

3.3.4 Transferência Isossíncrona

Transferência isossíncronos são trocas de dados contínuas, geralmente em tempo real. Este tipo de transferência deve ser efetuado de acordo com uma determinada taxa previamente configurada. O envio de dados é feito de acordo com essa taxa, desprezando eventuais perdas de blocos de dados neste período. Não há uma preocupação com a perda de dados, mas sim com a taxa de transmissão.

Um exemplo de uso deste tipo de transferência é o uso de dispositivos de comunicação de áudio em tempo real ou de telefonia. Devem seguir uma taxa específica; caso exista alguma perda de dados no meio da transmissão não há sentido a retransmissão desses dados, pois não é mais interessante. Não há uma rigidez quanto à correção de erros, contudo há uma preferência quanto à taxa de transmissão. Geralmente neste tipo de transmissão o dispositivo faz uma reserva de memória para o recebimento dos dados no inicio da transmissão.

Na Figura 5 podemos observar como são as características dos quatro tipos de comunicação.

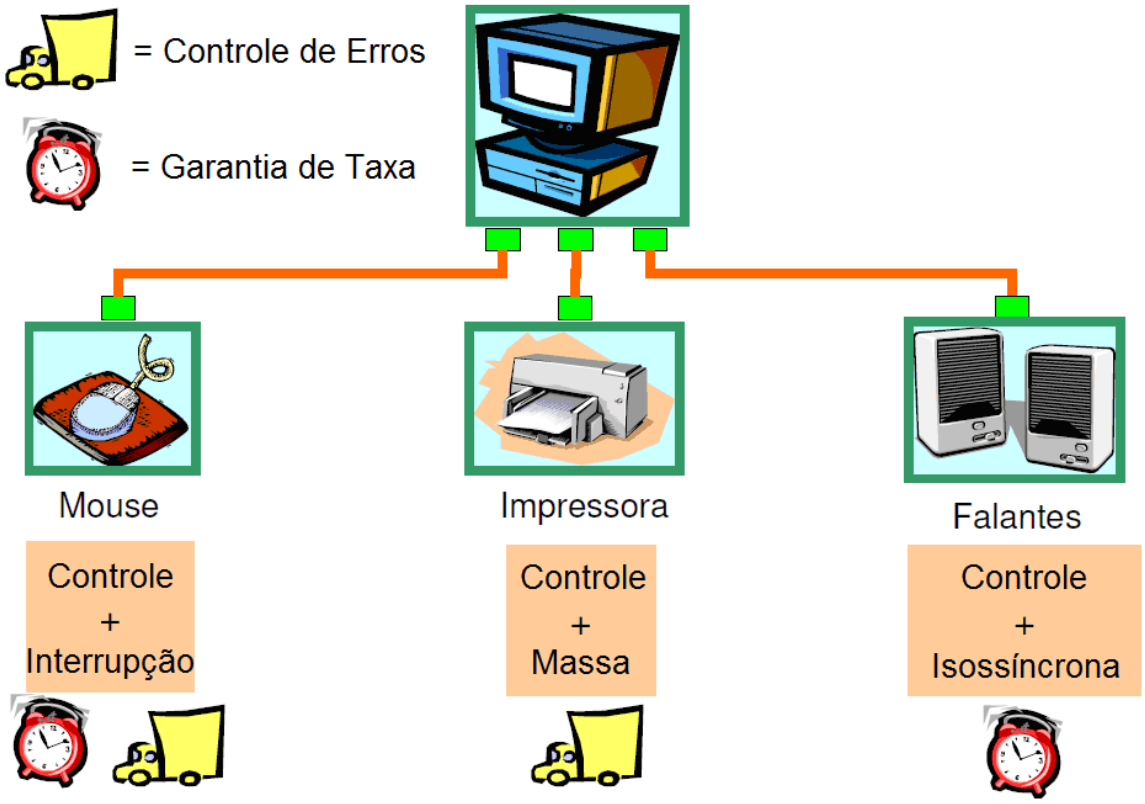

Figura 5. Tipos de fluxos de dados da comunicação USB

3.4 Descritores

Todos os dispositivos USB têm uma hierarquia de descritores que informam ao *host* o que compõe o dispositivo e suas características de funcionamento, como: número de série do produto, identificação do fabricante, tipo do dispositivo (impressora, scanner, modem, mouse, etc.), número de configurações, número de *endpoint* (local físico onde será armazenado o fluxo de dados), tipo de transferência, tipo de interface, entre outros. É possível visualizar a hierarquia dos descritores analisando a Figura 6.

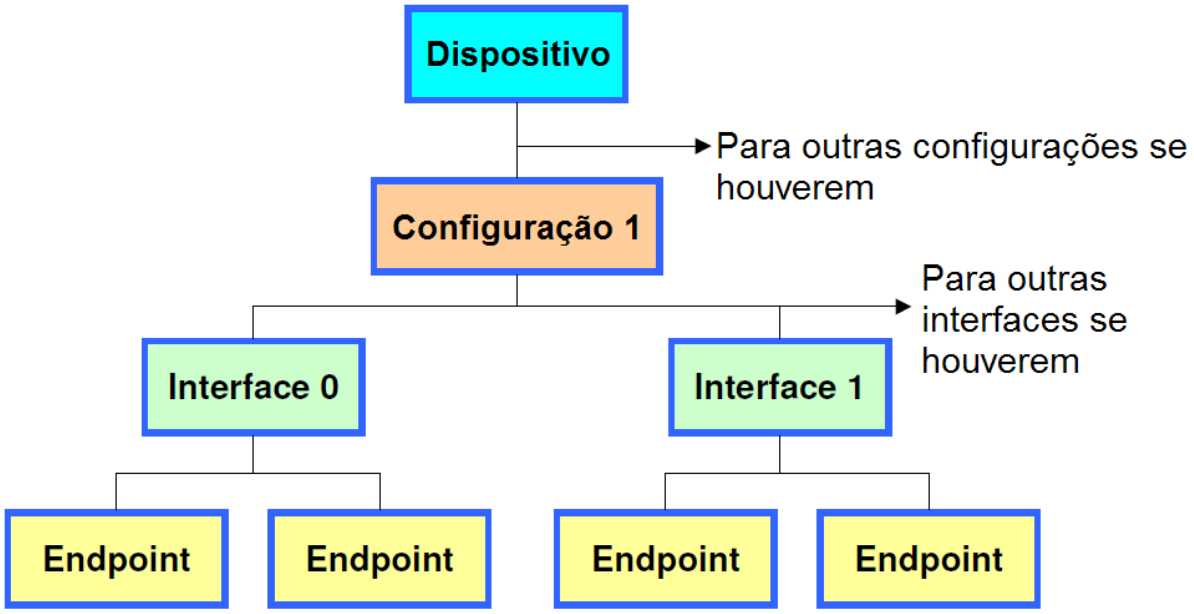

Figura 6. Hierarquia de descritores USB

O primeiro nível da hierarquia de descritores é o descritor de dispositivo. Nele é informado o tipo de dispositivo, a classe, nome do produto, fabricante, entre outros. Abaixo na hierarquia existem os descritores de configuração. Nele é informado se o dispositivo tem alimentação própria ou não, número de interfaces, entre outros. Existe, seguindo a hierarquia, o descritor de interface que informa os parâmetros que identificam a relação entre o dispositivo e o *endpoint*. O descritor de *endpoint* tem por função informar o *host* sobre uma via de comunicação específica, visto que podemos configurar o dispositivo para possuir mais de um via [6] [7].

3.5 Classes de dispositivos

3.5.1 *Human Interface Device* (HID)

Os dispositivos que fazem parte desta classe USB são dispositivos que geralmente ajudam os operadores a se comunicar de maneira fácil com os computadores. Sua instalação é simples, pois já estão previamente definidos os protocolos de comunicação para estes dispositivos. São dispositivos HID os mouses, teclados, controladores de jogos, entre outros.

Geralmente a implementação destes dispositivos não necessita de muito trabalho, pois os protocolos de comunicação já são disponíveis prontos e já estão implementados nos sistemas operacionais.

3.5.2 *Mass Storage Device* (MSC)

Os dispositivos que se enquadram nesta classe de comunicação são chamados dispositivos de armazenamento de grande quantidade de dados. Esses dispositivos são normalmente equipamentos que trabalham com transferência de um grande volume de dados, normalmente transferência de arquivos. São dispositivos que se identificam com essa classe os discos rígidos magnéticos externos, drives ópticos externos, incluindo os leitores e gravadores de CD e DVD, dispositivos portáteis de memória flash, câmeras digitais, entre outros.

3.5.3 *Communications Device Class* (CDC)

Os dispositivos do tipo de classe definido como CDC implementam um mecanismo de comunicação de propósito geral que pode ser usado para a comunicação entre a maioria dos dispositivos. Normalmente os dispositivos classificados como CDC são modems, dispositivos de rede, comunicação sem fio, telefonia. Além disso, estes dispositivos suportam implementação semelhante a uma placa de rede, fornecendo uma interface de transmissão de quadros *Ethernet* ou ATM para alguns suportes físicos.

3.5.4 Outros

Não há apenas estas classes de dispositivos. Pode-se implementar com o protocolo USB diversos tipos de dispositivos. Na Tabela 2 podemos visualizar diversas formas já determinadas.

Tabela 2. Tipos de dispositivos

Código	Classe de dispositivos
00h	Reservado
01h	Interface de áudio
02h	Comunicação e controle CDC
03h	Interface HID (*Human Interface Device*)
05h	Interface física
06h	Interface virtual
07h	Interface de impressão
08h	Interface *Mass Storage*
09h	Dispositivo Hub
0Ah	Interface CDC de dados
0Bh	Interface para cartão inteligente
0Dh	Interface de conteúdo de segurança
0Eh	Interface de vídeo
0Fh	Interface para dispositivos de cuidado com saúde pessoal
DCh	Dispositivo de diagnostico
E0h	Interface controladora de redes em fio
EFh	Híbridos
FEh	Interface de aplicações especifica
FFh	Dispositivo comercial especifico

Capítulo 4

Sistema de Comunicação USB com Microcontrolador

Neste capitulo será apresentado os conceitos e conhecimentos adquiridos para o desenvolvimento do sistema incluindo as ferramentas utilizadas, detalhes sobre o microcontrolador, uma apresentação detalhada sobre a biblioteca de funções e a implementação final e funcionamento do sistema.

4.1 Ferramentas usadas

4.1.1 Linguagem C para microcontroladores

A linguagem C é uma linguagem de alto nível e estruturada. Sua sintaxe é simples e portátil, isto é, pode ser usado o mesmo programa em várias plataformas. C é muito utilizada para a programação de microcontroladores e também tem o poder de interagir com a plataforma em baixo nível podendo incluir códigos em *assembly* em sua implementação.

A portabilidade é uma vantagem para o desenvolvimento de programas em C, os quais uma vez construídos podem ser usados em várias plataformas e dificilmente haverá modificações. Como exemplo, pode-se citar a escrita de algumas rotinas para uso no microcontrolador PIC 16F877A [2], o qual foi usado em testes inicialmente para o aprendizado, e posteriormente migrando esses programas para uso no microcontrolador PIC 18F4550 [3]. A migração destas rotinas para diferentes plataformas necessitou de poucas modificações no programa, as modificações feitas são relacionadas a funcionalidades não suportadas entre uma família e outra, ou mudanças na especificação de cada um dos microcontroladores, como o cálculo de tempo dos temporizadores.

As funções desenvolvidas em C são de fácil implementação, pois são usadas, geralmente, um conjunto de rotinas simples para a execução de rotinas complexas.

Também é possível verificar a vasta quantidade de documentação a respeito da programação nesta linguagem. Com C é possível o desenvolvimento de biblioteca de propósito específico, a qual foi desenvolvida neste projeto e será descrita em seções posteriores.

A exploração de rotinas de baixo nível é possível em C, pois existe suporte a programação em *assembly* dentro de seus programas. Deste modo pode-se desfrutar de todas as vantagens da implementação em alto nível e também fazer uso das vantagens da implementação em baixo nível explorando rotinas que podem não ser otimizadas através da programação em alto nível.

No caso de projetos com microcontroladores é disponibilizado geralmente rotinas (bibliotecas de funções) em C pelos seus fabricantes, pois é a linguagem geralmente utilizada para programação em microcontroladores pelas diversas vantagens informadas até este momento. Também é possível a programação diretamente em *assembly*, contudo é uma abordagem que exige muito mais experiência e trabalho.

O CCS C *Compiler* é um compilador de linguagem C usado neste projeto para o desenvolvimento das rotinas para os microcontroladores [8]. O CCS provê completa estrutura lógica para a programação e tem baixo custo de implementação. O CCS é composto por operadores que foram empacotados em bibliotecas que são específicos para os microcontroladores PICs, bem como acesso aos recursos de hardware com C.

O compilador é usado para gerar o código fonte em C e *assembly* em linguagem de máquina. Possui interface para facilitar a implementação do sistema assim como fornece exemplos, *drivers* e funções em forma de bibliotecas. O CCS foi concebido para dar suporte a todas as famílias dos microcontroladores PICs da Microchip. Na Figura 7 podemos verificar as distribuições e as respectivas famílias.

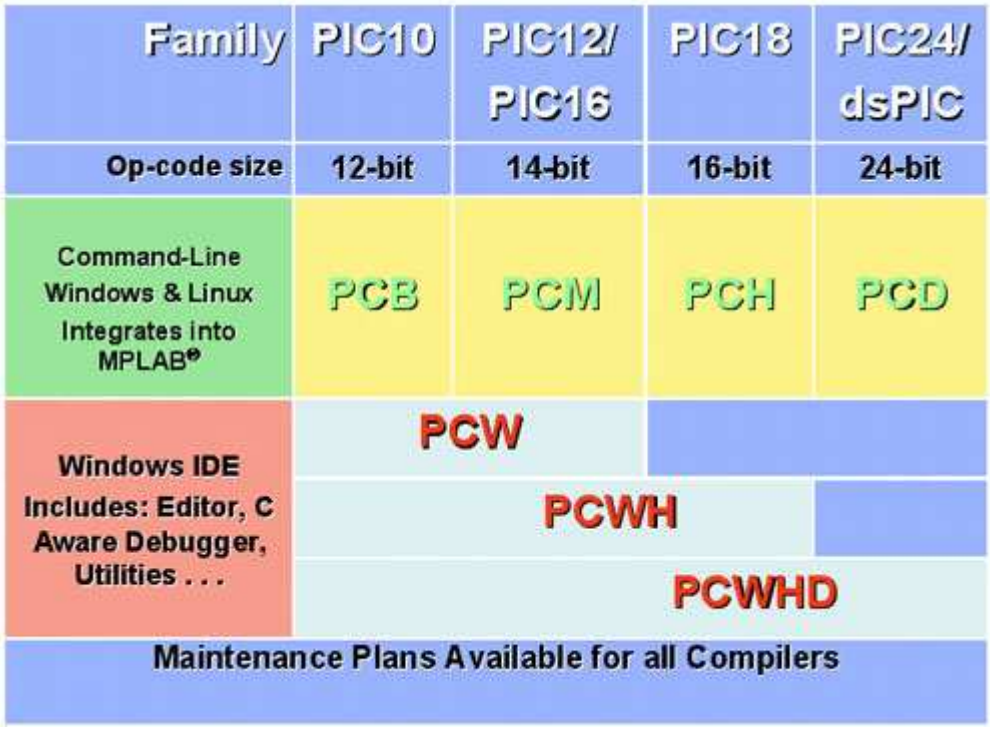

Figura 7. Distribuições e respectivas famílias do compilador CCS

No desenvolvimento deste trabalho foi usado o compilador com suporte ao PIC18, já que este projeto está baseado no microcontrolador PIC18F4550 [3] como veremos nas próximas seções. Também foi necessário o suporte a comunicação USB que é essencial para este desenvolvimento.

4.1.2 Ferramenta Computacional de Simulação

O Proteus é uma ferramenta útil para a simulação e a construção de circuitos elétricos. Nele estão disponíveis vários componentes, incluindo microcontroladores, com a possibilidade de observar o funcionamento de forma virtual. Pode ser utilizada tanto para circuitos analógicos quanto para circuitos digitais ou híbridos.

No Proteus ainda são disponibilizadas ferramentas de laboratório como osciloscópios, multímetros, geradores de sinais entre outros, permitindo o aprendizado da operação dos principais instrumentos essenciais em uma bancada de eletrônica.

O uso do Proteus foi essencial neste trabalho, para a execução de testes de funcionamento e aprendizado, pois foram usados vários exemplos até a absorção e

adaptação com as plataformas utilizadas. O uso do Proteus foi necessário para a verificação da confecção do circuito e corrigir erros de programação.

4.1.3 Envio e Recepção de Dados no Computador

O programa de recepção e envio de dados usado no computador é o programa *Advanced Serial Port Terminal*. Ele é responsável por receber e enviar dados do computador para um dispositivo usando uma porta serial. O *Advanced Serial Port Terminal* é uma ferramenta para ser usada por desenvolvedores, programadores e usuários e é possível se conectar a qualquer dispositivo para verificar, solucionar problemas, depurar ou usar o sistema [9]. O *Advanced Serial Port Terminal* é muito mais funcional do que o terminal padrão do Windows, o *Hyper Terminal*, uma vez que é capaz de enviar e receber vários tipos de dados (string ASCII, binário, octal e hexadecimal) ao longo de fluxo de comunicação.

4.2 Microcontrolador PIC 18F4550

O microcontrolador explorado neste projeto é o PIC18F4550 [2] que possui muitas funcionalidades, isto é, ele é considerado um microcontrolador completo. As características desse microcontrolador são:

- Memória de programação e de dados que permitem apagamento e re-escrita de valores milhares de vezes;
- É auto programável, isto é, através de um controle interno por software, o microcontrolador pode escrever na sua própria memória de programa. Usando uma rotina de *bootloader*, alocada no bloco *Boot* no topo da memória de programa, é possível atualizar a aplicação em campo, sem a utilização de um gravador.
- 1 Módulo *Universal Serial Bus* (USB). O PIC18F4550 possui um USB SIE (*Serial Interface Engine*) compatível com *full-speed* e *low-speed* USB o que possibilita a rápida comunicação entre um *host* USB e o microcontrolador. Além disso, o módulo USB possui 16 *endpoints* bidirecionais e suporta 4 tipos de transferências: controle, interrupção, isossíncrona e *Massa*.

- 13 Conversores analógico-digitais de 10 bits;
- Memória de programa *Flash* com 32 *Kbytes*;
- Memória de dados EEPROM 256 *Bytes*;
- Memória RAM 2 *Kbytes*;
- Freqüência de operação até 48 MHz;
- Portas bidirecionais de entrada e saída: A, B, C, D e E;
- 4 *Timers*;
- 1 Módulo *Capture*/*Compare*/PWM;
- 2 Comparadores;
- 1 *Streaming Parallel Port* (SPP).

4.3 Biblioteca de funções

A biblioteca de funções é um conjunto de funções extras que tornam transparente ao programador a manipulação do dispositivo USB. Nele existem funções de inicialização, transmissão e recepção de dados usando o protocolo de comunicação USB. Esta biblioteca foi desenvolvida para uso em sistemas que usem o microcontrolador de modelo PIC 18F4550 [3] e o compilador CCS [8].

A transmissão de dados USB neste projeto é realizada por um microcontrolador que foi programado para se comportar como um dispositivo do tipo CDC. Em um computador ele é reconhecido como uma transmissão através de uma porta serial de comunicação (COM). Todas as funções referenciadas ou desenvolvidas nesta biblioteca referem-se a esse tipo de transmissão.

Existem funções nativas fornecidas com a instalação do compilador, e estas são documentadas nesta biblioteca de forma que esteja disponível ao programador uma documentação básica para efetuar transmissão via USB. Essas funções nativas documentadas nesta biblioteca podem ser necessárias para o desenvolvimento de sistemas que usem transmissão de dados.

As funções nativas são:

- usb_enumerated(): Verifica se o dispositivo está pronto para a comunicação;
- usb_detach(): Desconecta o dispositivo. Deve ser usada antes de sua remoção física do computador;
- usb_attach(): Re-conecta o dispositivo, deve ser usada para re-conectá-lo quando o dispositivo foi desconectado, mas ainda não removido literalmente;
- usb_cdc_putc(char c): Envia um *caracter* via USB;
- usb_cdc_kbhit(): Verifica se existe algum dado no buffer de recepção;
- usb_cdc_getc(): Recebe um *caracter.* Deve-se usar o usb_cdc_kbhit() descrito anteriormente para verificar se existem dados;
- get_float_usb(): Recebe um numero ponto flutuante;
- get_long_usb(): Recebe um numero inteiro longo;
- get_int_usb(): Recebe um inteiro;
- get_string_usb(char *s, int max): Recebe uma string;
- gethex_usb(): Recebe um hexadecimal.

Maiores informações sobre estas funções nativas podem ser encontradas na documentação do compilador *CCS compiler* na pasta de instalação \...\PICC\Drivers nos arquivos USB.c, pic18_usb.h e usb_cdc.h.

As funções extras necessárias que foram implementadas e estão incluídas nesta biblioteca com a intenção de completar as funções nativas existentes são:

- usb_cdc_putString(char *p): Envia uma string;
- usb_cdc_putEEPROM (int posicaoI, int tamanho): Envia um bloco da memória de dados;
- usb_cdc_putFLASH (char posicaoI, char tamanho): Envia um bloco da memória de programa;
- usb_cdc_conectar(): Inicializa e configura o dispositivo USB.

Também está presente nesta biblioteca instruções a respeito do desenvolvimento de sistemas. As instruções são duas: a inclusão de um descritor CDC diferente e a criação de um novo arquivo descritor USB. O nome da biblioteca foi determinado como usb_biblioteca.h.

Existem dois arquivos que envolvem os descritores USB: usb_desc_cdc.h e usb_cdc.h, ambos se encontram na pasta "\...\PICC\Drivers". O primeiro é a implementação dos descritores USB. A criação de um novo arquivo descritor USB é necessária caso se deseje efetuar alguma modificação no descritor original para a criação de um descritor personalizado ou mais especifico. Devem ser feitas as modificações e salvá-las na mesma pasta onde se encontra o projeto mudando o nome do arquivo. O segundo arquivo é o usb_cdc.h. Esse arquivo indica qual o arquivo que implementa o descritor usado no sistema. Caso seja criado um novo arquivo de descritor deve ser modificada a referência dentro deste arquivo.

4.4 Comunicação USB

O desenvolvimento do sistema tem como finalidade efetuar a transferência de dados, da memória do microcontrolador ou gerados pelo programa, para um computador e transferir dados do computador para uso nas rotinas do microcontrolador ou para gravação na sua memória. Essa comunicação tem como objetivo prover um mecanismo de transferência de dados que tem a potencialidade de ser aplicado em outros sistemas.

4.4.1 Descritores USB

Inicialmente, neste trabalho, foram implementados os descritores USB. Foi necessário a modificação dos descritores originais, sendo criado um novo arquivo descritor chamado usb_desc_TCC.h. Na cadeia de hierarquia de descritores, o descritor no topo é o descritor de dispositivo. Na Tabela 3 podem-se visualizar os parâmetros aplicados nesse descritor de dispositivo e a uma explicação sobre esses parâmetros. Os valores usados foram inicialmente retirados do arquivo original usb_desc_cdc.h. Alguns dos valores atribuídos são valores já configurados pelos arquivos de especificação USB.

Tabela 3. Descritor de dispositivo

Nº	Campo	TAM.	Valor	Descrição
0	*bLength*	1	USB_DESC_DEVICE_LEN	Tamanho em bytes do descritor
1	*bDescriptorType*	1	01h	Tipo de descritor
2	*bcdUSB*	2	0110h	Versão USB utilizada
4	*bDeviceClass*	1	02h	Classe do dispositivo
5	*bDeviceSubClass*	1	00h	Sub Classe do dispositivo
6	*bDeviceProtocol*	1	00h	Protocolo utilizado
7	*bMaxPacketSize0*	1	USB_MAX_EP0_PACKET_LENGTH	Tamanho max. do pacote
8	*idVendor*	2	0461h	Id do fabricante
10	*idProduct*	2	0033h	Id do produto
12	*bcdDevice*	2	0100h	Versão do produto
14	*iManufacturer*	1	01h	Informações do produto
15	*iProduct*	1	02h	Informações do produto
16	*iSerialNumber*	1	00h	Nº de Serie do produto
17	*bNumConfigurations*	1	USB_NUM_CONFIGURATIONS	Nº de configurações disponíveis

Cada dispositivo USB possui apenas um descritor de dispositivo que contém informações gerais. Estes parâmetros são os primeiros a serem coletados quando o dispositivo USB é conectado [6]. A definição da classe e subclasse da interface segue os valores da Tabela 2 que definem o tipo de dispositivo USB.

O parâmetro neste descritor que pode sofrer alterações quando implementado em outros sistemas, é o parâmetro *bcdUSB*. Esse campo configura a versão USB utilizada. Neste caso está configurado como versão 1.1, no entanto pode também assumir o valor outras versões como a versão 2.0 (0200h).

Foi também implementado o descritor de configuração. No caso deste sistema só foi disponibilizado uma configuração, portanto outras configurações podem ser definidas, sendo utilizada uma por vez. Geralmente existe apenas uma configuração por dispositivo USB. Podem-se observar os valores atribuídos ao descritor de configuração na Tabela 4. O *Remote Wakeup* é uma função que desativa o dispositivo USB enquanto não é utilizado. Neste projeto não utilizamos esta funcionalidade e conseqüentemente foi desativada.

Tabela 4. Descritor de configuração

Nº	Campo	Tam.	Valor	Descrição
0	*bLength*	1	USB_DESC_CONFIG_LEN	Tamanho em bytes do descritor de configuração
1	*bDescriptorType*	1	USB_DESC_CONFIG_TYPE	Tipo de configuração
2	*wTotalLength*	2	USB_TOTAL_CONFIG_LEN	Tamanho de todas as configurações
4	*bNumInterfaces*	1	02h	Nº de interfaces
5	*bConfigurationValue*	1	01h	Id da configuração
6	*iConfiguration*	1	01h	Índice do descritor de string da configuração
7	*bmAttributes*	1	C0h	Definição de vários atributos
	4..0: Reserved		...00000	
	5: Remote Wakeup		..0.....	Suporte a *Remote Wakeup*
	6: Self Powered		.1.....	Alimentação do circuito
	7: Reserved		1.....	
8	*bMaxPower*	1	32h	Maxima corrente requerida (maximum milliamperes/2)

No descritor de configuração devem ser observados dois parâmetros para o desenvolvimento de outros projetos: o tipo de alimentação do circuito e a corrente máxima requerida, respectivamente os campos *Self Power* e *bMaxPower* [6].

No descritor de interface é atribuída a classe da interface de transmissão. Na configuração de dispositivos USB é possível atribuir várias interfaces de comunicação de classes diferentes. A definição da classe da interface segue os valores da Tabela 2. Neste sistema foi implementado duas interfaces: uma de controle e outra de transmissão de dados do tipo CDC. Podem-se analisar os atributos destas interfaces na Tabela 5 e na Tabela 6.

Tabela 5. Descritor de interface de controle

Nº	Campo	TAM.	Valor	Descrição
0	*bLength*	1	USB_DESC_INTERFACE_LEN	Tamanho do descritor
1	*bDescriptorType*	1	USB_DESC_INTERFACE_TYPE	Tipo de descritor
2	*bInterfaceNumber*	1	02h	Id da interface
3	*bAlternateSetting*	1	00h	Configuração alternativa
4	*bNumEndpoints*	1	01h	Numero de *endpoits*
5	*bInterfaceClass*	1	02h	Classe da interface
6	*bInterfaceSubClass*	1	02h	Sub classe
7	*bInterfaceProtocol*	1	01h	Protocolo utilizado
8	*iInterface*	1	00h	Índice do descritor de string da interface

Tabela 6. Descritor da interface de transmissão de dados do tipo CDC

Nº	Campo	TAM.	Valor	Descrição
0	*bLength*	1	USB_DESC_INTERFACE_LEN	Tamanho do descritor
1	*bDescriptorType*	1	USB_DESC_INTERFACE_TYPE	Tipo de descritor
2	*bInterfaceNumber*	1	01h	Id da interface
3	*bAlternateSetting*	1	00h	Configuração alternativa
4	*bNumEndpoints*	1	02h	Numero de *endpoits*
5	*bInterfaceClass*	1	0Ah	Classe da interface
6	*bInterfaceSubClass*	1	00h	Sub classe
7	*bInterfaceProtocol*	1	00h	Protocolo utilizado
8	*iInterface*	1	00h	Índice do descritor de string

Os descritores de *endpoints* definem o tipo de transferência de dados suportada pelo *endpoint* (controle, *massa*, interrupção e isossíncrona), a direção dos dados, o tamanho máximo do pacote de dados. Há neste trabalho três *endpoints* como se pode observar nos parâmetros dos descritores de interface. Um *endpoint* é usado para a transferência dos dados de controle, outras duas são usadas para o envio e recebimento dos dados gerados pelo sistema. Nas Tabela 7, Tabela 8 e Tabela 9 respectivamente estão exibidos os valores utilizados na implementação.

Tabela 7. Descritor de *endpoint* 0

Nº	Campo	Tam.	Valor	Descrição
0	*bLength*	1	USB_DESC_ENDPOINT_LEN	Tamanho da descrição
1	*bDescriptorType*	1	USB_DESC_ENDPOINT_TYPE	Tipo de descritor
2	*bEndpointAddress*	1	USB_CDC_COMM_IN_ENDPOINT	Endereço
3	*bmAttributes*	1	03h	Tipo de transferência
	1..0: Transfer Type		11	(Interrupção)
	7..2: Reserved		000000..	
4	*wMaxPacketSize*	2	USB_CDC_COMM_IN_SIZE	Tamanho do pacote
6	*bInterval*	1	250	Intervalo temporal (ms)

Tabela 8. Descritor de *endpoint* 1

Nº	Campo	TAM.	Valor	Descrição
0	*bLength*	1	USB_DESC_ENDPOINT_LEN	Tamanho da descrição
1	*bDescriptorType*	1	USB_DESC_ENDPOINT_TYPE	Tipo de descritor
2	*bEndpointAddress*	1	USB_CDC_DATA_OUT_ENDPOINT	Endereço
3	*bmAttributes*	1	02h	Tipo de transferência
	1..0: Transfer Type		10	(Interrupção)
	7..2: Reserved		000000..	
4	*wMaxPacketSize*	2	USB_CDC_DATA_OUT_SIZE	Tamanho do pacote
6	*bInterval*	1	250	Intervalo temporal (ms)

Tabela 9. Descritor de *endpoint* 2

Nº	Campo	TAM.	Valor	Descrição
0	bLength	1	USB_DESC_ENDPOINT_LEN	Tamanho da descrição
1	bDescriptorType	1	USB_DESC_ENDPOINT_TYPE	Tipo de descritor
2	bEndpointAddress	1	USB_CDC_DATA_IN_ENDPOINT	Endereço
3	bmAttributes	1	02h	Tipo de transferência
	1..0: Transfer Type		10	(Interrupção)
	7..2: Reserved		000000..	
4	wMaxPacketSize	2	USB_CDC_DATA_IN_SIZE	Tamanho do pacote
6	bInterval	1	250	Intervalo temporal (ms)

4.4.2 Bits de configuração

Os bits de configuração dão as diretrizes da configuração das funções do microcontrolador e se encontram sempre no inicio do código fonte em C. Esses bits são na verdade valores binários que são atribuídos a posições específicas na memória do microcontrolador. Esses dados são atribuídos no momento da compilação e gravação para configurar determinadas opções no microcontrolador.

Algumas dessas configurações não são removidas com facilidade. Um exemplo pode ser a proteção do dispositivo impedindo a leitura do programa através de um gravador externo, para evitar a sua duplicação ou engenharia reversa. Essa funcionalidade é configurada por bits de configuração. Em alguns casos a atribuição destes bits é irreversível em outros só podem ser desprogramados apagando todo o conteúdo da memória.

O desenvolvimento do sistema desenvolvido neste trabalho contém vários bits de configuração. Será abordado neste tópico os bits usados e o significado de cada um deles, mostrando as principais funções possíveis em outros projetos e as aplicadas neste.

Os bits responsáveis pela configuração de proteção do programa são: PROTECT, NOPROTECT. As rotinas usadas neste trabalho usam o bit NOPROTECT, pois o foco é disponibilizar as rotinas e torná-las mais adaptável possível a outras aplicações. Existem bits que envolvem outros tipos de proteção ou restrição envolvendo o programa contido no microcontrolador como, por exemplo, a proteção de memória do programa contra a auto modificação ou até proteção de alguns blocos de memória.

Alguns bits são o *Code-Protect bit* (CPn), o *Write-Protect bit* (WRTn) e o *External Block Table Read bit* (EBTRn). O CPn é responsável pela leitura ou escrita na memória por dispositivos externos. O WRTn é responsável pela permissão de escrita em blocos de memória do microcontrolador, seja o pedido de manipulação proveniente de dispositivo externo ou do próprio microcontrolador, e o EBTRn é responsável pela leitura de blocos de memória por dispositivos externos. Não é utilizado neste trabalho nenhuma destas funções de proteção, contudo em trabalhos futuros pode existir esta necessidade.

O *Brown-out Reset* monitora a alimentação do microcontrolador comparando-a com um valor de tensão de referencia previamente configurada. Caso a tensão de alimentação seja menor do que a de referencia o microcontrolador re-inicializa o sistema a fim de evitar mau funcionamento. Os valores de referência disponíveis estão entre os bits de habilitação do *Brown-out* via software (BROWNOUT_SW), o de desativação (NOBROWNOUT), o de ativação (BROWNOUT) e os de valores de tensão de referência de 2.0V (BORV20), 2.8V (BORV28), 4.3V (BORV43) e 4.6V (BORV46). A alimentação padrão para os microcontroladores é de 5V. O *Brown-out Reset* é configurado neste projeto como desativado. A alimentação do circuito é segura, a respeito da variação, pois o baixo consumo do projeto não ocasionará perdas prejudiciais à fonte de energia, a qual será especificada nos próximos tópicos. É possível nos trabalhos futuros a possibilidade do seu uso.

O *Power up timer* é uma funcionalidade que torna o dispositivo com uma maior margem de segurança a respeito do seu correto funcionamento. Após um estado de re-inicialização, seja após ser ligado, seja após algum evento de *Brown-out Reset*, é causado um atraso no seu efetivo funcionamento. Isso se destina a aguardar que a tensão de alimentação atinja um nível aceitável e estável e que a geração de *clock* se torne bem definida. Os bits para a ativação e desativação são respectivamente PUT, NOPUT. No projeto é aconselhável a ativação desta função, já que ela garante que não existirá um mau funcionamento no início da operação do dispositivo. Por existir a transmissão de dados em alta velocidade, foi apropriado o uso desta função para evitar a ocorrência de erros e problemas inesperados.

Nesse microcontrolador há outra funcionalidade que também é configurada por bits de configuração. Ela é chamada de Cão de Guarda (*Watchdog Timer* -

WDT). Ela funciona garantindo o funcionamento normal do sistema contra possíveis acontecimentos inesperados pela execução do programa como, por exemplo, travamentos. É configurada uma constante de tempo em um contador o qual produz uma re-inicialização do programa (*reset*) caso não seja zerado. Este contador, em operação normal, deve ser zerado pelo programa antes de seu estouro. Caso seja zerado indica que houve um mau funcionamento e provoca a re-inicialização. O nome dado a esse contador é Contador de *Watchdog*. Esse contador usa um circuito contador independente do contador do núcleo do microcontrolador. Os valores possíveis para a configuração são os bits: WDT1, WDT2, WDT4, WDT8, WDT16, WDT32, WDT64, WDT128, WDT256, WDT512, WDT1024, WDT2048, WDT4096, WDT8192, WDT16384, WDT32768. Estes bits representam valores de escalas de tempo entre 4ms e 131.072 s (2.18 minutos). Os bits NOWDT e WDT desativam e ativam esta função respectivamente. O sistema implementado promove a transferência de dados não críticos e a inclusão de um botão de re-inicialização externo, portanto não é necessária a ativação desta função. É definido no programa o bit de configuração NOWDT configurando a desativação desta função.

A re-inicialização do dispositivo é possível através de diversas funcionalidades. Vários bits configuram possíveis ocorrências, contudo existe a possibilidade de provocar no microcontrolador uma re-inicialização via meios externos. O pino MCLR é o pino específico para o uso desta função e pode ser configurado como provocador de re-inicialização. A re-inicialização ocorre quando o pino MCLR é alimentado com nível lógico zero. Os bits que configuram esta função são MCLR (para habilitar) e NOMCLR (para desabilitar). Foi definido um botão externo para provocar a re-inicialização do sistema, a fim de corrigir qualquer mau funcionamento que venha a se desenvolver durante sua operação. Está característica de pino de re-inicialização foi definida com o bit MCLR no código do programa definindo como ativada a função.

Existem os bits que permite a depuração do programa no circuito. A configuração desta função é atribuída aos bits NODEBUG (não permite a depuração) e DEBUG (permite a depuração). Neste sistema não foi permitida a depuração, já que não é necessário o uso deste tipo de funcionalidade.

A gravação do programa no microcontrolador é feita por um gravador específico fornecido pelos fabricantes. No caso dos microcontroladores da família PIC da microchip é fornecido ao pino MCLR uma tensão de 13V indicando que o microcontrolador está em modo de gravação. Nem todos os sistemas que fazem gravação no próprio circuito podem suportar a recepção dessa tensão. Para resolver este problema existe a função *Low-Voltage ICSP Programming* (LVP) que permite a gravação no microcontrolador com baixa tensão. Esta função é configurada pelos bits de configuração NOLVP e LVP. Na ativação a indicação que o microcontrolador está em modo de gravação é a atribuição de valor lógico alto no pino PGM no microcontrolador. Note que como o PGM é geralmente compartilhado com uma porta, está fica inutilizada se a LVP estiver habilitada. O ICSP significa "In Circuit Serial Programming", ou seja, programação serial no circuito. Neste sistema é usado o bit de configuração NOLVP pois mesmo que a gravação do PIC seja no próprio circuito do sistema, há um isolamento do circuito de gravação e de comunicação USB.

O bit de configuração VREGEN ativa o uso do Vusb. O Vusb é uma entrada no microcontrolador que funciona como alimentação dos resistores *pull-up* para a detecção da velocidade da transmissão USB, isto é, detecta se o dispositivo opera em *Full Speed Device* ou em *Low Speed Device*. A não ativação desta função provoca uma necessidade de alimentação externa deste pino por uma tensão de 3,3V. Ele funciona também como regulador interno de tensão para o dispositivo USB. O bit que desabilita esta funcionalidade é o NOVREGEN.

4.4.3 Configuração de *clock*

O microcontrolador usado pode assumir três fontes de geração de *clock*: oscilador primário, oscilador secundário e oscilador interno. Também pode assumir diversas configurações diferentes com cada tipo de oscilador ou a combinação entre eles [3]. Estes tipos de fontes diferentes permitem a utilização de *clock* de funcionamento diferente para o núcleo do microcontrolador e o dispositivo de comunicação USB. No projeto foi utilizada uma fonte e o oscilador primário como gerador de *clock* para todo o circuito.

O PLL (*Phase Locked Loop* - Laço Travado em Fase) é um módulo do microcontrolador que provoca pré e pós escalamento dos valores do *clock* do núcleo

do microcontrolador e do circuito USB. Também existem os bits de configuração USBDIV e NOUSBDIV que são projetados para configurar qual o *clock* do dispositivo USB integrado com o microcontrolador. Ambos os bits são usados para definir como será feito o cálculo da frequência de operação no microcontrolador, tornando adaptável a cada aplicação. Os bits configuram os blocos internos do microcontrolador conforme a Figura 8.

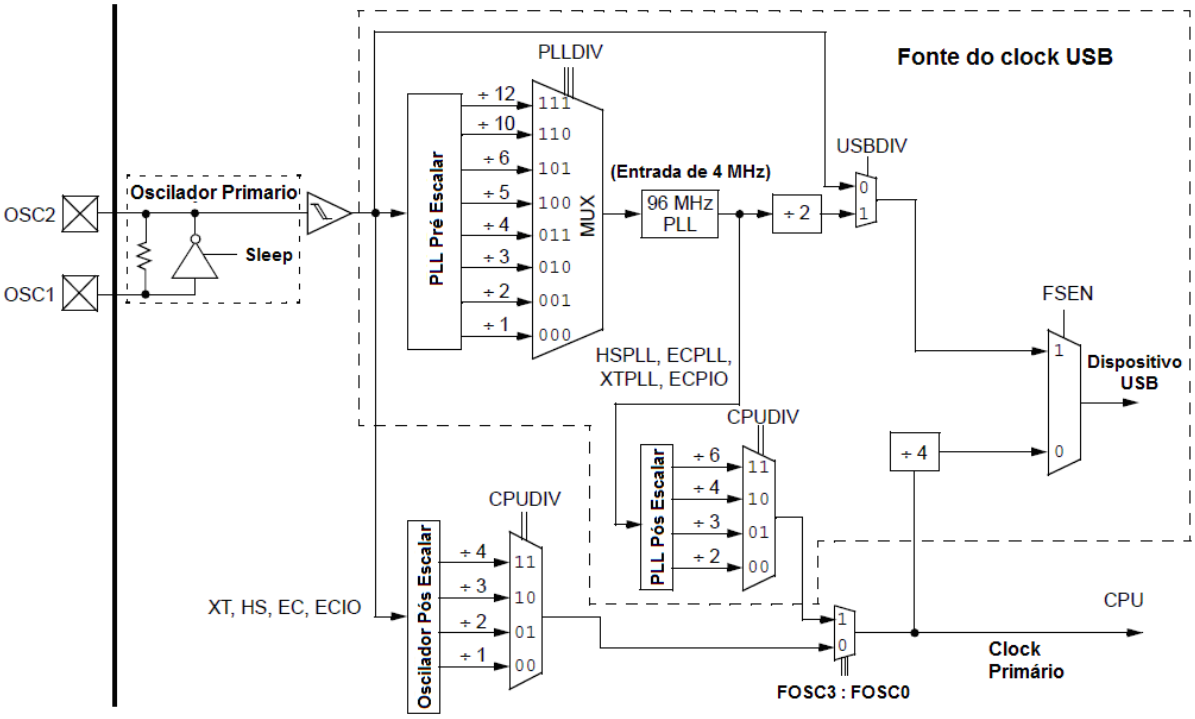

Figura 8. Diagrama do *clock* para o oscilador primário, adaptada e extraída de [3].

A respeito do circuito do gerador de frequência o microcontrolador divide os cristais osciladores em dois tipos possíveis, os cristais XT (de 200 kHz a 4 MHz) e os HS de alta velocidade (de 4MHz a 20 MHz). Os bits para a configuração da frequência são: XT (oscilador a cristal), XTPLL (oscilador a cristal com PLL habilitado), HS (oscilador a cristal de alta velocidade), HSPLL (oscilador a cristal de alta velocidade com PLL habilitado), EC (gerador de *clock* externo), ECIO (gerador de *clock* externo com saída no RA6), ECPLL (gerador de *clock* externo com PLL habilitado), ECPIO (gerador de *clock* externo com PLL habilitado e saída no RA6), entre outros.

Os valores possíveis para os escalares PLL são divididos em dois tipos: pré escalares e pós escalares. Os valores de pré escalares são chamados de PLLDIV e são PLL1, PLL2, PLL3, PLL4, PLL5, PLL6, PLL10 e PLL12. Os valores de pós escalares são chamados de CPUDIV e são CPUDIV1, CPUDIV2, CPUDIV3 e CPUDIV4.

No programa principal a configuração da frequência de operação é algo que deve ser observado com bastante cautela. No circuito o cristal opera em uma freqüência de 20MHz. Há no PIC18F4550 uma grande variação de possibilidades de *clock* de operação. O dispositivo USB deve ser configurado para operar com 6 MHz, caso esteja configurado para transmissão *Low Speed*, ou 48MHz, caso esteja configurado para transmissão *Full Speed,* o qual foi o caso implementado neste sistema. Os bits de configuração do oscilador foram definidos como HSPLL (oscilador a cristal de alta velocidade com PLL habilitado).

As possibilidades de configuração desses bits para um oscilador de 20MHz seguem a Tabela 10.

Tabela 10. Configuração da freqüência de *clock*

Freqüência do oscilador de entrada	**PLLDIV**	**Modo de *clock***	**CPUDIV**	**Freqüência do *clock* do microcontrolador**
20 MHz	÷5 (100)	HS, EC, ECIO	None (00)	20 MHz
			÷2 (01)	10 MHz
			÷3 (10)	6.67 MHz
			÷4 (11)	5 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz

Os valores de configuração atribuídos, isto é, usados neste sistema, são os bits USBDIV, PLL5, CPUDIV1. Isso significa que é usado o modulo PLL pré escalar com a configuração PPL5. Conforme a **Figura 8** o cálculo com o valor de freqüência de entrada 20 MHz é divido por 5 gerando uma frequência de 4 MHz. Este valor é o adequado para o módulo de geração de 96 MHz. Este sinal é encaminhado para dois módulos diferentes. O primeiro módulo é controlado pelo USBDIV, o qual divide o sinal por 2 gerando uma freqüência de 48 MHz para o módulo USB. O outro é

controlado pela PLL pós escalar. Este valor recebido pelo módulo PLL pós escalar é dividido por 2, conforme CPUDIV1 e é enviado para o núcleo do microcontrolador. Como foi observado o núcleo do microcontrolador e o periférico USB, ambos trabalham com a freqüência de 48 MHz.

4.4.4 Alimentação do Circuito

O sistema pode ser executado em dois modos de operação, gravação com o ICD2 [4] e execução de comunicação USB. A alimentação do sistema pode ser feita de duas origens dependendo do modo de operação. Em ambos os modos o circuito é alimentado com 5V contudo o esquema de alimentação determina que função o sistema deve desempenhar. Vale salientar que a alimentação depende da arquitetura de comunicação em que o sistema está conectado, já que são arquiteturas de comunicação deferentes. A alimentação do circuito em modo de gravação é fornecida através da conexão do conector RJ-12 com o gravador.

A alimentação do circuito no modo USB pode ser feitas através de três opções. A alimentação elétrica através do dispositivo conectado na USB, isto é, o computador fornece a tensão para o funcionamento do circuito. Pode ser feita também por meio de alimentação externa, através de pilhas, baterias ou adaptadores ligados na corrente elétrica fornecendo tensão diretamente ao circuito. A alimentação pode também ser hibrida, sendo composta por alimentação do dispositivo USB ou externa. Está última configuração permite que o circuito seja alimentado por fonte externa, porém na ausência ser alimentado pela conexão USB. Pode-se obter alternância entre a alimentação neste último caso.

Neste projeto foi usada a alimentação pela comunicação USB. Isso torna o circuito mais portátil e mais prático. Não é necessária a substituição de pilhas/baterias ou a preocupação de ter disponível tensão ao executar as funções do circuito. O esquema elétrico desta ligação segue a especificação da Figura 9.

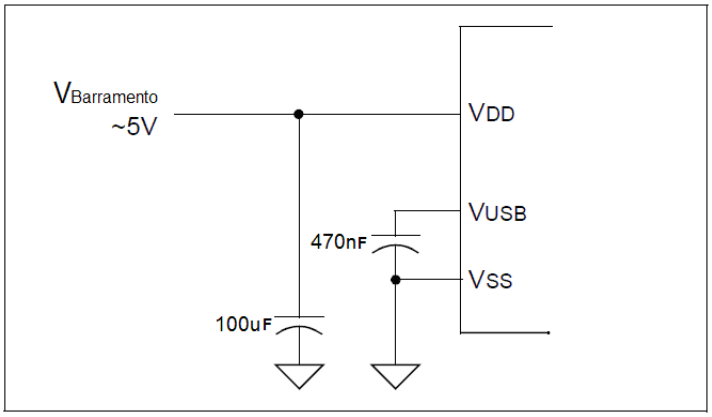

Figura 9. Esquema de alimentação do circuito

A alimentação do Vbarramento é fornecido pelo pino de alimentação USB. Um capacitor de 100uF é ligado entre a alimentação e o terra para que seja mantida uma estabilidade na tensão. De mesmo modo é feita a conexão do Vusb e o valor do capacitor neste caso é de 470nF. O Vss é o pino terra do microcontrolador.

Deve-se mencionar a existência do pino sensitivo. Este pino serve para monitoramento interno do microcontrolador a fim de detectar alguma conexão do circuito em uma porta USB. No sistema desenvolvido não existe necessidade da implementação do uso deste pino, pois a evidência de uma conexão é o seu funcionamento. Este pino somente é implementado nos casos de alimentação do circuito via fonte externa ou alimentação hibrida.

4.4.5 Circuito de Gravação

Um microcontrolador necessita de um sistema de gravação para inserir códigos em sua memória. Ao compilar um programa fonte em C é gerado um arquivo pronto para ser gravado, e é esse sistema de gravação que executa o processo da transferência do programa para o microcontrolador. Existem diversos gravadores no mercado e neste projeto é usado o ICD2 [4]. Este gravador é fabricado e comercializado pela Microchip. Ele é composto por duas partes, o gravador/depurador e o soquete de gravação para o microcontrolador. O gravador faz a conversão dos dados gerados pelo compilador em dados a serem armazenados no microcontrolador. Pode ser observado na Figura 10 um gravador ICD2. O soquete de gravação é o circuito que fornece a comunicação do gravador

com o microcontrolador a ser gravado. Cada microcontrolador tem uma configuração especifica no soquete de gravação.

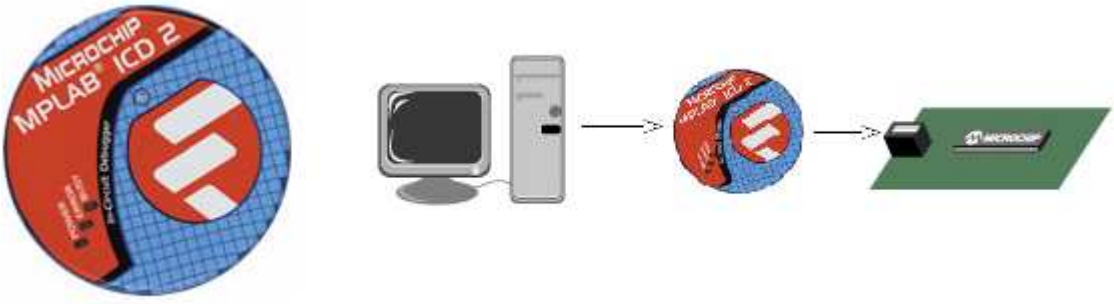

Figura 10. Gravador e esquema de gravação ICD2

Neste trabalho foi adicionado ao sistema de comunicação USB um circuito semelhante ao soquete de gravação. A implementação desse circuito traz a facilidade de gravação do programa no microcontrolador sem a necessidade da retirada do mesmo do circuito.

A seleção do modo de operação do sistema é feita através de uma chave seletora presente no circuito. Só deve ser usada uma das duas opções na execução, isto é, ou modo de operação de gravação ou execução de comunicação USB.

4.4.6 Programa Principal

O primeiro passo para o desenvolvimento do programa foi a definição das diretrizes de compilação iniciais. Essas são instruções para o compilador atribuir, na etapa de compilação, valores específicos de configuração de cada microcontrolador. Podem ser observadas algumas diretivas na Tabela 11.

Tabela 11. Diretivas de compilação

Diretiva	Descrição
#define	Definição de constantes de cadeia de caracteres
#include	Incluir arquivos de códigos fontes
#fuses	Define os bits de configuração
#use delay	Informa ao compilador o valor da freqüência em Hz

A diretiva *#define* é apropriada para fornecer um rotulo a um valor ou cadeia de caracteres que serão substituídos pelos respectivos valores no momento da

compilação. Ideal para a definição de constantes ou oferecer nomes a cadeia de caracteres ou valores. Sua principal vantagem é que não é necessária a alteração do valor em todas as ocorrências, apenas sua alteração é necessária em um ponto do código. Existe na implementação do sistema a definição dos comandos que podem ser enviados durante a execução. A definição segue o código abaixo.

```
#define INSTRUCOES 00
#define LEITURA 01
#define GRAVACAO 02
#define ROTINA 03
#define DESCONECTAR 04
```

A diretiva *#use delay* informa ao compilador qual o *clock* de operação da CPU do microcontrolador e os valores são informados em Hz. A operação deste sistema é baseada em uma freqüência de 48 MHz. No programa é definido conforme código abaixo.

```
#use delay(clock=48000000)
```

A diretiva *#include* indica arquivos com códigos de programação extras. Esses arquivos de código fonte são adicionados ao programa principal no momento da compilação. Evitam a repetição de um determinado código em vários arquivos. Funções especiais podem ser definidas em arquivos separados e adicionados em vários programas sem a necessidade de reescrevê-los. No programa desenvolvido foi usado está diretiva para a inclusão de dois arquivos, são eles: 18F4550.h e usb_biblioteca.h. As inclusão é definida conforme descrito abaixo.

```
#include <18F4550.h>
#include <usb_biblioteca.h>
```

O arquivo 18F4550.h é fornecido pelo fabricante e contém informações e funções aplicadas ao PIC 18F4550. Nesse arquivo são definidas funções de manipulação de temporizadores, entrada e saída de dados, configuração, comparadores, conversores analógico-digitais, interrupções, entre outras. É um arquivo essencial para o uso desse microcontrolador.

O arquivo usb_biblioteca.h é o arquivo desenvolvido neste trabalho que contem funções de comunicação USB. Esse arquivo promove transferência de

dados do tipo de dispositivo USB CDC. Ele inclui com a diretiva #include, o arquivo usb_cdc.c que contém mais funções para esse tipo de comunicação. O arquivo usb_cdc.h é um arquivo de suporte do compilador CCS. No arquivo USB_cdc.c é incluído o arquivo de descritores, o USB_desc_TCC.

O arquivo usb_desc_TCC.h contém a implementação dos descritores USB. Os descritores deste sistema foram detalhados nas seções anteriores. Este arquivo contem às informações que descrevem detalhadamente o dispositivo USB implementado.

A diretiva #fuses informa ao compilador os bits de configuração adicionados ao programa na compilação e, conseqüentemente, na gravação. Estes bits são específicos de cada microcontrolador, apesar de alguns serem iguais em alguns tipos de microcontroladores.

Esses bits são chamados pelo compilador CCS de fusíveis de configuração. Os bits de configuração usados podem ser visualizados na Tabela 12 incluindo uma breve descrição do significado. Maiores detalhes sobre esses parâmetros foram expressos em seções anteriores.

Tabela 12. Bits de configuração do programa principal

Bits de Configuração	Descrição
HSPLL	Cristal oscilador de alta velocidade com PLL habilitado
NOWDT	Desativação do cão de guarda
NOPROTECT	Desativação de proteção de código
NOLVP	Desativação da gravação em baixa tensão
NODEBUG	Desativação de depuração em circuito
USBDIV	Ativação do multiplexador USBDIV
PLL5	Uso de PLL pré escalar com valor 5
PUT	Ativação do *Power-Up Timer*
MCLR	Ativação da re-inicialização externa
CPUDIV1	Uso de PLL pós escalar com valor 1
VREGEN	Ativação do uso do Vusb como regulador de tensão interna USB

O programa oferece cinco opções, são elas: Instruções, Ler EEPROM, Gravar na EEPROM, Mostrar a Rotina e Desconectar dispositivo.

O comando “Instruções” fornece todas as opções que o usuário poderá escolher. O comando enviado ao microcontrolador será em hexadecimal e seu valor

é 00. Nessa opção o microcontrolador enviará uma string armazenada na sua memória com todas as opções possíveis que podem ser enviadas. Este comando usa a função usb_cdc_putString() presente na biblioteca para enviar a mensagem com as instruções conforme código em C abaixo:

```
char mensagem1[] = {"
\nSistema de Comunicação USB com Microcontrolador
TCC - 2009.1 - DSC / POLI / UPE
Aluno: Leonardo Santos
Orientador: Sergio Campello

Opções:
00 -> Instruções
01 -> Ler EEPROM
02 -> Gravar na EEPROM
03 -> Mostrar Rotina
04 -> Desconectar\n"};

// Envia uma string para o computador
usb_cdc_putString(mensagem1);
```

O comando “Ler EEPROM” mostra o conteúdo armazenado na memória do microcontrolador da posição 00 até a 2F. Os valores são organizados por linha e coluna. Onde na primeira linha são mostrados os valores da memória das posições 00 até a 0F, na próxima linha serão mostrados os valores das posições 10 até 1F, e assim sucessivamente até a ultima posição. Esses valores foram organizados dessa forma para facilitar a visualização das alterações dos valores com o comando “Gravar na EEPROM”. O comando é representado no microcontrolador como o valor em hexadecimal 01. A chamada para a execução deste comando usa a função usb_cdc_putEEPROM() desenvolvida na biblioteca de funções para transmitir dados da memória do microcontrolador para o computador.

O comando “Gravar na EEPROM” é usado para alteração da memória de dados. Com ele o usuário pode transmitir valores para serem armazenados na memória do microcontrolador. Ao executar esse comando o programa solicitará uma posição de memória e em seguida um valor em hexadecimal. Cada posição de memória pode armazenar dois bytes, caso seja fornecido um valor que não possa

ser gravado nesta quantidade de bytes, o valor será truncado. A sua representação no microcontrolador é o hexadecimal 02. Os valores são recebidos com o uso da função gethex_usb() descrita na biblioteca.

O comando “Mostrar Rotina” informa o número do Contador de Rotinas. O Contador de Rotinas é uma variável inteira longa que é incrementada enquanto o dispositivo está em funcionamento. Ela foi criada com a intenção de simular um trabalho qualquer que o microcontrolador ficará executando durante seu funcionamento. Este contador pode ser substituído por qualquer rotina continua em trabalhos futuros. A representação desse comando é o hexadecimal 03 no microcontrolador.

O comando “Desconectar” prepara o dispositivo para ser efetivamente desconectado do computador. Ele encerra a comunicação e o funcionamento do programa principal. Ao executar este comando o dispositivo pode ser removido com segurança do computador. O valor hexadecimal que representa este comando é 04 no microcontrolador. A desconexão é feita pela função usb_detach() descrita na biblioteca.

Existe um fluxo no programa que mostra uma mensagem ao usuário caso ele forneça algum comando não definido. O microcontrolador informa ao usuário a mensagem “Digite um valor válido!” conforme o trecho de código abaixo. Este fluxo funciona como filtro para não serem inseridos valores que provoquem um funcionamento não esperado.

```
char mensagem6[] = {"\n\nDigite um valor válido !\n"};
usb_cdc_putString(mensagem6);
```

O desenvolvimento do programa segue alguns passos. Inicialmente há uma definição das variáveis de uso no programa,em seguida há a inicialização do dispositivo USB através da função usb_cdc_conectar(),. Esta função está definida na biblioteca criada neste trabalho e descrita em seções anteriores. A seguir, o programa entra em um loop onde verifica se o dispositivo USB está devidamente configurado. Ainda neste loop é executado o incremento do Contador de Rotinas. Há uma verificação a cada loop se existe algum dado enviado ao microcontrolador, caso isso aconteça, indica que foi enviada algum comando, estão o programa recebe

esse dado e verifica que comando foi acionado e executa a devida tarefa. O comando que encerra a execução do programa é o de Desconectar.

4.4.7 Simulação

A simulação é um bom tipo de análise do funcionamento do dispositivo, pois é possível a verificação da implementação de forma virtual. O aplicativo usado foi o Proteus 7.4 com o módulo virtual USB. O circuito é dividido em seis partes, são elas: a gravação com o ICD2, o circuito selecionador de função, o circuito oscilador, as luzes informativas, o circuito de re-inicialização e a comunicação USB.

A interligação entre o computador e o gravador ICD2 é feita pela comunicação USB ou por porta RS-232. A conexão entre o gravador e o circuito de gravação é realizada através do conector RJ-12. Este circuito de gravação foi incluído no sistema, não existindo assim a necessidade da retirada do microprocessador do circuito para a gravação. Esse desenvolvimento proporciona todas as características do circuito original de gravação como, por exemplo, a depuração em tempo real.

A simulação de gravação com o ICD2 não é possível porque o Proteus não tem suporte já que é uma comunicação entre o gravador e o sistema. Contudo foi incluída no circuito de simulação para visualização dos componentes presentes e sua interligação com o circuito completo. A falta dos conectores na documentação suporte do Proteus foi suprida com a inclusão de pinos de entrada simples, que representam o conector RJ-12 usado na comunicação com o gravador ICD2. Podem-se observar todas as conexões entre este módulo e o microcontrolador na Figura 11.

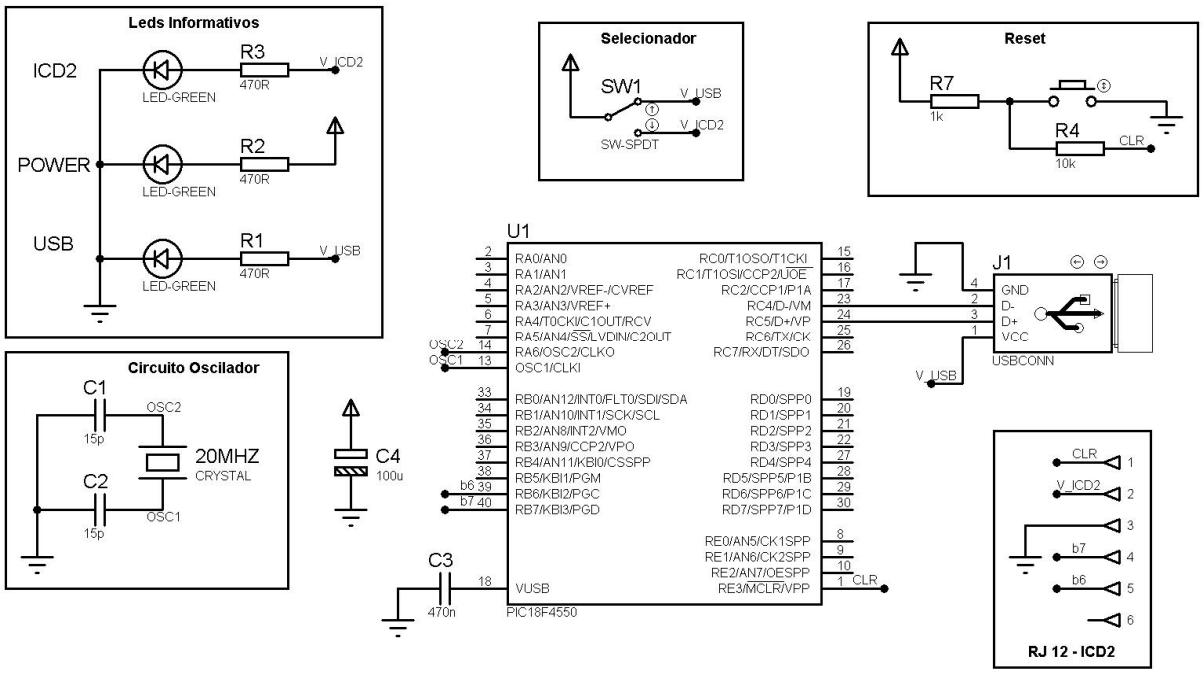

Figura 11. Circuito de simulação no Proteus

O circuito pode ser executado operando com duas funções. A primeira é como sistema de comunicação USB e a outra é como circuito de gravação. O Circuito Selecionador funciona como um seletor para definir de que modo o circuito será executado. É nessa parte do circuito que é controlada a alimentação, que depende do modo de operação do mesmo. Está chave tem duas posições, USB e ICD2. Podem ser observadas suas conexões na Figura 11.

O circuito oscilador é o responsável por gerar o *clock* de entrada no circuito. O circuito usa um cristal de 20 MHz e dois capacitores cerâmicos de 15pF. O microcontrolador internamente configura o *clock* conforme descrito em seções anteriores. Este circuito está presente na simulação para verificar sua integração com todo circuito, pois a sua simulação não é possível. A definição da freqüência é informada diretamente no Proteus.

As luzes informativas são para monitoração do funcionamento do circuito. Estão presentes três luzes indicadoras dos estados do circuito, são elas: circuito em execução, USB ativado e gravação ativada. A luz que indica quando o dispositivo está em funcionamento verifica se o sistema está em operação em algum dos dois modos. A luz de USB ativada indica quando o sistema está sendo executado como

comunicação USB. A luz de gravação ativada indica quando o circuito de gravação está sendo executado. A distinção destes tipos de operação é dada pela alimentação, portanto não é recomendada a operação de ambos os circuitos ao mesmo tempo, sendo possível a danificação do sistema.

O circuito de re-inicialização é responsável por provocar uma re-inicialização do sistema completo por meio externo. Caso exista um mau funcionamento do circuito existe esta opção de restauração.

Existe no circuito a presença de dois capacitores não inclusos em nenhum dos módulos definidos anteriormente. O primeiro se encontra entre uma ligação do Vdd com o Vss do circuito. Esse funciona como um filtro para a retirada dos ruídos de alta freqüência da alimentação do circuito. O segundo se encontra entre uma ligação do Vusb e o Vss. Esse é uma recomendação do fabricante do dispositivo para manter a estabilidade do circuito USB interno do microcontrolador.

4.4.8 Circuito Elétrico

Na simulação não é possível verificar algumas características como tensão no microcontrolador, configuração do oscilador e comunicação com tipos de comunicação não suportados. O circuito elétrico envolve todas as conexões, simuladas ou não. A estrutura da construção física do circuito varia conforme disponibilidade de espaço, confecção de trilhas, colocação de componentes e conectores.

Para a sua confecção foram seguidas todas as recomendações do fabricante do microcontrolador em relação aos valores de componentes utilizados. Neste circuito desenvolvido não é necessária alimentação via barramento, como foi especificado em seções anteriores. O desenho da placa de circuito impresso é demonstrado na Figura 12.

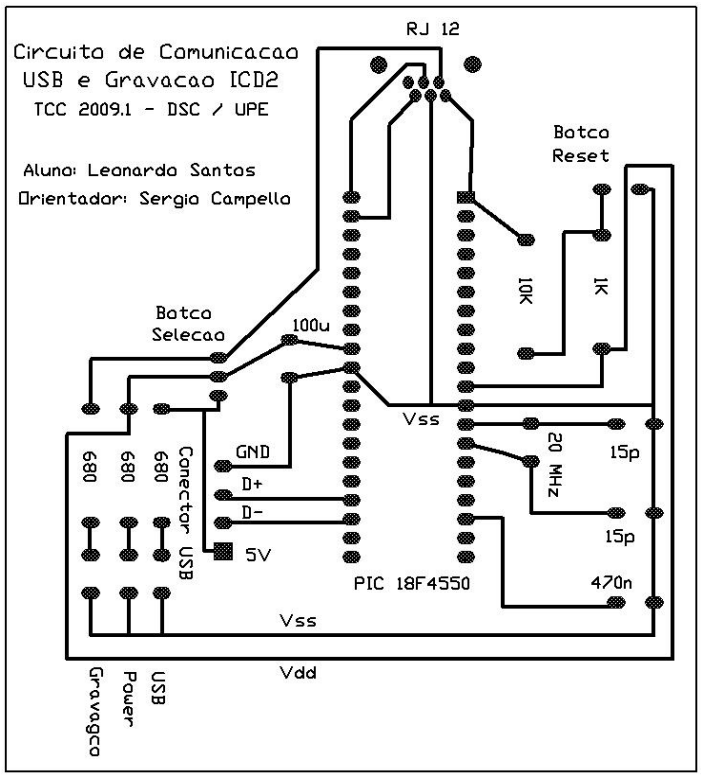

Figura 12. Desenho do circuito impresso

O circuito construído pode ser visualizado na Figura 13 e na Figura 14. Seu funcionamento e instalação são detalhados nas seções posteriores.

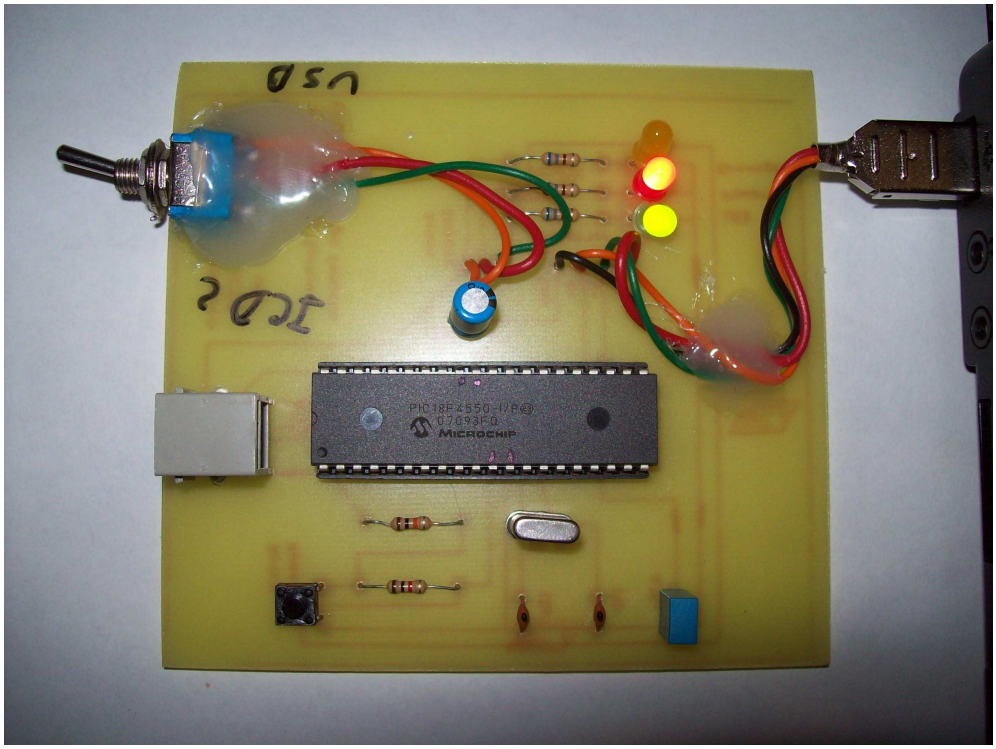

Figura 13. Face dos componentes do circuito construído

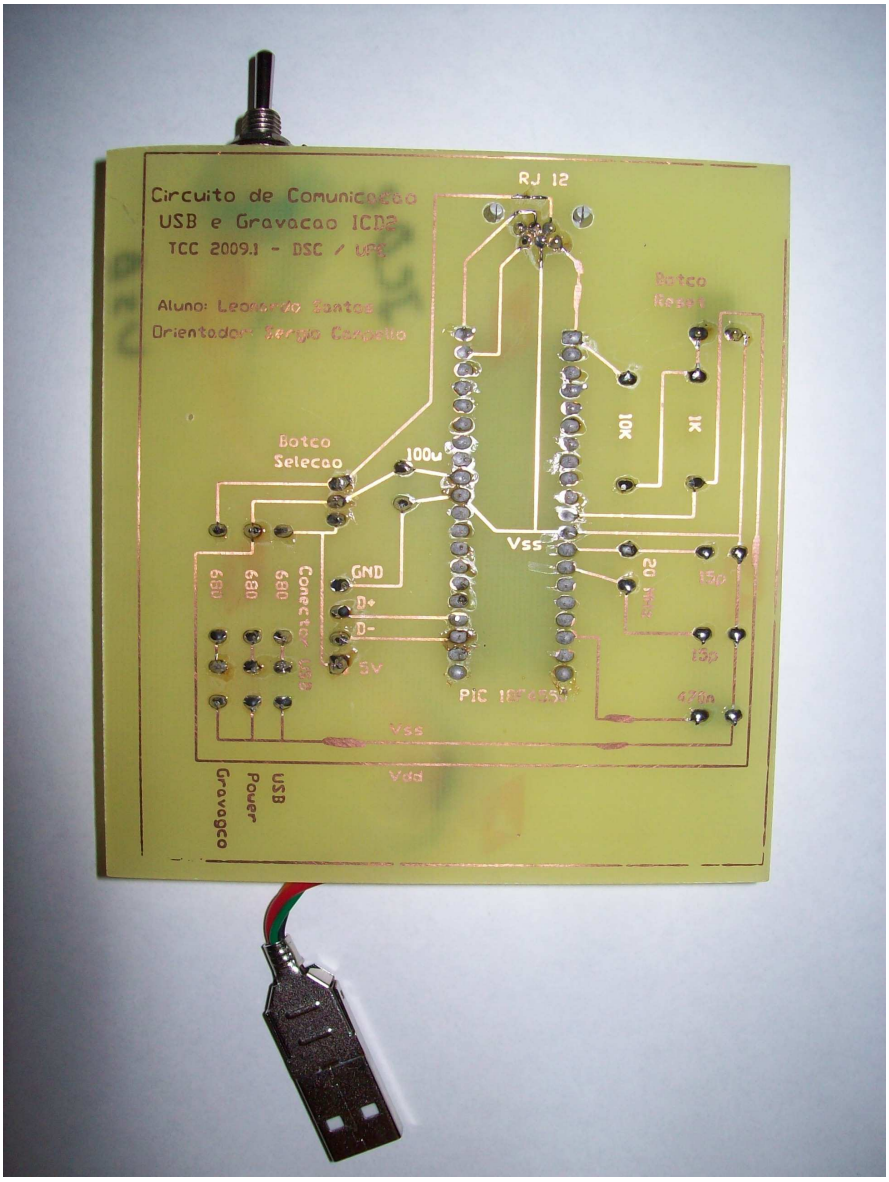

Figura 14. Face das soldas e trilhas do circuito construído

4.4.9 Instalação e Funcionamento

A instalação do sistema é dada de modo simples como costumam ser os dispositivos de comunicação USB. Após o sistema ser totalmente construído, isto é, em circuito impresso, com os componentes e o programa gravado no microcontrolador, é necessário iniciar a instalação no computador. Quando o sistema foi conectado ao computador é reconhecido como TCC – LEO SANTOS – SCUSB conforme Figura 15.

Figura 15. Reconhecimento no computador do SCUSB (Sistema de Comunicação USB)

A seguir é interrogado quanto ao *driver* de instalação. O *driver* é um arquivo fornecido pela instalação do CCS C *Compiler* que se encontra na pasta de instalação \...\PICC\Drivers e são os arquivos cdc9Xpt1.inf, cdc9Xpt2.inf, cdc_NTXP.inf e cdc_NTXPVista.inf, respectivamente para os sistemas operacionais Windowns 9xpt1, 9xpt2, NT/XP e Vista. No termino da instalação é possível fazer a comunicação USB com o sistema. Ao conectar o dispositivo e concluir a instalação é adicionada uma nova porta serial ao computador.

O programa utilizado é o *Advanced Serial Port Terminal*. Neste programa é possível abrir um fluxo de comunicação. No *Advanced Serial Port Terminal* deve-se criar uma nova seção, escolhendo a porta, a taxa e demais configurações. Os parâmetros seguem os representados na Figura 16. A única mudança será a porta, pois depende da porta em que foi instalado o sistema. Nesse exemplo a porta usada foi a COM21. Configurada a conexão, deve-se abrir a porta de comunicação para iniciar o fluxo de dados conforme Figura 17.

O microcontrolador no momento da configuração já está executando o Contador de Rotinas e aguarda o recebimento de algum comando.

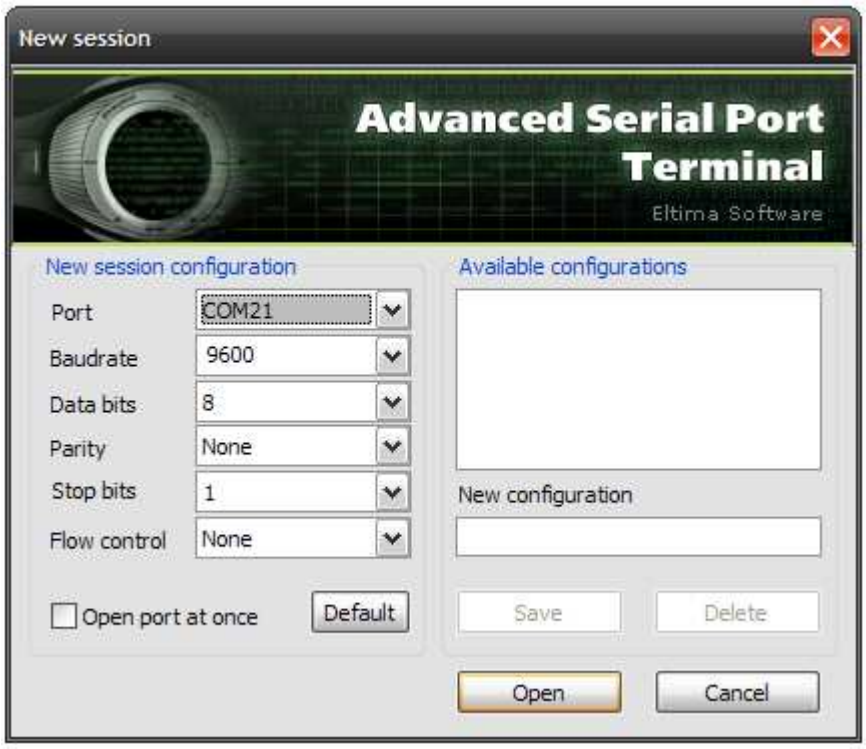

Figura 16. Configuração da conexão do terminal no computador

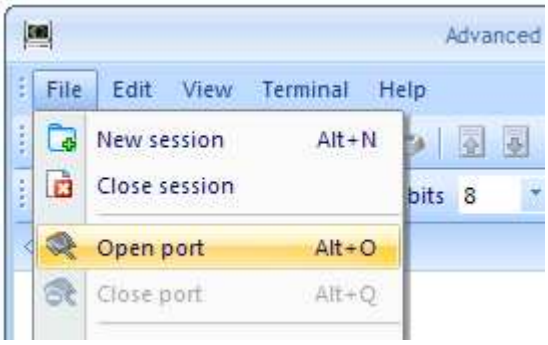

Figura 17. Abrir porta de comunicação no programa terminal

Os comandos são os valores “00”, “01”, “02”, “03”, “04” que representam respectivamente Instruções, Ler EEPROM, Gravar na EEPROM, Mostrar Rotina, Desconectar. A resposta dos comandos Instruções, Ler EEPROM, Gravar na EEPROM, e Mostrar Rotina podem ser observados respectivamente na Figura 18, Figura 19, Figura 20 e Figura 21.

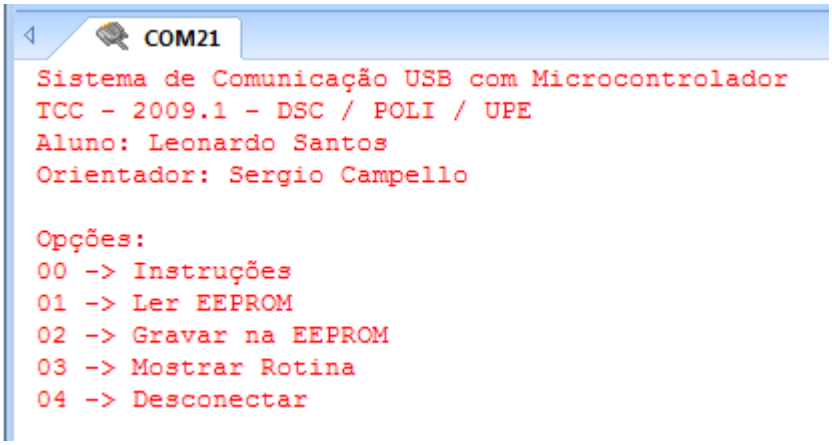

Figura 18. Resposta obtida pelo envio do comando Instruções

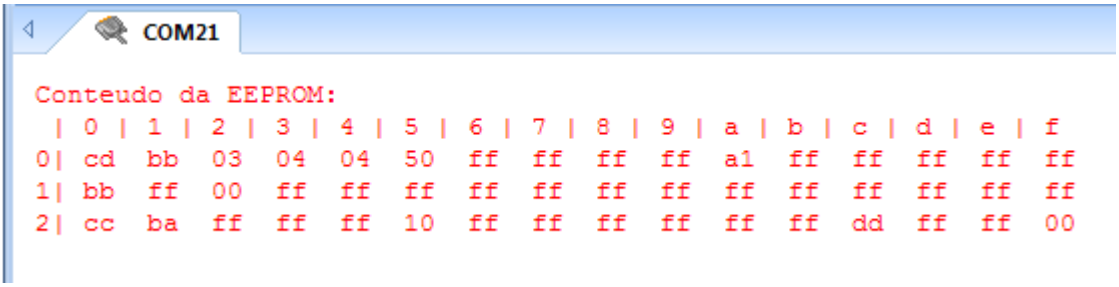

Figura 19. Resposta obtida pelo envio do comando Ler EEPROM

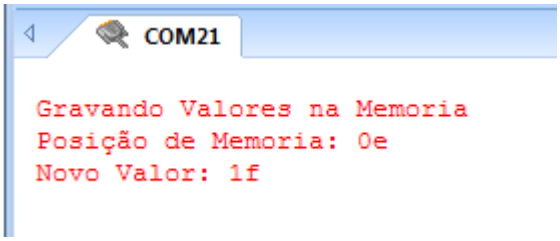

Figura 20. Resposta obtida pelo envio do comando Gravar na EEPROM

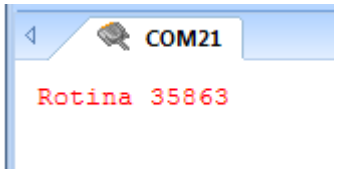

Figura 21. Resposta obtida pelo envio do comando Mostrar Rotina

Capítulo 5

Conclusão e Trabalhos Futuros

Atualmente a comunicação entre computadores e sistemas que usam microcontroladores é de extrema importância para uma grande variedade de projetos. Existe em algumas aplicações a necessidade de colher informações, prover comunicação e enviar dados para esses sistemas. Há sempre também uma busca por simplicidade e abstração quando se usa comunicação nestes sistemas. O protocolo de comunicação USB traz uma boa parte destas vantagens, por isso é de suma importância o seu desenvolvimento em sistemas que necessitem comunicação com os computadores.

A comunicação USB além de ser de fácil configuração, *Plug and Play* e presente na maioria dos computadores atuais, tem várias outras vantagens como a alimentação direta no circuito que é usado neste sistema. Este trabalho fornece abstração de seu funcionamento a baixo nível tornando transparente ao desenvolvedor a comunicação USB.

O desenvolvimento deste trabalho proporciona uma fácil implementação do tipo de comunicação USB em microcontroladores. Isso facilita a disseminação do uso de comunicação com esse tipo de dispositivo. Assim como proporciona maior adaptabilidade de aplicações nesta área já que diversos tipos de comunicação têm entrado em desuso devido às novas vantagens oferecidas com estas formas de comunicação.

Este trabalho proporciona como desenvolvimentos futuros a implementação de uma interface própria no computador para realizar a comunicação com o dispositivo que contém o microcontrolador. Bibliotecas são disponibilizadas pelos fabricantes de forma que possa ser desenvolvida uma interface de comunicação no computador mais específica para cada aplicação.

Vale salientar que este trabalho fornece privilégios a implementação deste tipo de comunicação em qualquer sistema. E que no futuro seja integrado com

sistemas de uso específicos. No inicio deste trabalho foi observada a aplicação deste projeto no sistema de detecção óptica de descargas parciais em cadeias de isoladores de transmissão de alta tensão [1], contudo a generalização para alcançar varias aplicações se tornou um ponto forte.

O tipo de dispositivo USB CDC obteve um ótimo desempenho em relação à fácil instalação e facilidade de comunicação, contudo o desenvolvimento de comunicação USB usando outros tipos de dispositivos como *mass storage device* pode ser também tratado em trabalhos futuros como forma de aprimorar ou comparar as vantagens e desvantagens desses outros tipos de comunicação em relação ao que foi desenvolvido.

Bibliografia

[1] S. C. Oliveira, “Sistema de Detecção Óptica de Descargas Parciais em Cadeias de Isoladores de Transmissão de Alta Tensão”. 119 f. Tese de Doutorado, Depto. de Engenharia Elétrica da Universidade Federal de Pernambuco, Recife – PE, Brasil, 2008.

[2] *Data Sheet* do microcontrolador PIC16F87A disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf. Acessado em 08/06/2009

[3] *Data Sheet* do microcontrolador PIC18F4550 disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/39632b.pdf. Acessado em 08/06/2009

[4] *Microchip Technology Inc,* disponível em http://www.microchip.com. Acessado em 08/06/2009.

[5] STALLINGS, William. Arquitetura e Organização de Computadores 5a Edição. São Paulo: Prentice Hall, 2002.

[6] USB.Org, disponível em http://www.usb.org/. Acessado em 08/06/2009.

[7] Ibrahim, Dagan. *Advanced PIC microcontroller Projects in C: from USB to RTOS with the PIC18F series*, Newnes, 2008.

[8] CCS, Inc. – Home, disponível em http://www.ccsinfo.com/. Acessado em 08/06/2009.

[9] *Serial Port Terminal – Test and debug drial port devices with advanced serial port terminal*, disponível em http://www.eltima.com/products/serial-port-terminal. Acessado em 08/06/2009.

Apêndices

Arquivo usb_serial_TCC.h

```
// Inclue a biblioteca basica do PIC 18F4550
#include <18F4550.h>

// Atribui os Bits de configuração e define o clock do nucleo do microcontrolador
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN,MCLR,PUT
#use delay(clock=48000000)

// Inclue a biblioteca de funções
#include <usb_biblioteca.h>

#define INSTRUCOES 00
#define LEITURA 01
#define GRAVACAO 02
#define ROTINA 03
#define DESCONECTAR 04

void main()
{
  // Declaração de variaveis
  int address, value, opcao;
  unsigned long int cont = 0;

  // Inicializa, configura e enumera o dispositivo USB
  usb_cdc_conectar();

  do
  {
    if (usb_enumerated()) // Verifica se o dispositivo está pronto
```

```
{
  // Simula uma rotina de um processo qualquer usual de microcontrolador
  cont++;
  // Verifica se existe caracter para recepção
  if(usb_cdc_kbhit())
  {
    // Recebe um hexadecimal
    opcao = gethex_usb();
    // As opções são 00, 01, 02, 03 e 04
    switch (opcao)
    {
      case INSTRUCOES:
      {
        char mensagem1[] = {"
\nSistema de Comunicação USB com Microcontrolador
TCC - 2009.1 - DSC / POLI / UPE
Aluno: Leonardo Santos
Orientador: Sergio Campello

Opções:
00 -> Instruções
01 -> Ler EEPROM
02 -> Gravar na EEPROM
03 -> Mostrar Rotina
04 -> Desconectar\n"};

        // Envia uma string para o computador
        usb_cdc_putString(mensagem1);
        break;
      }
      case LEITURA:
      {
        char mensagem2[] = {"\n\nConteudo da EEPROM:\n"};
        usb_cdc_putString(mensagem2);
        printf(usb_cdc_putc,  " | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f");
```

```
    printf(usb_cdc_putc,  "\n0| ");

    // Envia um bloco da memoria de dados do microcontrolador
    usb_cdc_putEEPROM (0x00, 15);
    printf(usb_cdc_putc,  "\n1| ");
    usb_cdc_putEEPROM (0x10, 15);
    printf(usb_cdc_putc,  "\n2| ");
    usb_cdc_putEEPROM (0x20, 15);
    printf(usb_cdc_putc,  "\n");
    break;
  }
  case GRAVACAO:
  {
    char mensagem3[] = {"\n\nGravando Valores na Memoria"};
    char mensagem4[] = {"\nPosição de Memoria: "};
    char mensagem5[] = {"\nNovo Valor: "};
    usb_cdc_putString(mensagem3);
    usb_cdc_putString(mensagem4);
    address = gethex_usb();
    usb_cdc_putString(mensagem5);
    value = gethex_usb();
    // Grava na memoria do microcontrolador
    write_eeprom( address, value );
    printf(usb_cdc_putc,  "\n");
    break;
  }
  case ROTINA:
  {
    printf(usb_cdc_putc, "\n\nRotina %lu", cont);
    printf(usb_cdc_putc,  "\n");
    break;
  }
  case DESCONECTAR:
  {
    // Desconecta o dispositivo
```

```
            usb_detach();
            break;
          }
          default:
          {
            char mensagem6[] = {"\n\nDigite um valor valido !\n"};
            usb_cdc_putString(mensagem6);
            break;
          }
        }
      }
    }
  }
  while (TRUE);
}
```

Arquivo usb_biblioteca.h

```
/*
**********   usb_biblioteca.h   **********
```

Esta é uma biblioteca para transmissão de dados através do protocolo de USB de comunicação. Contém funções básicas de transmissão e recepção de dados. É implementada para simular uma PORTA COM VIRTUAL.

Modelo do microprocessador: PIC 18F4550

Compilador: CCS 4.057

TCC 2009.1 - DSC / UPE

Aluno: Leonardo de Sá Leal Santos

Orientador: Sergio Campello

** Instruções **

* Inclusão de um descritor CDC diferente

A inclusão de descritores é feita no arquivo usb_cdc.h que está localizado na pasta "\...\PICC\Drivers", a deve-se trocar o arquivo usb_desc_cdc.h pelo nome do novo descritor caso deseje trocá-lo.

* Criação de um novo arquivo descritor usb

Caso deseje efetuar alguma modificação no descritor, o mesmo se encontra na pasta "\...\PICC\Drivers" com o nome usb_desc_cdc.h. Faça as modificações e salve na mesma pasta onde se encontra o projeto, não esquecendo de adicioná-lo nos arquivos de configuração.

*/

```
#include <usb_cdc.h>

/* ** usb_cdc_putString **
Envia uma string
  char *p -> ponteiro que represente a string

  obs: A string deve estar declarada para ser enviada, por exemplo:

  char temp[] = {"valor da string"};
  usb_cdc_putString(temp);

  Há uma forma alternativa para o envio de string e com ela não é necessário
  a declaração da string anteriormente.

  printf(usb_cdc_putc, "valor da string");   */

void usb_cdc_putString(char *p)
{
    char i = 0;
    while (*(p+i) != '\0')
    {
        usb_cdc_putc(*(p+i));
```

```
        i ++;
    }
}

/* ** usb_cdc_putEEPROM **
Envia um bloco da memória de dados
  int posicaoI -> Posição inicial do bloco de memória
  int tamanho  -> Comprimento do bloco de memória
  Obs: Pode enviar os valores em hex ou em caracteres ASCII (código comentado)*/

void usb_cdc_putEEPROM (int posicaoI, int tamanho)
{
   int i;

   for(i=0; i<=tamanho; i++)
   {
      printf(usb_cdc_putc, "%2x  ", read_eeprom(i + posicaoI) );
      //usb_cdc_putc(read_eeprom(i + posicaoI));
   }
}

/* ** usb_cdc_putFLASH **
Envia um bloco da memória de programa
  int posicaoI -> Posição inicial do bloco de memória
  int tamanho  -> Comprimento do bloco de memória
  Obs: Pode enviar os valores em hex ou em caracteres ASCII (código comentado)*/

void usb_cdc_putFLASH (char posicaoI, char tamanho)
{
   char i;
   int16 dado;

   for(i=0; i<=tamanho; i++)
   {
      dado = read_program_eeprom(i + posicaoI);
```

```
      usb_cdc_putc(dado);
   }
}

/* ** usb_cdc_conectar **
Inicializa, configura e enumera o dispositivo USB
  Obs: Deve ser usado quando for conectado o dispositivo USB no
  computador para efetuar a transmissão    */

void usb_cdc_conectar()
{
//   usb_cdc_init();
   usb_init_cs();// não trava na configuração e é necessário chamar usb_task
   usb_init();
   usb_task();
   while(!usb_cdc_connected()) {}
}

/* *****    Funções nativas     *****

 ** usb_enumerated() **
Verifica se o dispositivo está pronto para a comunicação
Retorna True ou False

          Obs: Maiores informações sobre estas funções e mais funções
   estão no arquivo usb.c em \...\PICC\Drivers


 ** usb_detach() **
Desconecta o dispositivo, usado antes de sua remoção física do computador

 ** usb_attach() **
Re-conecta o dispositivo, usado para re-conectá-lo
quando o dispositivo foi desconectado mas ainda não removido literalmente
```

Obs: Maiores informações sobre estas funções e mais funções estão no arquivo pic18_usb.h em \...\PICC\Drivers

** usb_cdc_putc(char c) **
Envia um caracter para a transmissão

** usb_cdc_kbhit() **
Verifica se existe algum dado no buffer de recepção
Retorna True ou False

** usb_cdc_getc() **
Recebe um caracter do buffer de recepção, deve-se usar o usb_cdc_kbhit() anteriormente para verificar se existe dados

** get_float_usb() **
Recebe um numero ponto flutuante

** get_long_usb() **
Recebe um numero longo

** get_int_usb() **
Recebe um inteiro

** get_string_usb(char *s, int max) **
Recebe uma String

** gethex_usb() **
Recebe um Hexadecimal

obs: Maiores informações sobre estas funções e mais funções
estão no arquivo usb_cdc.h em \...\PICC\Drivers */

Arquivo usb_desc_TCC.h

```
#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

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

  const char USB_CONFIG_DESC[] = {

  //Descritor de configuração 1

      USB_DESC_CONFIG_LEN, //length of descriptor size        ==0
      USB_DESC_CONFIG_TYPE,        //constant        CONFIGURATION
(CONFIGURATION 0x02)    ==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, // significa = (11000000) -> 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)   (0x32  =
100mA)  ==8

  //Descritor de interface 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


  //Descritor de interface 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) &
0xFF, //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
    };


    #define USB_MAX_NUM_INTERFACES   2
    const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={2};
    const                                                          int16
USB_CLASS_DESCRIPTORS[USB_NUM_CONFIGURATIONS][USB_MAX_NUM_I
NTERFACES][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


//   Descritor de Dispositivo


    const char USB_DEVICE_DESC[USB_DESC_DEVICE_LEN] ={
        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
        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
  };

//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)
                'T',0,
        'C',0,
        'C',0,
  //string 2
        50, //length of string index
        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
```

```
                'T',0,
        'C',0,
        'C',0,
        ' ',0,
        '-',0,
        ' ',0,
        'L',0,
        'E',0,
        'O',0,
        ' ',0,
        'S',0,
        'A',0,
        'N',0,
        'T',0,
        'O',0,
        'S',0,
        ' ',0,
        '-',0,
        ' ',0,
        'S',0,
        'C',0,
        'U',0,
        'S',0,
        'B',0
};

#ENDIF
```